Lecture Notes in Computer Science 2467

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Bruce Christianson Bruno Crispo
James A. Malcolm Michael Roe (Eds.)

Security Protocols

9th International Workshop
Cambridge, UK, April 25-27, 2001
Revised Papers

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Bruce Christianson
James A. Malcolm
University of Hertfordshire, Department of Computer Science
Faculty of Engineering and Information Sciences
College Lane, Hatfield, Herts, AL10 9AB, UK
E-mail: {b.christianson/J.A.Malcolm}@herts.ac.uk

Bruno Crispo
Vrije Universiteit, Department of Computer Science
De Boelelaan 1081A, 1081 HV Amsterdam, The Netherlands
E-mail: crispo@cs.vu.nl

Michael Roe
Microsoft Research Limited
7 J J Thomson Avenue, Cambridge, CB3 0FB, UK
E-mail: mroe@microsoft.com

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Security protocols : 9th international workshop, Cambridge, UK, April 25 - 27, 2001 ; revised papers / Bruce Christianson ... (ed.). - Berlin ; Heidelberg ; New York ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2002
(Lecture notes in computer science ; Vol. 2467)
ISBN 3-540-44263-4

CR Subject Classification (1998): E.3, F.2.1-2, C.2, K.6.5, J.1, K.4.1, D.4.6

ISSN 0302-9743
ISBN 3-540-44263-4 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN 10871275 06/3142 5 4 3 2 1 0

Preface

Hello and welcome. These are the proceedings of the 9th International Workshop on Security Protocols, the first to be held in the new millennium. This year our theme was "mobile computing versus immobile security". As usual, the insights and challenges which emerged during the workshop are reflected in the position papers, which appear here in rewritten form.

Transcripts are also included of the discussions which took place in Cambridge as the initial versions were presented. These transcripts are intended to provide a perspective on lines of argument which are worth pursuing further. Our desire is that you will join with us in this activity, and that as a result you will, like many of our participants, feel moved to propound something quite different from what you originally planned.

Our thanks as always to Prof. Roger Needham, FRS and to Microsoft Research Ltd. (Cambridge) for the use of the meeting room and coffee machine. Thanks also to Lori Klimaszewska of the University of Cambridge Computing Service for transcribing the audio tapes (and for revealing in "Audrey James" a previously unsuspected double life of a well-known double agent), and to Dr. Mary Buchanan for her assistance in editing the transcripts into a Thucydidean mould.

Actually, we are often asked how we go about producing the transcripts, especially upon those occasions when, for various reasons, no audio recording was made. This year we bow to pressure and reveal the details of our methodology in the Afterword.

We hope you find these proceedings enjoyable, infuriating, and fruitful. And we look forward to hearing from you.

Christmas Eve 2001

Bruce Christianson
Bruno Crispo
James Malcolm
Michael Roe

Previous Proceedings in This Series

The proceedings of previous International Workshops on Security Protocols were also published by Springer-Verlag as Lecture Notes in Computer Science, and are occasionally referred to in the text:

8th Workshop (2000), LNCS 2133, ISBN 3-540-42566-7
7th Workshop (1999), LNCS 1796, ISBN 3-540-67381-4
6th Workshop (1998), LNCS 1550, ISBN 3-540-65663-4
5th Workshop (1997), LNCS 1361, ISBN 3-540-64040-1
4th Workshop (1996), LNCS 1189, ISBN 3-540-63494-5

Table of Contents

Keynote Address: Mobile Computing versus Immobile Security

Roger Needham

Microsoft Research

The theme set down is "mobile computing versus immobile security". I think I know what mobile computing is, but I don't know what immobile security is.

Michael Roe: The idea behind that somewhat provocative title is that existing security protocols assume that nothing moves. The allowed assumption is that the security problem is just about solved for things that don't move.

Reply: I think that is an optimistic assumption myself, but there is no doubt that mobility makes things more difficult. An issue that's been rising in my consciousness over the last year or so is that although one may have protocols for things like key exchange and the authentication of individuals, and one may have some confidence that these protocols are reasonably OK, they don't take you very far down road to the goal of achieving some good quality security. The reason they don't is that security depends upon a great deal and more than knowing who somebody is. It depends on all sorts of metadata like access control lists, statements of who is permitted to do what, statements of who is permitted to adopt what rôle, statements of who is permitted to change that sort of metadata and — even in static systems — you find that all that data is not in the same place. Usually nobody knows where or what it all is, and even if you could bring it all in the same place, you wouldn't be able to do a computation on it which would say whether or not it implements the security policies of the organisation it refers to.

If everything is mobile then life is even harder, because people will have immobile things — pieces of security metadata — which are extremely difficult of access for anybody else, and of course you don't even know where to go and look. It seems to me that the problems produced by all of this have not had anywhere near enough attention paid to them. It can sometimes even happen that there are mathematical obstacles. The VAX/VMS access control allows you to set-up systems which are formally undecidable: if you want to answer the question, what's the exhaustive list of people who could do this, then you can set things up in such a way that you *can't* find out. But it is (presumably) the ensemble of information like that which constitutes an organisation's security policy.

If one looks at the Microsoft Corporation, which I think is in this regard not very untypical of large industrial companies, it has some security requirements that are actually somewhat strange. You may, for example — though this is an IBM example not a Microsoft one — be very anxious that your sales force doesn't know what products are coming down the road. Because if they do know then they'll sell it before it exists. And if you're a big enough company, that gets you into trouble with people. You also have very considerable restrictions

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 1–3, 2002.

on access to human resources data: and if you're a manager (such as I am) then you can look up the pay of everybody who reports through you, but you can't look up your own pay, which is perhaps a little strange.

There are all sorts of rules like this, and it is humanly almost impossible to see whether they do the right thing, or whether they are being applied in a correct manner. As soon as you have distributed systems, you have people responsible for security in all sorts of places, and they have to apply rules which in general terms they don't understand. This is not good. Ross Anderson has this delightful story of two senior bank people who were sent the two halves of a cryptographic key separately with instructions to put them into a security module. Because they were sufficiently senior that they didn't like using their own fingers for any purpose at all, they each independently handed the key material over to the same technician, and said put these two numbers in. Thus they completely lost the purpose of having cut the key into two parts, because they had to follow rules they didn't understand.

If you've got to follow rules that you don't understand, you're liable to circumvent them, particularly if they're boring and obstructive. And rules about security are often boring and obstructive, because although people like us like it, security is in many ways a boring and obstructive subject. That's what it's for, it's for stopping people doing things. Since you can't have an exact match between the technical methods you've got to support security and the security requirements that you want to enforce, you always wind up forbidding some actions which would actually be perfectly reasonable, because it's regarded as better to do that than to permit some actions that are not reasonable. And what this does is to encourage local security managers — who want to be popular with the people they are looking after and don't wish to be seen as obstructive and nasty people — to circumvent the rules, and not do their job properly, to not enforce the constraints they're supposed to enforce.

I'm reliably told that this is a particular problem in the military, because military people are quite adept at circumventing regulations. A great deal of military things have to be done in the field in circumstances of chaos and confusion, and what actually happens is quite different from what was supposed to happen according to the manuals made in Whitehall or the Pentagon or wherever. I've been told by military people that the Ministry of Defence headquarters in Whitehall has one of the largest concentrations of people who know nothing about military actions that you can find anywhere. Indeed I can't resist quoting a remark made to me by a general who was not a Whitehall general: what's the difference between the Ministry of Defence headquarters and Jurassic Park?[1]

Another feature which I think tends to go with mobility is, things not working and having to work. It's happened to us here, from time to time, that we're unable to get on with our work because some person in the Pacific Northwest has set some security access controls wrongly and has then gone to bed. They're eight time zones away, and we can't do anything about it because the person who can fix it is asleep at home. We, being researchers, can stop work for a day

[1] One is a theme park inhabited by dinosaurs and the other is a movie.

and go to the pub if we want to, or we may be able to spend a day doing research and not our administration, and that makes us happy rather than sad. If you're a supermarket, or even more if you are a military unit fighting in action, then you can't stop just because the security's messed up. You have to be able to recover, in short, you have to be able to circumvent the security. You have to be able to substitute the authority of people, *i.e.* the things that senior military officers can say must happen, for the control mechanisms which computerists like us love to put into place to stop things happening.

I wish I could see how to conduct research in this sort of area, how to do research in designing systems that will actually survive and flourish in the real world, which is a very scaly and awkward place. But I must say, I don't know how to begin, and if anyone could discover how to begin to do this type of research I think this would be a great service.

Many years ago (in 1997) I was visiting at Xerox Park and they were doing experiments in what at that time was still laughably known as the paperless office. In businesses forms are passed round from one person to another and approved or signed off, possibly at several levels. We do it here, people file their expenses chits and their manager gets to authorise them. What they were doing at Park was to have forms which appeared on your screen, and you filled the thing in electronically and you signed it electronically, it matters not how, and you then passed it to somebody who was going to approve it electronically, and they would pass it to the next level of approver electronically, and eventually it would all get done, and there was no paper used, and this would be wonderful.

Unfortunately they paid some attention to security. They decided that once part of the contents of a form had been authorised by whoever it was whose responsibility it was to authorise them, then they couldn't be changed anymore, which is the sort of thing that would occur to people like us as being obviously sensible. They then tried this out in, I think, the Santa Clara County Sales Office of the Xerox Corporation. The whole operation ground to a complete halt simply because of an inability to alter things which had already been authorised.

I heard bout this a year or two later, when I was already Head of the Computer Laboratory. I knew instantly that such a thing would never work. I signed all the Department's purchase orders, and I would have been absolutely furious if, whenever anybody needed to change a purchase order after I'd signed it, it would have to come back to me to be signed again, because they'd realised when they're just about to ship it that they'd put a part number down wrongly or that they actually need 12 of something rather than 10.

So what was put in with a very well meaning security motive actually proved extremely destructive. That's another example of how we're not actually very good at matching security mechanisms to the rough and tumble of the real world. I shall close there, and let you get on.

Experiences of Mobile IP Security (Transcript of Discussion)

Michael Roe

Microsoft Research

As I'm doing local arrangements here I usually take the prerogative of giving a short talk. I realised when getting ready for this session that if I were to give the walkthrough of how the mobile IPv6 protocol actually works this would not be a short talk at all. So I'm going to deliberately avoid giving a detailed protocol walkthrough and leave this problem to later speakers if they need it. Instead, I'll talk in general terms about what we need for network level mobility protocols.

In the title I talk about IPSec and the question that really needs to be answered is, do we actually need IPSec? Do we actually need to do encryption at the network level? For many applications we don't. In the non-mobile case we discovered that something like secure sockets layer or Kerberos was perfectly adequate to meet the application needs; that the authenticated identity that those applications needed was frequently something other than an IP address — a site name or a user name, not the IP address which IPSec would have delivered; and furthermore that the data units that we wanted to be authenticated were bigger than IP datagrams. They were data units that were meaningful at the application level, and so it was more efficient to only put one header per application data unit rather than per IP data packet. So IPSec was not applicable in the non-mobile case.

But then we came to look at IPv6 mobility where a protocol, which I'm not going to describe, allows a host to attach at different points in the network. This caused us to re-evaluate IPSec and the reason for this that we have an access control problem. In the network layer there is, at a particular host, a database that says, these particular other hosts are currently at particular places in the network. Who is allowed access to this resource to change it? Who can say that host X has moved and is now at location Y? Well typically it's host X or the person managing it that's allowed to say that, and you need authentication to make sure that it's really is host X saying that before you make the access control decision and change mobility information. This is completely separate from any access control and authentication that might be going on in the application layer to protect the application's resources and to authenticate users that the application knows about. This is access control and authentication being done in the network layer to protect the network itself, and the reason we can't authenticate at a higher layer in this application is because there isn't a higher layer. So we have to use IPSec.

Matt Blaze: Sorry, can I heckle for a moment. Maybe it's because I'm in a sleepy haze, but you appear to have been saying that in the non-mobile case there's no point in ever using IPSec. Were you being provocative or do you really mean that?

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 4–11, 2002.

Reply: I was being slightly provocative but not totally unreasonable. If I wanted to give a different talk I could say that using IPSec in tunnel mode to construct virtual private networks is indeed a fine use of for it that actually is being seen in practice, but end to end IPSec where you separately authenticate each host is not being used much and the reason it's not being used much is there's other ways of doing end to end security than using IPSec.

Matt Blaze: But that's not the point of your talk, right.

Reply: That's not the point of my talk, that would be a different talk.

Matt Blaze: OK, well I'll give you a tough time about that later.

Reply: The reason why I personally was not interested in IPSec were those reasons and this is what caused me and my team to suddenly start looking at it again. So, we turn on IPSec and encounter problem number 1. IPSec has this thing called a security policy database, which is a thing that lives in the network layer and says, for each particular sort of packet whether you should or should not apply particular kinds of crypto processing to it. The default policy most implementations give you is to accept a packet whether it's authenticated or not. So if there's no crypto pass it straight up, if there is crypto on it check the crypto. Now this makes migration from a non-IPSec universe to an IPSec universe very easy. You can gradually add IPSec to hosts and you interoperate nicely with hosts that don't have security.

For confidentiality this might have been a fine policy. You have the sender decide whether the data is worth encrypting, and the receiver will decrypt it if it's encrypted, and just deal with it anyway if it's not encrypted. This is completely hopeless for authentication because the attacker decides whether they want to authenticate themselves or not. Of course any attacker worth their salt is going say, no I don't want to authenticate, just trust me, I'm really the person I say I am. So clearly to stop this you have to change the security policy database from its default setting to a different policy, and the obvious one is to authenticate all packets. This unfortunately doesn't work because of very nasty nitty gritty details of what really happens inside the network layer.

In IPv6, before you send a unicast packet to a destination, there's all kinds of initialisation that needs to take place first. The host needs to be sure that its own address is unique. So it sends out on the network a packet that says, I don't know who I am yet, I could be anybody, but I would like to know if anybody else is already using this address, and if it gets no reply it assumes that it's safe to use it. This is a really hard thing to authenticate because the source address is the "don't know" address which you don't have credentials for. Also there's neighbour solicitation. Before A sends a packet to B, A has to work out what B's link level MAC address is. So A sends out a packet to a multicast group saying, all hosts whose addresses match this pattern, do any of you know what the Ethernet address of this host is? And hopefully gets a reply back. This is hard to do symmetric key authentication with because you're sending out a message that goes to a group of a large number of hosts some of whom maybe you don't trust and so sharing one single session key with that group does not make a lot of sense. So in both of these cases, requiring simple IPSec authentication is not a

viable option. So what you've got to do is change your policy to authenticate all packets except for these ones, and this then leaves you with a burden of doing some analysis to make sure you haven't introduced all kinds of nasty attacks by virtue of turning off authentication for these packets. I'm not going to go through that analysis here, or even claim that it's actually true that you've not introduced any attacks by turning these things off, but I will just say you have to bypass these things.

Problem number 3 is routers. Now routers route packets between interfaces, in particular they route packets that are "not for them", they route packets that are destined for somebody else. If it was the case that a router were to demand authentication for itself on all packets that it routed, firstly this would become extremely inefficient and secondly you'd get cascading headers, as each packet would have to have an authentication header for every router it would go through. This would completely mess up the Internet architecture, so clearly routers have got to be able to route packets without authenticating them — except of course for those messages that aren't really being routed by the router but actually are destined for it. A packet that is actually destined for the router and says, please update the mobility information you've got about this other host out there, should be authenticated — see the previous argument. This is where it gets really nasty. Routers have multiple interfaces and the router often sees a packet for itself twice. It comes in on one interface, it goes once through the packet processing code as a transit packet, and then it's processed a second time as it gets to the second interface and gets passed up the stack. It depends on how your router is implemented, but this is typically how it works. And so your access control policy has got to say, for messages that match this pattern don't demand authentication the first time you see them when you're routing between interfaces but do demand authentication for them the second time you see them when you pass them up the stack. And after you've done all this, your access control policy looks very complex indeed, and of course this very complex policy is something you are checking for every single packet you're receiving, which is like once every couple of microseconds. This is all starting to look like extremely bad news.

Problem number 4 is IKE, the Internet Key Exchange protocol. It's very complex, and this is not a good omen in a security protocol. I think I can quote some of the OpenBSD people who did an implementation of IKE: they said it was 300 pages of specification, 36,000 lines of implementation, and too big to debug[1]. Well, of course I wouldn't say that we would get an implementation wrong, but nevertheless we did not feel particularly happy when setting out to implement something that big. The second thing that's wrong with IKE is that it's a multi-message protocol. Before you've even got a security association in place, you've got to send several messages. Now that means you've got to have

[1] "A Roundtable on BSD, Security and Quality", Jack Woehr, Dr. Dobb's Journal (on-line version), January 2001. This article is a transcript of a panel session at USENIX Security Symposium 2000.

a bypass in place. The messages that you send to set up a session key must be sent unencrypted.

So, that's standard technology, you bypass the IPSec processing for IKE messages. But also you've got to bypass mobility processing because you don't want, as a result of sending a message, to cause mobility processing to happen, which requires a security association, which requires you to send another message and so causes a infinite loop. So the IKE implementation needs to have special knowledge of mobility in addition to its existing 300 pages of specification and 60,000 lines of code. So, once again, this is all bad news.

We do have some recommendations. Firstly a sane model for how we talk about IPv6 mobility. The way you should think about it is that each node, even if it's not mobile, has two completely different sorts of address. There's the home address, which is an identifier for the node which remains the same regardless of where it is, and there's the care-of address, which is an identifier for a point in the network to which the node is currently attached. In both cases a host may have multiple of these, but they're separate things. In the case of a non mobile host, they just happen to have the same value, but in your implementation and in your specification, you have to be clear that these things are conceptually separate. Any time in a protocol where you use an address you have to think to yourself, what is the right thing to use here? Do I really need an identifier that will stay the same if the host moves, or do I need an identifier that will change if it moves?

This is not the way that the RFCs are written. The way they describe it, there's a bit field and at some point in the packet processing the value in the bit field gets overwritten with a different bit field. Everything which processes it before sees the one value and everything that processes it after sees the other value. This is just not a good way to view it. It assumes that in your packet processing there's a point before which everything is only interested in the care of address and after it everything is only interested in the home address. If you look at real implementations this is not entirely true. As things move up through the protocol stack there's a point at which you're interested in both pieces of information and you need to keep track of which is which. So, firstly, have a sane model in your mind when talking about IPv6 mobility.

The second recommendation is that routers should ignore mobility and just get on with routing packets. In particular, there's a trick that at first sight looks a good idea. Home agents, the things that keep track of where a host currently is, could be co-located with routers. This is quite nice because it saves you having two boxes. But then when you work out what that box is going to have to do, in particular when you work out what kind of access control policy it's going to have to apply on each packet that comes in, you discover this wasn't such a good idea after all. In the interest of simplifying your access control policies, it's far better to move the home agent functionality onto another box that's got its own address and write your policies accordingly. So, no mobility processing in routers.

Recommendation 3 is just say no to IKE. There's another talk I could give you that I won't give here, just to say that we have a paper in this April's Computer Communications Review[2] on, not a general purpose replacement for IKE that does all the 1001 things that IKE does, but a simple protocol that does the one thing that we need in this particular circumstance, and does it much, much more simply.

Recommendation number 4 is don't ever again invent a protocol that does what the duplicate address detection protocol does. It's a standard thing in networking that networks may lose or duplicate messages. So what do they do? They invent a protocol that relies on networks never losing or duplicating messages when the network has just been unplugged and re-connected at the physical level. It turns out this is absolutely the worst time to rely on networks not losing or duplicating messages. There are many network interface cards that discard all packets the host asks them to send for the first ten seconds. The duplicate address detection protocol fails in that case. We've also seen the opposite problem, where for a couple of seconds after you've come into range of a new base station, packets are duplicated because you're getting the packet in two different ways. So, replace the duplicate address protection protocol with something else. Replacing it with nothing would be better, because the only reason it's there is because people believe Ethernet addresses aren't always unique. You do this complicated protocol which doesn't work to try and find out if some other machine on the same Ethernet as you has been given the same Ethernet address.

OK, that's about it for the initial pitch, I'll open the discussion for questions. Matt?

Matt Blaze: I can't help but be amused by your initial premise, where essentially you point out that IPSec — except maybe for tunnel mode, you conceded — is pointless because you can be doing application layer security, and then went on to say, except for mobile IP in v6. The reason I can't help but be amused by this is that in 1992 when John Ioannidis was finishing his dissertation on mobile IP where he wrote the initial mobile IP protocol, he and I sat down to try to figure out how to add security to mobile IP. What we came up with was a simple encapsulation based protocol called Swipe that later evolved into IPSec, because we ended up concluding that trying to do security for mobility as a special case was just kind of stupid, we might as well solve the general IP security problem. And then we ended up concluding — not to put anyone's thesis down, particularly my collaborator's — that maybe mobile IP was kind of stupid in the first place and what you really want for mobility is IPSec tunnelling. So, are these problems worth solving? We have IPSec in tunnel mode, what do we need all this mobile IP nonsense for since it obviously leads to these ridiculous network layer problems that can easily be broken by slightly bad protocol assumptions?

Reply: Perhaps, it's a strong argument. IP mobility may just die because people realise they don't need it, but one of the things that people are trying to do

[2] "Child-proof authentication for MIPv6 (CAM)", Greg O'Shea and Michael Roe, Computer Communications Review, April 2001.

is route optimisation. The idea is that if two hosts that are both mobile suddenly find themselves together on a high bandwidth local area network, they can send data to each other locally without going via some remote agent that's the other side of the world across a very slow link. The need to do route optimisation is what's driving all this complexity. If we didn't have route optimisation we could do away with a great deal of this.

Pekka Nikander: Well I think that we should actually go a little bit deeper and think about what a host is, what an address is, and what's the relationship between them. It used to be that each host had approximately one address, and it used to be a static address. Starting from that point it made sense to look at things like mobile IP. But nowadays it's not true anymore. The host is not the IP address, the host just happens to have one or more addresses at a given point in time and the address is just an identifier for a topological location in the network.

John Ioannidis: So every time this mobile IP and security thing comes up I sort of feel the need to speak, and it's not clear why. Mobile IP looked like a good idea in 1991. It's been like ten years now, and I have yet to see a real need for it. There is a kind of mobility that is solved extremely well at the link layer, like the Wavelans do, and there is this other kind of mobility that cell phones do that has so much interaction with other stuff that maintaining the IP address as the host moves is the least of our worries. Now that said there are some valid, research location issues which can be disguised as mobile IP, which is finding out where a particular host is attached to the network. We can do that with the standard mobile IP way of saying, we'll go to the real address and then we'll get redirected or tunnelled or in some other way pointed to the care of address, and exactly how we do that is an orthogonal problem. The problem of finding out where a host is right now so we can talk to it and conversely have them find where I am so they can talk back, and the problem of how we get there, are two orthogonal problems that can share mechanisms for implementing them. You need two pieces of information to talk to a mobile host. You need to know where it is, and you need to know how to get there. The same piece of equipment usually knows both pieces of information. Because it's the same piece of equipment we have a tendency to think of those two problems as the same problem. Now getting from here to there is a simple tunnelling problem, we have about 15 tunnelling protocols of which IPSec is only one. Actually figuring out if it's worth going through all the trouble of finding and optimising the route to the host is a different problem. My issue with all the mobile IP security schemes is that they're trying to solve both the problems simultaneously, ignoring the security work of IPSec and other work in research locations.

Reply: I think the real research issue is that there's an architectural issue that has to be sorted out, namely that we can no longer assume that a particular host is always connected to the same point in the network topology. At some point in the long train of argument from the application's knowledge of which host it want to talk to, down to how the packet gets there, there's got to be an indirection. You're right that it may not have to happen at the IP layer and

there are indeed strong arguments that it might make sense to do the mobility at a layer higher, at a level above IP.

Matt Blaze: What problem isn't solved by DHCP plus a reasonable dynamic DNS system for finding the host's current IP address by name?

Reply: If only we had a dynamic DNS that worked ...

Tuomas Aura: That would be slower than mobile IP, but we could use something similar.

John Ioannidis: On the other hand. DHCP exists and mobile IP doesn't. At least, it isn't deployed everywhere. The sad thing is that mobile IP existed first.

Matt Blaze: Neither of those things [DHCP and dynamic DNS] involves changing the network.

Bruce Christianson: The point where you came in on this, Mike, was that the network is itself a distributed system with a lot of resources. It needs to protect itself against idiots, and the trouble with mobility is that the neighbour you know may now be acting as a proxy for somebody that you don't know. Consequently you need now to know who it's acting as a proxy for before you come to a view about whether or not it might be behaving like an idiot.

Reply: I would phrase it slightly differently. I think what's happening at the moment in networks is that access control and authorisation is being done on the basis of topology. On this building's internal network we're behind a firewall, and there is an assumption that anything that connects to the network in here is authorised because it was allowed into the building. In the mobile world where anything can connect from anywhere you can no longer rely on a connection point in the topology as an indication that something is authorised to do a particular thing. So you need something else, and we've got a problem.

Bruce Christianson: It was actually dangerous to rely on that all along, even behind a firewall.

Reply: It was dangerous all along, but it just about held together and it's all about to fall apart horribly when we introduce mobility.

John Ioannidis: There's actually a very elegant solution to that problem. Traditionally we separate the inside from the outside of a network topologically; a much better solution is to separate by knowledge of the right IPSec associations, and there is a paper[3] in either the last or the next USENIX Security Symposium by Steve Bellovin and Sotiris Ioannidis, my brother, on the topic of distributed firewalls, that's what they've called it, where what's inside the firewall is decided by whether you know the right keys, not by whether you are connected to the right piece of wire.

Pekka Nikander: Do you think that the situation changes if you consider end hosts multihoming? That is becoming popular — more and more end hosts attach simultaneously (more or less) to topologically different parts of the network. You might want to use both of those connections, and they might have

[3] See also "Implementing a distributed firewall", S. Ioannidis et al., ACM conference on Computer and Communications Security, Athens, November 2000.

different quality of service facilities and so on, so that you might want to change how you use the connections dynamically.

Reply: I could see you could want to do that. When you introduce mobility features you see ways you could use them for things that are really nothing to do with mobility, like switching between different providers or switching between different links that have different quality of service, or indeed migrating services between different CPUs and making it look like they're one host that has just moved physically. I feel this is going to buy us into a whole new lot of problems when we start investigating this, I don't feel confident to say anything more than this other than it looks a dangerous area for us to start walking into.

Tuomas Aura: If you're moving from one local area to another, you are registered in one place and you want to register in a new place, and if you use DHCP to get the new local address then that is too slow because you will have maybe one second of interruption in the connection. If you use duplicate address detection you still have a delay. If you could use the old address in the old network and the new address in the new network in the area where these networks are overlapping, would that help?

Reply: I think we need to do mobility at different layers for different timescales. For movement between different base stations, on the scale of seconds, I don't think it makes any sense to do that at the IP layer because it just takes too long and has too much overhead. I think IP level mobility or application level mobility is for the case that happens on a slower timescale, when you move between two different domains. You walk out of this room and into the street and go from being on a private network to being on a public network. I think it's optimistic of us to expect that we can make the network level or application level mobility fast enough that it's going to work between base stations that have ranges of a few metres, I don't think we can do that efficiently enough.

John Ioannidis: There's no conceptual reason why we can't, it's just that the mechanisms we have proposed try to hide too much of the underlying hardware for there to be enough interactions between the network, the Internet layer and the underlying stuff for it to happen quickly, You have to do the same kind of work to move from cell to cell as to move from IP address to IP address, you just need to know where you are and get the new information. However, exactly because we're trying to hide the details of the underlying link and physical implementation at the IP layer, this generality that we're trying to invoke hurts us in trying to do it quickly. It's not that there is something magical about doing it in the link layer that allows it to happen more quickly, it's just that we have more information about where things are physically and how they're behaving physically that we don't have at the IP layer.

Reply: IP's reason for existing is to provide an abstraction that's independent of the particular hardware, but to do these fast handoffs you need to know an awful lot about the hardware. That seems to be a strong argument for putting mobility in the link layer that has the information needed to do it properly.

Denial-of-Service, Address Ownership, and Early Authentication in the IPv6 World

Pekka Nikander

Ericsson Research
Pekka.Nikander@nomadiclab.com

Abstract. In the IPv6 world, the IP protocol itself, *i.e.*, IPv6, is used for a number of functions that currently fall beyond the scope of the IPv4 protocol. These functions include address configuration, neighbour detection, router discovery, and others. It is either suggested to or required that IPsec is used to secure these functions. Furthermore, IPsec is used to protect a number of functions that are considered dangerous in the IPv4 world, including mobility management and source routing. Now, the currently prominent method for creating IPsec Security Associations, the Internet Key Exchange (IKE) protocol, is both relatively heavy and requires that the underlying IP stacks are already fully functional, at least to the point that UDP may be used. As a result, the combination of the widened responsibility of IPsec and the relative heavy weight of IKE creates a vicious cycle that is a potential source of various denial-of-service attacks. Additionally, if we want to use IPsec to secure IPv6 autoconfiguration, a chicken-and-egg problem is created: fully configured IPsec is needed to configure IP, and fully configured IP is needed to configure IPsec. In this paper, we describe these problems in detail.

1 Introduction

In IPv6, security is assumed to be based on IPsec. First, IPsec is used to protect basic application traffic such as standard TCP connections and UDP packet flows. Second, IPsec may be used for protecting ICMPv6 messages [1], which are, among other things, used to configure the IPv6 stack during boot time [2]. Third, IPsec is required to be used to protect a number of message types that have various IP specific control functions. These message types include Mobile IPv6 Binding Updates and Binding Acknowledgements [3], the Routing Extension Header [1], the IPv6 Router Renumbering messages [4], and there will probably be more in the future. Since this paper *only* concerns IPv6, from here on the terms IP and ICMP are used to denote IPv6 [1] and ICMPv6 [5], unless explicitly otherwise noted.

As we show in this paper, this approach of using IP to secure IP has a number of potential security pitfalls. These problems are not necessarily exceptionally hard to solve, but their solving requires a crisp understanding of the situation. Unfortunately, based on the discussion at the IETF ipsec mailing list [6], many of the people currently involved in the IPv6 design and deployment have hard

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 12–21, 2002.

time to understand the scope and nature of these problems. In this paper, we attempt to analyse the specific nature of these problems, illustrating the underlying principles when appropriate.

To lay background for the forthcoming discussion, some understanding of the IPv6 and IPsec design is needed. We assume some familiarity with both IPsec and IPv6 from the reader, and do not discuss the technical differences between IPv4 and IPv6. On the other hand, we do illustrate the differences in the IP-internal signalling mechanisms, since they are one of the fundamental source so the security problems discussed.

In this paper, we aim to analyse a chicken-and-egg and the so called address ownership problems and propose a possible approach to alleviate them. Thus, the rest of this paper is organized as follows. First, in section 2, we discuss how the IPv6 architecture differs from IPv4 architecture, and how this, together with some already existing deficiencies, leads to a number of potential denial-of-service and authorization problems. Based on the analysis, in section 3, we outline a number of requirements for a potential protocol that could be used to alleviate the situation. In section 4 we define a number of building blocks that could be used to design such a protocol. Finally, section 5 includes a summary.

2 Security Problems in IPv6

In this section, we explain in detail an IP autoconfiguration *vs.* IPsec chicken-and-egg problem and the so called address "ownership" problem. When looking at these problems, our main focus on looking at potential denial-of-service attacks, and attacks based on improper assumptions and insufficient authorization. In sub-section 2.1, we look at the IPv6 autoconfiguration phase, discuss the chicken-and-egg problem, and also note some related phenomena. Sub-section 2.2 discusses the address "ownership" problem, mainly in the light of Mobile IPv6.

2.1 Autoconfiguration

The stateless autoconfiguration specification [2] defines a method of providing initial boot time configuration for any IP host. Since the autoconfiguration is based on solely local traffic, physical link security is often good enough to provide adequate security against potential DoS and other attacks. However, if we ever want to use stateless autoconfiguration in wireless networks, such as future ad hoc networks, some sort of security must be provided. The security measures can be either provided by some mechanism that is outside the scope of IP and autoconfiguration (*e.g.* through a layer 2 protocol), or the autoconfiguration mechanism can be enhanced to deal with the potential threats. In this paper, we focus on the latter approach.

In the basic stateless autoconfiguration process, a booting host sends a series of *Neighbor Solicitations* [9] to the local link. These messages contain a *tentative IP address* that the host would like to use at its link local IP address. If the

tentative address is already in use by some other host, it will send a *Neighbor Advertisement* as a reply, and the booting host must select a new tentative address. On the other hand, if the booting host receives no replies to its solicitations, it is free to use the address.

Once the host has a link local address, it enters the second phase of autoconfiguration. It sends a series of *Router Solicitation* messages, basically requesting the local routers to announce themselves. The local routers are expected to answer with *Router Advertisements*. In addition to identifying a router, each Router Advertisements also contains a number of *routing prefixes*, which the booting host can use for creating globally routeable addresses.

To enhance security, the above mentioned Neighborhood Discovery messages may be protected with IPsec AH [9]. Potentially, AH could be used by the booting host to validate that the Neighbor Advertisements and Router Advertisements that it receives do contain proper and accurate information. That is, given a suitable set of AH SAs, the host can verify that the Advertisements it receives are really valid and authorized. An approach to define the required SAs trough manual configuration is defined in [10].

Unfortunately, there is currently no other mechanism (but manual configuration) to provide such SAs. There are two basic reasons for this. First, the only currently available automatic method for creating SAs, *i.e.*, IKE, requires a functional IP stack in order to be usable. Thus, IKE cannot be used to set up SAs before the IP has at least an established local address, and secure establishment of the local address requires SAs. Second, even if some other non-manual means of establishing SAs was available, it is not necessarily easy to determine on which bases the hosts are authorized to claim "ownership" of addresses.

Thus, to summarize, in order to secure stateless autoconfiguration for wireless and ad-hoc networks, we first have to find out a way of solving the chicken-and-egg problem, then find a way of defining authority over local IP addresses and routing prefixes, and finally provide some protection against denial-of-service problems.

2.2 Address "Ownership" and Mobile IP

The address "ownership", mentioned above, is actually a quite generic problem, not only limited to stateless autoconfiguration and local IP addresses. In fact, it appears to be a hard problem to solve in the global scale. This becomes apparent once we start to consider IP level signalling that allows one to change a host's idea how to reach other hosts. That is, when such signalling is not used, the host uses the default routers and explicit routes (manually configured or learned through stateless autoconfiguration) to send packets destined to a remote host; on the other hand, when such IP level signalling is used, the host will no more send the packets based on the route information but use the information learned through signalling. Clearly, this is a potential source of various traffic diversion and denial-of-service attacks.

As a (partial) solution, all the current specifications require that the signalling messages must be protected with IPsec AH. However, most of the specifications

forget to require that, in order to be secure, the SA must be *authorized* to define new reachability information for the particular address(es). Furthermore, none of the current specifications define how to determine proper authorization. Moreover, it turns out to be a hard problem to determine who "owns" a specific address, and therefore who has the authority to define how to reach that address. Now, before going into the "ownership" problem and some potential solutions, let us briefly consider a potential attack in the light of Mobile IPv6.

Mobile IP signalling. In Mobile IPv6, a mobile node (MN) may send any other node — called a correspondent node (CN) — a Binding Update (BU). Basically, a BU says that "this MN can be currently better reached through this route". Both the MN and the route are specified in the terms of IP addresses. The MN is identified by a home address, the "better route" by a so called care-of-address (CoA), which is expected to be a temporary address that the MN is currently reachable at.

When the CN receives a BU, it has to determine if it can trust the information in the BU. First, it has to verify that the BU is untouched and that its source is known. IPsec AH takes care of this, and usually the AH processing is performed separately, before BU processing. Once the CN knows whom it received the BU from (or, equally, which AH SA was used to protect the BU), it has to decide if the sender is actually authorized to specify a binding for that particular home address. If it fails to make this decision, and blindly assumes that AH protection is enough, basically anybody having a valid AH SA can redirect any traffic to itself. To do so, the attacker simply has to include BU that specifies the target address as the home address and an address controlled by the attacker as the care-of-address. If the CN accepts the BU, all traffic sent to the target address will be sent to the attacker specified care-of-address.

In global scale, the problem of providing the CN with information to decide whether to act according to a BU or not seems to quite hard. That is, if the MN is not already known to the CN, the CN has a hard time finding out whether the MN really "owns" the claimed home address, *i.e.*, whether it is authorized to create a binding for the claimed home address[1]. Clearly, it cannot simply believe MN's claims since the MN may lie. And it may lie even if its identity (but not authority) had been authenticated during the SA creation process.

One suggested approach [6], suggested by Tero Kivinen, is to include the necessary information in the certificates (or other credentials) used by IKE when creating the SA. Unfortunately, even this may not sufficient, since the certificate or credential issuer might lie or not know better. That is, unless the underlying infrastructure reflects the "ownership" of IP addresses and address prefixes, one has to rely on that all of the certificate authorities (CA) are honest and competent. Since, in the global case, there will most probably be thousands (or maybe

[1] As the default case, the CN should decline from creating the binding unless it knows better. The traffic will still flow through the home agent, anyway. Unfortunately, this solution also undermines the benefits of Mobile IPv6 Binding Updates, making them usable only in special cases

millions) of CAs, this is an unreasonable assumption. In the rest of this paper, we call this approach the "generic PKI approach."

Another proposed approach [6] is to rely on the IETF Authentication, Authorization and Accounting (AAA) infrastructure[7]. Unfortunately, this approach suffers from the same problems as the generic PKI approach. That is, unless all of the operators participating in the AAA infrastructure are honest and competent, the operators either have to locally include information about the addresses "owned" by the other operators, or blindly trust the other operators. Furthermore, it seems unlikely that all of the Internet would use the AAA infrastructure, ever.

If the reverse DNS database was accurate and secured with DNSsec, it would provide a kind of PKI that effectively reflects the ownership of IP addresses. Unfortunately, a typical secure reverse DNS lookup would most probably require at least two or more DNS queries, and verification of quite a few public key signatures. Thus, using secure DNS to check validity of Mobile IP Binding Updates is likely to be far too expensive in the terms of performance and resource usage.

We think that solutions to the address "ownership" problem should not be sought from using global security infrastructures, but either by using the existing routing infrastructure or through binding the IP address directly to a public key [8].

3 Requirements for a Weak Early Authentication Protocol

Our goal is to design a protocol that can be used early, when two nodes start to communicate, to provide both nodes initial assurance about the benevolence of the peer. In particular, we want a protocol that allows two previously unrelated nodes to create shared keying material without relying on any external infrastructure. Since we want to protocol to work without requiring any existing security context, the protocol cannot provide assurance about the identity or "real honesty" of the peers. However, since the purpose of the created keying material is only to provide relative security in the early phases of communication, weaker properties are often sufficient. In our case, we aim to provide assurance about the reachability of the peer, combined with denial-of-service protection.

The result protocol is planned to be usable for creating initial IPsec SAs for various purposes. In particular, the SAs should be usable for protecting early ICMP messages used for autoconfiguration, and for protecting Mobile IPv6 Binding Updates. Applicability to protecting ICMP router advertisements, ICMP redirects, and router renumbering messages would definitely a plus, but we do not currently see how to protect them without any local configuration.

3.1 Reachability Requirements

As a minimal level of protection against misbehaving nodes, the nodes should be able to check that the alleged peer node is really reachable and replies to

messages sent to it. This is especially important to the responding party, since it has no prior information about the sender of the first packet received. Furthermore, since the responding party has no assurance about the benevolence of the sender, it should use a minimum amount of resources in gaining assurance about reachability. That is, the aim of the packet sender might be plain resource exhaustion, *e.g.*, by filling the memory of the recipient.

Thus, the requirements can be specified as follows.

R1. The initiator must find out if the responder is reachable, *i.e.*, whether the responder replies to packets sent to it.
R2. The responder must find out if the initiator is reachable, *i.e.*, whether the initiator replies to packets sent to it.
R3. The responder should not need to consume any memory to check reachability.
R4. The responder should be able to use a minimal amount of CPU to check reachability.
R5. Since the CPU and memory usage are interdependent, the responder should be able to make a choice between memory usage and CPU usage, depending on the current conditions.

3.2 Denial-of-Service Protection Requirements

Since our protocol is meant to be run early in the communications, we cannot assume anything about the honesty or competence of the peer nodes. That is, we must assume that possibly the only reason for a peer to send packets is to cause a denial-of-service situation. Thus, we want to design the protocol in such a way that we can discard improper packets as soon as possible. Furthermore, we want facilities that allow a node to require some level of commitment from a peer before committing communication with it.

Again, protecting a recipient from a malicious initiator seems more important than protecting the initiator from a malicious responder. Thus, when there is a conflict, we choose to protect the recipient. On the other hand, in some cases the initiator may benefit from learning a public key of the recipient as early as possible, and even from gaining assurance that the recipient actually possesses the corresponding public key. Thus, we may define the following, partially conflicting, requirements.

R6. The recipient should be able to discard illicit packets as soon as possible, after using the minimum possible amount of resources.
R7. The recipient must not need to use resources, such as memory or CPU, before the initiator has used a reciprocal amount of resources.
R8. The recipient should be able to control its resource usage, allowing it to vary behaviour from committing resources, without requiring reciprocity, in low load situations, to requiring full reciprocity in high load situations.
R9. The initiator should not need to consume large amount of resources, for reciprocity or for other purposes, before it has some assurance that the responder requiring it to do so is a legitimate one.

R10. Running the protocol should result in a situation where both the initiator and the recipient are able to easily discard packets that have not been sent by the peer.

3.3 Higher Level Goals

Since we do not assume any permanent identifiers in the context of our protocol, we cannot define the protocol goals using typical identity assumptions. Instead, we must be very careful and speak only about addresses and reachability in one hand, and mutual symmetric keys or public key pairs on another hand. That is, when specifying goals from the initiator point of view, we cannot speak about a specific "responder" since such a party may not exist. Instead, we may make requirements about addresses, reachability, and keys of an assumed recipient.

Given these preliminaries, the goals of running the protocol may be defined as follows. The reader should note that these goals do not adequate address denial-of-service protection; the more specific requirements need also to be considered.

G1. The initiator knows that the there is a party reachable at the used address, and that the party replies to messages sent to it.
G2. The responder knows that there is a party at reachable at the used address, and that the party replies to messages sent to it.
G3. The initiator believes that the reachable party shares a piece of symmetric keying material with it, and assuming that the peer has not voluntarily divulged it, nobody else possesses that material.
G4. The responder believes that the reachable party shares a piece of symmetric keying material with it, and assuming that the peer has not voluntarily divulged it, nobody else possesses that material.
G5. The responder believes that the peer, *i.e.*, the (alleged) party sharing the keying material with it, has actually consumed a reciprocal amount of memory and/or CPU resources during the protocol run.
G6. Optionally, the initiator believes that the peer, *i.e.*, the (alleged) party sharing the keying material with it, has access to a private key corresponding to a particular public key. The public key may be learned during the protocol run, or the initiator may have had prior knowledge about the key.
G7. Optionally, the responder believes that the peer, *i.e.*, the (alleged) party sharing the keying material with it, has access to a private key corresponding to a particular public key. The public key is usually learned during the protocol run, but it is also possible that the responder had prior knowledge about the key.

3.4 Non-security Related Requirements

In order to be really usable in the aimed environment, the protocol should also fulfil a number of "protocol engineering" requirements. These requirements do not necessarily have anything directly to do with security.

R11. The protocol should use a minimal number of messages. Two messages (one from initiator to responder and one reply) would be ideal, but seems impossible. A trigger message (from the initiator to the responder) plus two actual messages (responder → initiator & initiator → responder) seems achievable.

R12. The protocol messages should fit in a minimal number of bytes. Ideally, they should easily fit in IPv6 extension headers piggybacked in initial TCP messages. Thus, if possible, the messages should be less than 500 bytes in length.

R13. The protocol should be designed to be flexible. That is, if possible, it should consist of reusable pieces like the ISAKMP protocol does, but it must be far simpler than the ISAKMP protocol is.

R14. It must be possible to implement the protocol completely in an operating system kernel. This means that the implementation must fit in relatively small space, ideally in less than 4 kilobytes or so (not counting the actual cryptographic algorithms).

This concludes our requirement and goal setting.

4 Building Blocks

In this section, we define a number of mechanisms that can be used as building blocks in developing the kind of protocol we desire. In particular, we use the low order bits of IPv6 addresses as cryptographic tokens, suggest that reverse hash chain revealing could be used to resolve certain conflicts, define how to use the existing routing infrastructure to check reachability, and how puzzles can be used to solve the computation reciprocity requirement.

4.1 Using IPv6 Addresses as Cryptographic Tokens

As first publicly suggested in [8], 62 of the low order bits of an IPv6 address can be used to store a cryptographic hash of a public key. The basic mechanism can be defined as follows:

`host ID` = HASH_{62}(public key)

However, 62 bits are too few to gain strong security and real protection against birthday collisions. One possible way to alleviate the situation is to add some random data as follows:

`host ID` = HASH_{62}(public key | `random`)

Now, by using these bits as the low order 62 bits, the host implicitly claims that it generated the `host ID` since it knows `random`, and that it intends to use the public key. Unfortunately backing this claim works only once, since once the host has divulged `random`, it has to use expensive public key crypto to prove that it knows the secret key corresponding to the public key.

4.2 Extending the Address Token Idea with Reverse Hash Chain Revealing

Let us now consider a slightly more complex construction, as follows:

H_N := HASH_{160}(public key | `random`)

H_i := HASH_{160}(public key | H_{i+1})

`host ID` := HASH_{62}(H_0)

Now the host can show that it has generated the sequence H_0, ..., H_N, one by one, without needing to use public key cryptography. Thus, in the case of collision, both parties just reveal their H_1. Since the hash values H_i do not need to be just 62 bits long but can be *e.g.* 160 bits long, collisions become extremely unlikely and the only reason why two hosts would reveal the same H_1 is that one of them has learned the value from the second. In that case the dispute can be resolved by revealing H_2. Furthermore, if a host is ever demanded to reveal H_i with i larger than 2, it can be virtually sure that an attack is going on, and act accordingly.

4.3 Relying on the Routing Structure

The routing infrastructure is the essence of the Internet. It takes care of delivering the packets to the addressee. If the routing infrastructure is compromised, the whole Internet is compromised. Consequently, we can *assume* that the routing infrastructure remains uncompromised, and base some protocol elements on this assumption. That is, if our assumption does not hold, *i.e.* if the routing infrastructure is compromised, there is no benefit from perform any signalling related checks either.

The basic idea is to check apply a challenge-response protocol. That is, the verifying party sends a challenge to the address whose "ownership" or reachability it wants to check, and waits for a response. The actual nature of the challenge-response depends on the other aspects of the protocol; for example, it can be combined with the ideas presented above.

4.4 Puzzles

To further protect against resource exhausting denial-of-service, the protocol responder can set up a puzzle that the initiator must solve before further conveying the protocol [11]. Furthermore, it is possible to construct the puzzle so that it is implicit and protocol initiator specific, allowing the protocol initiator to solve the puzzle before contacting the recipient, or in parallel with waiting for the first reply from the recipient.

For example, the recipient may require the initiator to provide a random number R so that there are K right most bits zero in HASH(A_I | A_R | `time` | R). Once the initiator has solved the puzzle, it must send the recipient `time` and R, using A_I as its own IP address and A_R as the recipients IP address. To verify, the recipient checks that `time` is relatively close to its idea of time, computes the hash value, and checks that there are the required number of zeros.

5 Summary

In this paper, we have discussed a number of security deficiencies related to IPv6. In particular, we have discussed potential security pitfalls caused by using IPsec to protect IP level signalling in IPv6. To illustrate some potential problems, we have discussed a number of problems related the IPv6 autoconfiguration and Mobile IPv6 Binding Updates. To us, it seems that there are two major problems: a chicken-and-egg problem related to using IPsec to configure IP *vs.* using IP to configure IPsec, and a problem called address "ownership" problem that mainly relates to mobility and routing related IPv6 signalling.

Based on the discussion of the above mentioned problems, we have derived an initial list of requirements and goals for a protocol that would protect IPv6 hosts from a number of denial-of-service and traffic diversion attacks. Finally, we have outlined some building blocks which could be parts of such a protocol.

References

1. S. Deering and R. Hinden, Internet Protocol, Version 6 (IPv6) Specification, RFC2460, Internet Engineering Task Force, December 1998.
2. S. Thomson and T. Narten, IPv6 Stateless Address Autoconfiguration, RFC2462, Internet Engineering Task Force, December 1998.
3. David B. Johnson and Charles Perkins, Mobility Support in IPv6, work in progress, Internet Draft draft-ietf-mobileip-ipv6-12.txt (expired), Internet Engineering Task Force, April 2000.
4. M. Crawford, Router Renumbering for IPv6, RFC2894, Internet Engineering Task Force, August 2000.
5. A. Conta and S. Deering, Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification, RFC2463, Internet Engineering Task Force, December 1998.
6. Discussion at the IETF ipsec mailing list in January 2001, ftp://ftp.tis.com/pub/lists/ipsec/ipsec.0101
7. C. de Laat, G. Gross, L. Gommans, J. Vollbrecht, D. Spence, Generic AAA Architecture, RFC2903, Internet Engineering Task Force, August 2000.
8. Greg O'Shea and Michael Roe, Child-proof authentication for MIPv6 (CAM). Computer Communications Review, April 2001.
9. T. Narten, E. Nordmark, and W. Simpson, Neighbor Discovery for IP Version 6 (IPv6), RFC2461, Internet Engineering Task Force, December 1998.
10. Tero Kivinen, Markku Rossi and Jari Arkko, Manual SA Configuration for IPv6 Link Local Messages, work in progress, Internet-Draft draft-arkko-manual-icmpv6-sas-01.txt (expired), February 2001.
11. Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo, DOS-resistant Authentication with Client Puzzles, presented at Cambridge Security Protocols Workshop 2000, April 3–5, 2000, Cambridge University. LNCS 2133, pp. 170–178, Springer, 2001.

Denial of Service, Address Ownership, and Early Authentication in the IPv6 World (Transcript of Discussion)

Pekka Nikander

Ericsson Research

I'm going to speak about more or less about the same problem as Mike, but from a different point of view — denial of service. If we consider networks where we don't have any kind of authentication infrastructure, for example, if we're considering ad-hoc networks based on IPv6, we will find a number of new security problems which I have given names.

The first one is what I call the configuration denial of service and chicken and egg problem. The chicken and egg problem is basically the same problem that Mike described, why you can't use IKE when you are trying to protect the auto-configuration phase of IPv6 with its network solicitation and network advertisement messages.

There is another problem which I call the address ownership problem, and a particular of flavour of it which I call future address stealing. I'm trying to work out a solution which will remedy some of these problems in the kind of network where you don't have an authentication infrastructure. If we assume that there are no prior security relationships, it means that we can't really solve the problems in the strong sense — we can't really trust the other parties — but I'd like to get some level of trust and some level of authentication to protect against various denial of service problems and make the boot up phase of IPv6 easier. So far, I have come up with four ingredients for this kind of solution. The host ID part of the IPv6 address can be used as a crypto token, which is exactly the same idea that Mike and Greg had in their paper[1]. Then we can create the host ID as a series of one-time passwords. We can use random addresses so that we can protect against the future address stealing problem, and then we can rely on the routing infrastructure.

Let's go and have a look at why IPv6 is different from IPv4. There is the stateless auto-configuration of mobile IPv6, which means from the functional point of view that in IPv6 a host selects an address for itself. We have RFC 3041 which allows you to select a random address for yourself, and mobile IPv6 allows you to inject routing information to your peer. Then there is the bigger issue — I believe that hosts will not be represented by single addresses anymore. We will have a dynamic set of addresses which is the host. Then there is the HIP[2] faction in IETF which believes that we have to represent the host as some kind of identifier. I wasn't at the HIP BOF where JI shot down most of HIP, but the

[1] "Child-proof Authentication for MIPv6 (CAM)", Greg O'Shea and Michael Roe, Computer Communications Review, April 2001.

[2] "Host Identity Payload Architecture", R. Moskowitz, Internet Draft, February 2001.

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 22–26, 2002.

idea is more or less the same, that the host is not static in the address sense anymore.

So, if we look at IPv6 autoconfiguration we have the denial of service problem, if we are doing duplicate address detection it's very easy to prevent a host from getting any address. In duplicate address detection you send a neighbourhood solicitation message, saying "I would like to use this address", and an attacker will simply say, "No, you can't use that address because I'm already using it", and that can go forever and you don't get an address for your host. This is another reason why duplicate address detection is such a bad idea.

Then there is the chicken and egg problem. To use IPSec to protect these messages we need security associations, and to use security associations we have to establish them, and to establish them we have to use IKE, and to run IKE we have to use UDP, and to use UDP we must have an IP address that we don't have yet.

The second problem, which I think is actually a bigger problem, is what I call the address ownership problem. If we consider mobile IPv6 and other mechanisms, and there are maybe half a dozen mechanisms in IPv6 which are very similar in the sense that they change the routing information of end hosts. The question is, who is really authorised to change this routing information of end hosts? One answer is to say, whoever is bound to or controls the address whose routing information is being changed. OK, who owns that address and how do you show that you own the address?

Some people have proposed that we must have a global PKI and secure reverse DNS, where IANA says we distribute the addresses in this way, and then all the ISPs use secure DNS and then you can do a secure DNS look-up. You check about six signatures, you do about four look-ups in the general case and it takes maybe 15 seconds to get a reply from the secure DNS — if it existed.

And then there are the AAA people who believe that every host in the Internet will be controlled by the network operators. They believe that AAA will do that as well, but there is the same deployment problem. Even if everybody were going to use it there is the problem that AAA doesn't carry authorisation information in the way that it should do. So those are not solutions.

One particular interesting attack based on the address ownership problem is what I call the future address stealing attack. The outline goes like this. You have a victim that you want to attack and you know the IP address that your victim is going to use, so you go before your victim to the wireless link that it's going to use and use the IP address that your victim is going to use as your home address. Using it as your home address, you contact the other host that your victim will contact and establish a binding on that other host and then you go away. So when your victim comes here and starts to communicate with the other host there is already a binding at the host saying that all the reply packets are sent to the care of address in the binding. So you get all the response packets after this attack. Of course, what you can do with them depends on whether your victim is using IPSec and so on, but at least you can do a denial of service pretty easily.

So what could we do to solve these kinds of problems? One realisation is that in IPv6 addresses there is the host ID part, and in the host ID part we have 62 bits that are random bits according to RFC 3041 — they don't have any semantics. Several people have come up with the idea that we can use those bits as a crypto token and bind an address to the public key so that these 62 bits are a part of a hash code of the public key. Another ingredient might be that we could protect the addresses using one-time password like mechanisms, so that instead of just generating one random part, we generate a series of these random parts using a hash function, and then we reveal them one by one. In this case we can solve the denial of service problem for autoconfiguration and neighbour solicitation and neighbourhood advertisement messages.

We can just use random addresses so that nobody can steal our future address because nobody knows our future address, and then maybe we can rely on the routing infrastructure to verify some part of the ownership by a simple challenge response protocol. The routing infrastructure is supposed to define the topology of the Internet, so if somebody gets to the routing infrastructure we are in trouble in any case! So maybe we can rely on that.

In practice, how could we do that? How could we first use the host ID as a crypto token and then use one time passwords? We could generate the host ID by making a hash of the public key and some random number, and then by revealing the random number we show that it was me who generated this host ID because I knew the random number and this is also the public key that I want to use. That only works once, because after that everybody knows the random number and it's not a password anymore. So we can do the one-time password thing as in S/KEY, we generate a series of these and reveal them in reverse order.

I'm working on a protocol where all these issues are all combined together, but I don't want to go into the details, at least at this stage. This is a three message protocol. First the claimant sends to the verifier its address, a public key, a nonce and H_0, which is the last of the hash series. The verifier calculates a challenge and sends the challenge, its public key and a certificate to the claimant. The claimant has to calculate the response to the challenge. It also prepares a session key, and encrypts the session key and the next in the hash series with the public key of the verifier. Then the verifier gets that stuff and checks it. I have taken quite a lot of care that this should be denial of service resistant in the sense that all the operations that we're doing initially are cheap operations. We do the expensive operations only at the end. This protocol only protects verifiers, it doesn't take care of protecting the claimant.

There is still one interesting thing that might be added to this protocol. It's the idea that several people have published in the last couple of years, of giving a puzzle to a claimant and saying, before I talk to you, you have to solve this puzzle so that I know that you have done some work before I will do some work. In the case of IPv6 we could have an implicit puzzle. We both know what the time is right now, and your challenge, because you know this information already before you send any packets to me, is to generate a number so that when you

calculate a hash over this concatenation[3], there are a number of zeroes in the end or beginning of the hash value. You have to generate random numbers until you get the specified number of bits. Maybe we could set the default number of zeroes to four zeroes, or something like that, so that you don't need to try that many random numbers before you solve the puzzle. And then you send your idea of the current time and the random number that you generated, and the verifier checks that the time is relatively recent and there are the required number of zeroes in the hash value.

The idea here is that if you are under an attack you could revert to a protocol where you don't just accept the default puzzle but you immediately just send harder puzzles that say, this is my idea of the time (it doesn't matter whether it's current or whether it's in the future) and I want to have eight zeroes or twelve zeroes instead of four zeroes right now because I'm under attack. In that way you could require the client to come back to the server when the server isn't under attack, so that you can protect the server better in that case. In the default case you just make the client do some work to compensate for the hashes that the server has to do when the client connects.

So in IPv6 we have a bunch of new security problems. Some of them are by design, because there have been designed mechanisms where you can change the routing information of your peer. Mobile IPv6 binding updates is the principle one of them, but there are others as well. Part of it is because you allow the hosts to select the addresses themselves — that's different from what we had before. Part of it is because a host doesn't have just a single address anymore — it's going to have several even though it only has one link. The ideas and ingredients that might be used are these: using the host ID as a crypto token; using a one-time password mechanism; using random addresses; and relying on the routing infrastructure.

OK. Any comments or questions, or do you want to go back to the actual protocol and shoot it down?

John Ioannidis: We've talked about this in the past but I'm still not sure I understand why future address stealing is a problem. If I have a host that shows up somewhere and talks to the server, presumably the address it has is a care of address, and it gives a binding update to the server, so the server will use the care of address to talk to the real address which is back at home.

I'm not disputing the fact that there is a problem with stateless configuration, but if the address I'm using is a care of address, isn't the server supposed to go back to the home agent? So if the real owner shows up again and gets the same care of address, the server should still go back to the home agent, which knows somebody else is using that address and is not going to allow it.

Greg O'Shea: You send the binding update in response to a packet from the server. It will never get the packet from the server because it's sent it off to a bogus care of address, the one that the attacker installed.

Reply: In the address ownership problem, the attacker comes to the link where the victim will be eventually, and it will use the victim's address as its

[3] The source address, the destination address, the current time, and a random number.

home address. It doesn't require a home agent here, it just uses the address as it's home address, and because it's at home at the time it can easily set up a security association between the attacker and the correspondent. It can send a binding update that creates a binding on the correspondent host, saying everything sent to that address should be forwarded to this address, and then the attacker goes there, or maybe doesn't if it's doing a denial of service attack. The victim comes by and starts to use the address. This happens if the victim is using its MAC address for creating its care of address. Nobody says today that you shouldn't do that.

John Ioannidis: OK, and then of course even if you could send the binding updates securely it doesn't help you. On the other hand this is an insider problem. In order to really claim that you have an address in the first place you have to be in the inside network.

Reply: You have to be on this link. Let's take an example, I just happen to be devious against my current employer and I find out what Kurt Hellstrom is doing every morning. I find out that he's using his 3G terminal every morning from his car using a particular base station that happens to be near the road that he takes from his home to his workplace. Then I go there one morning at 5 am and create this kind of binding to the server that he uses to read his news about financial happenings last night. Maybe I could provide him with some wrong information, and get interesting attacks on the stock market because of his reactions.

John Ioannidis: No, you won't be able to do that because if that's a problem he'd be using a secure tunnel from his address to the server, for which you won't have the key. So at worst what you're doing is you're denying service, but you're not really disclosing content.

Reply: It depends on the setting, of course.

Michael Roe: I have a question about the model of the attacker. We both developed cryptographic protocols which were broadly similar but different in some respects. Looking at them I realise we were making different assumptions about what the attacker would be doing. I was assuming that other hosts on the same link are trustworthy, and I rely on them not to do denial of service attacks on me. If they do denial of service attacks on me I will go and physically find them down the hall and tell them not to do it again. What the crypto is doing is protecting me against denial of service attacks from people the other side of the router. Looking at what you were doing, it seemed as though you also wanted to protect against local denial of service attacks.

Reply: Yes, as I tried to say in the beginning, my assumption is that we are doing ad-hoc networking. You might have something like a public wireless LAN base station. When you have a public wireless LAN access point where anybody can connect, and you want to use IPv6, then you have this problem. That's one of my assumptions, how wireless LAN might be used in the future. If you go to the battlefield you definitely want to use something like that because on the battlefield you can't prevent the enemy launching denial of service attacks on you.

Efficient, DoS-Resistant, Secure Key Exchange for Internet Protocols

William Aiello[1], Steven M. Bellovin[1], Matt Blaze[1], Ran Canetti[2], John Ioannidis[1], Angelos D. Keromytis[3], and Omer Reingold[1]

[1] AT&T Labs – Research, {aiello,smb,mab,ji,omer}@research.att.com
[2] IBM T.J. Watson Research Center, canetti@watson.ibm.com
[3] Columbia University in the City of New York, angelos@cs.columbia.edu

1 Introduction

Many public-key-based key setup and key agreement protocols already exist and have been implemented for a variety of applications and environments. Several have been proposed for the IPsec protocol, and one, IKE [1], is the current standard. IKE has a number of deficiencies, the three most important being that the number of rounds is high, that it is vulnerable to denial-of-service attacks, and the complexity of its specification. (This complexity has led to interoperability problems, so much so that, several years after its initial adoption by the IETF, there are still completely non-interoperating implementations.)

While it may be possible to "patch" the protocol to fix some of these problems, we would prefer to replace IKE with something better. With that in mind, we set out to engineer a new key exchange protocol specifically for Internet security applications. With a view toward its possible role as a successor to IKE, we call our new protocol "JFK," which stands for "Just Fast Keying."

1.1 Design Goals

We seek a protocol with the following characteristics:

Security: No one other than the participants may have access to the generated key.
Simplicity: It must be as simple as possible.
Memory-DoS: It must resist memory exhaustion attacks on the responder.
Computation-DoS: It must resist CPU exhaustion attacks on the responder.
Privacy: It must preserve the privacy of the initiator.
Efficiency: It must be efficient with respect to computation, bandwidth, and number of rounds.
Non-Negotiated: It must avoid complex negotiations over capabilities.
PFS: It must approach perfect forward secrecy.

The *Security* property is obvious enough; the rest, however, require some discussion.

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 27–39, 2002.

The *Simplicity* property is motivated by several factors. Efficiency is one; increased likelihood of correctness is another. But our motivation is especially colored by our experience with IKE [1]. Even if the protocol is defined correctly, it must be implemented correctly; if a protocol definition is too complex, implementors will get it wrong. This hinders both security and interoperability.

The *Memory-DoS* and *Computation-DoS* properties have become more important in the context of recent Internet denial-of-service attacks. Photuris [2] was the first published key management protocol for which DoS-resistance was a design consideration; we suggest that these properties are at least as important today.

The *Privacy* property means that the protocol must not reveal the initiator's identity to any unauthorized party, including an active attacker that attempts to act as the responder. Protecting the responder's privacy does not appear to be of much value, except perhaps in peer-to-peer communication: in many cases, the responder is a server with a fixed address or characteristics (*e.g.,* well-known web server). A third approach is to allow for a protocol that allows the two parties to negotiate who needs identity protection. In JFK, we decided against this approach: it is unclear what, if any, metric can be used to determine which party should receive identity protection; furthermore, this negotiation could act as a loophole to make initiators reveal their identity first.

The *Efficiency* property is worth discussing. In many protocols, key setup is must be performed frequently enough that it can become a bottleneck to communication. The key exchange protocol must minimize both computation as well total bandwidth and round trips. Round trips can be an especially important factor over unreliable media.

The *Non-Negotiated* property is necessary for several reasons. The first, of course, is as a corollary to *Simplicty* and *Efficiency.* Negotiations create complexity and round trips, and hence should be avoided. Denial of service resistance is also relevant here; a partially-negotiated security association is consuming resources.

The *PFS* property is perhaps the most controversial. Rather than assert that "we must have perfect forward secrecy at all costs", we treat the *amount* of forward secrecy as an engineering parameter that can be traded off against other necessary functions, such as resistance to denial-of-service attacks. In fact, this corresponds quite nicely to the reality of today's Internet systems, where a compromise during the existence of a security association will reveal the plaintext of any ongoing transmissions. Our scheme has the concept of a "forward secrecy interval"; associations are protected against compromises that occur outside of that interval.

Protocol design is, to some extent, an engineering activity, and we need to provide for trade-offs between different types of security. There are trade-offs that we made during the protocol design, and others, such as the trade-off between forward secrecy and computational effort, that are left to the implementation and to the user, *e.g.,* selected as parameters during configuration and session negotiation.

2 Protocol Definition

2.1 Notation

First, some notation:

$\{M\}_k$	Encryption of message M with symmetric key k.
$H_k(M)$	Keyed hash (e.g., HMAC [3]) of message M using key k.
$S_x[M]$	Digital signature of message M with the private key belonging to principal x (Initiator or Responder). It is assumed to be a non-message-recovering signature.

The message components used in JFK are:

g^x	Diffie-Hellman exponentials; also identifying the group-ID.
g^i	Initiator's current exponential.
g^r	Responder's current exponential.
N_I	Initiator nonce, a random bit-string.
N_R	Responder nonce, a random bit-string.
sa	defines cryptographic and service properties of the security association (SA) that the Initiator wants to establish. It contains a Domain-of-Interpretation which JFK understands, and an application-specific bitstring.
sa′	SA information the Responder may need to give to the Initiator (*e.g.,* the Responder's SPI in IPsec).
HK_r	A transient hash key private to the Responder.
K_{ir}	shared key derived from g^{ir}, N_I, and N_R used for protecting the application (*e.g.,* IPsec SA).
K_e	shared key derived from g^{ir}, N_I, and N_R used to protect messages 3 and 4 of the protocol.
ID_I	Initiator's certificates or public-key identifying information.
ID_R	Responder's certificates or public-key identifying information.
$\mathrm{grpinfo}_R$	all groups supported by the Responder, the symmetric algorithm used to protect messaged 3 and 4, and the hash function used for key generation.

Both parties must pick a fresh nonce at each invocation of the JFK protocol. The nonces are used in the session key computation in order to provide key independence when one or both prties reuse their Diffie-Hellman exponential; the session key will be different different between independent runs of the protocol, as long as one of the nonces or exponentials changes.

HK_R is a global parameter for the Responder – it stays the same between protocol runs, but it can change periodically. The Responder must bick a new g^r every time HK_R changes.

2.2 The Protocol

The basic JFK protocol consists of four messages, for a total of two round trips.

$$I \to R : N_I, g^i \pmod p \tag{1}$$

$$R \to I : N_I, N_R, g^r, \text{grpinfo}_R, \text{ID}_R, S_R[g^r], H_{\text{HK}_R}(N_I, N_R, g^i, g^r) \tag{2}$$

$$I \to R : N_I, N_R, g^i, g^r, H_{\text{HK}_R}(N_I, N_R, g^i, g^r), \{\text{ID}_I, \text{sa}, S_I[N_I, N_R, g^i, g^r, \text{ID}_R, \text{sa}]\}_{K_e} \tag{3}$$

$$R \to I : \{S_R[N_I, N_R, g^i, g^r, \text{ID}_I, \text{sa}, \text{sa}'], \text{sa}'\}_{K_e} \tag{4}$$

The key used to protect Messages (3) and (4), K_e, is computed as $H_{g^{ir}}(N_I, N_R, 1)$. The session key used by IPsec (or any other application), K_{ir}, is $H_{g^{ir}}(N_I, N_R, 0)$.

Message (1) is straightforward; note that it assumes that the Initiator already knows a group and generator that is acceptable to the Responder. The Initiator can reuse a g^i value in multiple instances of the protocol with the Responder, or other responders that accept the same group, for as long as she wishes her forward secrecy interval to be. We discuss how the Initiator can discover what groups to use later.

Message (2) is more complex. Assuming that the Responder accepts the Diffie-Hellman group in the Initiator's message (rejections are discussed in Section 2.4), he replies with a signed copy of his own exponential (in the same group, also $\pmod p$), information on what secret key algorithms are acceptable for the next message, a random nonce, his identity (certificates or a string identifying his public key), and an authenticator calculated from a secret, HK_R, known to the Responder; the authenticator is computed over the two exponentials and nonces. The authenticator key is changed at least as often as g^r, thus preventing replays of stale data. The Responder's exponential may also be reused; again, it is regenerated according to the Responder's forward secrecy interval. The signature on the exponential needs to be calculated at the same rate as the Responder's forward secrecy interval (when the exponential itself changes). Finally, note that the Responder does not need to generate any state at this point, and the only expensive operation is a MAC calculation.

Message (3) echoes back the data sent by the Responder, including the authenticator. The authenticator is used by the Responder to verify the authenticity of the returned data. The message also includes the Initiator's identity and service request, and a signature computed over the nonces, the Responder's identity, and the two exponentials. This latter information is all encrypted under a key derived from the Diffie-Hellman computation and the nonces N_I and N_R. The encryption and authentication use algorithms specified in grpinfo_R. The Responder keeps a copy of recently-received Message (3)s, and their corresponding Message (4). Receiving a duplicate (or replayed) Message (3) causes the Responder to simply retransmit the corresponding Message (4), without creating new state or invoking IPsec. This cache of messages can be reset as soon as g^r or HK_R are changed. The Responder's exponential (g^r) is re-sent by the Initiator because the Responder may be generating a new g^r for every new JFK protocol

run (*e.g.,* if the arrival rate of requests is below some threshold). It is important that the responder deal with repeated Message (3)s as described above. Responders that create new state for a repeated Message (3) open the door to attacks against the protocol.

Note that the signature is protected by the encryption. This is necessary, since everything signed is public except the sa, and that is often guessable. An attacker could verify guesses at identities if the signature were not encrypted.

Message (4) contains application-specific information (such as the Responder's IPsec SPI), and a signature on both nonces, both exponentials, and the Initiator's identity. Everything is encrypted by a K_e, which is derived from N_I, N_R, and g^{ir}. The encryption algorithm is specified in grpinfo_R.

2.3 Discussion

The design follows from our requirements. With respect to communication efficiency, observe that the protocol requires only two round trips. The protocol is optimized to protect the Responder against denial of service attacks on state or computation. The Initiator bears the initial computational burden and must establish round-trip communication with the Responder before the latter is required to perform expensive operations. At the same time, the protocol is designed to limit the private information revealed by the Initiator; she does not reveal her identity until she is sure that only the Responder can retrieve it. (An active attacker can replay an old Message (2) as a response to the Initiator's initial message, but he cannot retrieve the Initiator's identity from Message (3) because he cannot complete the Diffie-Hellman computation).

The Initiator's initial message, Message (1), is a straightforward Diffie-Hellman exponential. Note that this is assumed to be encoded in a self-identifying manner, *i.e.,* it contains a tag indicating which modulus and base was used. The nonce N_I serves two purposes: first, to allow the Initiator to reuse the same exponential across different sessions (with the same or different Responders, within the Initiator's forward secrecy interval) while ensuring that the resulting session key will be different. Secondly, it can be used to differentiate between different parallel sessions.

Message (2) must require only minimal work for the Responder, since at that point he has no idea whether the Initiator is a legitimate correspondent or, *e.g.,* a forged message from an denial of service attack; no round trip has yet occurred with the Initiator. Therefore, it is important that the Responder not be required at this point to perform expensive calculations or create state. Here, the Responder's cost will be a single authentication operation, the cost of which (for HMAC) is dominated by two invocations of a cryptographic hash function, plus generation of a random nonce N_R.

The Responder *may* compute a new exponential $g^b \pmod p$ for each interaction. This is an expensive option, however, and at times of high load (or attack) it would be inadvisable. The nonce prevents two successive session keys from being the same, even if both the Initiator and the Responder are reusing exponentials.

If the Responder is willing to accept the group identified in the Initiator's message, his exponential must be in the same group. Otherwise, he may respond with an exponential from any group of his own choosing. The field grpinfo_R lists what groups the Responder finds acceptable, if the Initiator should wish to restart the protocol. This provides a simple mechanism for the Initiator to discover the groups currently allowed by the Responder. That field also lists what encryption algorithm is acceptable for the next message. This is not negotiated; the Responder has the right to decide what strength encryption is necessary to use his services.

Note that the Responder creates no state when sending this message. If it is fraudulent — that is, if the Initiator is non-existent or intent on perpetrating a denial-of-service attack — the Responder will not have committed any storage resources.

In Message (3), the Initiator echoes content from the Responder's message, including the authenticator. The authenticator allows the Responder to verify that he is in round-trip communication with a legitimate potential correspondent. The Initiator also uses the key derived from the two exponentials and the two nonces to encrypt her identity and service request. (The Initiator's nonce is used to ensure that this session key is unique, even if both the Initiator and the Responder are reusing their exponentials and the Responder has "forgotten" to change nonces.) The key used to protect Messages (3) and (4) is computed as $H_{g^{ir}}(N_I, N_R, 1)$. The session key used by IPsec (or any other application) is $H_{g^{ir}}(N_I, N_R, 0)$.

Because the Initiator can validate the Responder's identity before sending her own and because her identifying information (ignoring her public key signature) is sent encrypted, her privacy is protected from both passive and active attackers. (An active attacker can replay an old Message (2) as a response to the Initiator's initial message, but he cannot retrieve the Initiator's identity from Message (3) because he cannot complete the Diffie-Hellman computation.) The service request is encrypted, too, since disclosure of it might identify the requester. The Responder may wish to require a certain strength of cryptographic algorithm for certain services.

Upon successful receipt and verification of this message, the Responder has a shared key with a party known to be the Initiator. The Responder further knows what service the Initiator is requesting. At this point, he may accept or reject the request.

The Responder's processing on receipt of Message (3) requires verifying an authenticator and, if that is successful, performing several public key operations to verify the Initiator's signature and certificate chain. The authenticator (again requiring two hash operations) is sufficient defense against forgery; replays, however, could cause considerable computation. The defense against this is to cache the corresponding Message (4); if a duplicate Message (3) is seen, the cached response is retransmitted; the Responder does not create any new state or notify the application (*e.g.,* IPsec).

Caching Message (3) and refraining from creating new state for replayed instances of Message (3) serves also another security purpose. If the Responder were to create a new state and send a new Message (4), and a new sa′ for a replayed Message (3), then an attacker that compromised the Initiator could replay a recent session with the Responder. That is, by replaying Message (3) from a recent exchange between the Initiator and the Responder, the attacker could establish a session with the Responder where the session-key is identical to the key of the previous session (which took place when the Initiator was not yet compromised). This could compromise the Forward Security of the Initiator.

There is a risk, however, in keeping this message cached for too long: if the Responder's machine is compromised during this period, perfect forward secrecy is compromised. We can tune this by changing the MAC key HK_R more frequently. The cache can be reset when a new g^r or HK_R is chosen.

In Message (4), the Responder sends to the Initiator any Responder-specific application data (*e.g.,* the Responder's IPsec SPI), along with a signature on both nonces, both exponentials, and the Initiator's identity. All the information is encrypted using a key derived the two nonces, N_I and N_R, and the Diffie-Hellman result. The Initiator can verify that the Responder is present and participating in the session, by decrypting the message and verifying the enclosed signature.

2.4 Rejection Messages

Instead of sending messages (2) or (4), the Responder can send a rejection instead. For Message (2), this rejection can only be on the grounds that he does not accept the group that the Initiator has used for her exponential. Accordingly, the reply should indicate what groups are acceptable. (For efficiency's sake, this information could also be in the Responder's long-lived certificate, which the Initiator may already have.)

Message (4) can be a rejection for several reasons, including lack of authorization for the service requested. But it could also be caused by the Initiator requesting cryptographic algorithms that the Responder regards as inappropriate, given the requester (Initiator), the service requested, and possibly other information available to the Responder, such as the time of day or the Initiator's location as indicated by the network. In these cases, the Responder's reply should list acceptable cryptographic algorithms, if any. The Initiator would then send a new Message (3), which the Responder would accept anew; again, the Responder does not create any state until after a successful Message (3) receipt.

3 What JFK Avoids

By intent, JFK does not do certain things. It is worth enumerating them, if only to forestall later attempts to add them in. The "missing" items were omitted by design, in the interests of simplicity.

The most obvious "omission" is any form of authentication other than certificate chain trusted by the each party. We make no provisions for shared secrets,

token-based authentication, certificate discovery, or explicit cross-certification of PKIs. In our view, these are best accomplished by outboard protocols. Initiators that wish to rely on any form of legacy authentication can use the protocols being defined by the IPSRA [4] or SACRED [5,6] working groups. While these mechanisms do add extra round trips, the expense can be amortized across many JFK negotiations. Similarly, certificate chain discovery (beyond the minimal capabilities implicit in ID_I and ID_R) should be accomplished by protocols defined for that purpose. By excluding the protocols for JFK, we can exclude them from our security analysis; the only interface between the two is a certificate chain, which by definition is a stand-alone secure object.

We also eliminate negotiation, in favor of ukases issued by the Responder. The responder is providing a service; it is entitled to set its own requirements for that service. Any cryptographic primitive mentioned by the Responder is acceptable; the Initiator can choose any it wishes. We thus eliminate complex rules for selecting the "best" choice from two different sets. We also eliminate state to be kept by the Responer; the Iniator can either accept the Responder's desires or restart the protocol.

Finally, we reject the notion of two different phases. As noted, the practical benefits of quick mode are limited. Furthermore, we do not agree that frequent rekeying is necessary. If the underlying block cipher is sufficiently limited as to bar long-term use of any one key, the proper solution is to replace that cipher. For example, 3DES is inadequate for protection of very high speed transmissions, because the probability of collision in CBC mode becomes too high after less than encryption of 2^{32} plaintext blocks. Using AES instead of 3DES solves that problem without complication the key exchange.

4 Related Work

4.1 Internet Key Exchange (IKE)

The Internet Key Exchange protocol (IKE) [1] is the current IETF standard for key establishment and SA parameter negotiation. IKE is based on the ISAKMP [7] framework, which provides encoding and processing rules for a set of payloads commonly used by security protocols, and the Oakley protocol, which describes an adaptation of the StS protocol for use with IPsec.

IKE is a two-phase protocol: during the first phase, a secure channel between the two key management daemons is established. Parameters such as an authentication method, encryption/hash algorithms, and a Diffie-Hellman group are negotiated at this point. This set of parameters is called a "Phase 1 SA." Using this information, the peers authenticate each other and compute key material using the Diffie-Hellman algorithm. Authentication can be based on public key signatures, public key encryption, or preshared passphrases. There are efforts to extend this to support Kerberos tickets [8] and handheld authenticators. It

should also be noted that IKE can support other key establishment mechanisms (besides Diffie-Hellman), although none has been proposed yet[1].

Furthermore, there are two variations of the Phase 1 message exchange, called "main mode" and "aggressive mode." Main mode provides identity protection, by transmitting the identities of the peers encrypted, at the cost of three message round-trips. Aggressive mode provides somewhat weaker guarantees, but requires only three messages.

As a result, aggressive mode is very susceptible to untraceable[2] denial of service (DoS) attacks against both computational and memory resources [9]. Main mode is also susceptible to untraceable memory exhaustion DoS attacks, which must be compensated for in the implementation using heuristics for detection and avoidance. In particular:

- The Responder has to create state upon receiving the first message from the Initiator, since the Phase 1 SA information is exchanged at that point. This allows for a DoS attack on the Responder's memory, using random source IP addresses to send a flood of requests. To counter this, the Responder could employ mechanisms similar to those employed in countering TCP SYN attacks [10,11,12]. JFK avoids maintaining state at all as a result of receiving the first message.
- An Initiator who is willing to go through the first message round-trip (and thus identify her address) can cause the Responder to do a Diffie-Hellman exponential generation as well as the secret key computation on reception of the third message of the protocol. The Initiator could do the same with the fifth message of the protocol, by including a large number of bogus certificates, if the Responder blindly verifies all signatures. JFK mitigates the effects of this attack by reusing the same exponential across different sessions.

The second phase of the IKE protocol is commonly called "quick mode" and results in IPsec SAs established between the two negotiating parties, through a three-message exchange. Parameters such as the IP security protocol to use (ESP/AH), security algorithms, the type of traffic that will be protected, *etc.* are negotiated at this stage. Since the two parties have authenticated each other and established a shared key during Phase 1, quick mode messages are encrypted and authenticated using that information. Furthermore, it is possible to derive the IPsec SA keying material from the shared key established during the Phase 1 Diffie-Hellman exchange. To the extent that multiple IPsec SAs between the same two hosts are needed, this two-phase approach results in faster and more lightweight negotiations (since the same authentication information and keying material is reused).

Unfortunately, two hosts typically establish SAs protecting all the traffic between them, limiting the benefits of the two-phase protocol to lightweight

[1] There is ongoing work (still in its early stages) in the IETF to use IKE as a transport mechanism for Kerberos tickets, for use in protecting IPsec traffic.

[2] The attacker can use a forged address when sending the first message in the exchange.

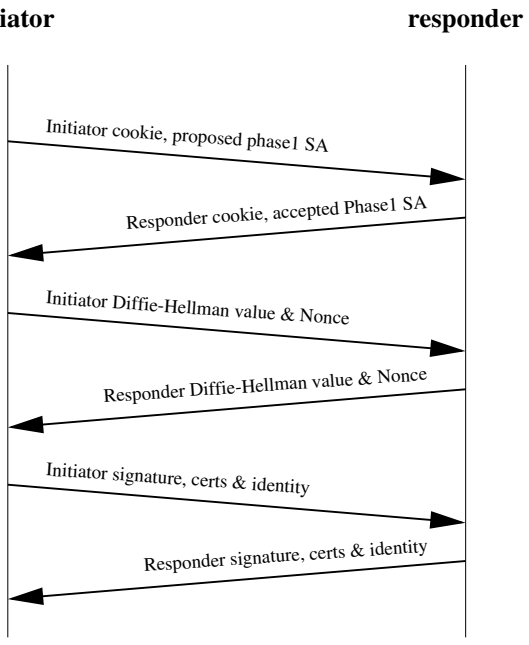


Fig. 1. IKE Main Mode exchange with certificates.

re-keying. If "Perfect Forward Secrecy" (PFS) is desired, this benefit is further diluted. (PFS is an attribute of encrypted communications allowing for a long-term key to be compromised without affecting the security of past session keys.)

Another problem of the two-phase nature of IKE manifests itself when IPsec is used for fine-grained access control to network services. In such a mode, credentials exchanged in the IKE protocol are used to authorize users when connecting to specific services. Here, a complete Phase 1 & 2 exchange will have to be done for each connection (or, more generally, traffic class) to be protected, since credentials, such as public key certificates, are only exchanged during Phase 1.

IKE protects the identities of the Initiator and Responder from eavesdroppers[3]. The "identities" include public keys, certificates, and other information that would allow an eavesdropper to determine which principals are trying to communicate. These identities can be independent of the IP addresses of the IKE daemons that are negotiating (*e.g.,* temporary addresses acquired via DHCP, public workstations with smartcard dongles, *etc.*). However, since the Initiator reveals her identity first (in message 5 of Main Mode), an attacker can pose as the Responder until that point in the protocol. The attacker cannot complete the protocol (since they do not possess the Responder's private key), but they can determine the Initiator's identity. This attack is not possible on the Responder,

[3] Identity protection is provided only in Main Mode (also known as Identity Protection Mode); Aggressive Mode does not provide Identity Protection for the Initiator.

since she can verify the identity of the Initiator before revealing her identity (in message 6 of Main Mode). However, since most Responders would correspond to servers (firewalls, web servers, *etc.*), the identity protection provided to them seems not as useful as protecting the Initiator's identity[4]. Fixing the protocol to provide identity protection for the Initiator would involve reducing it to 5 messages and having the Responder send the contents of message 6 in message 4, with the positive side-effect of reducing the number of messages, but breaking the message symmetry and protocol modularity.

Finally, thanks to the desire to support multiple authentication mechanisms and different modes of operation (Aggressive *vs.* Main mode, Phase 1 / 2 distinction), both the protocol specification and the implementations tend to be bulky and fairly complicated. These are undesirable properties for a critical component of the IPsec architecture.

[13] points out many deficiencies in the IKE protocol, specification, and implementation. It suggests removing several features of the protocol (*e.g.,* aggressive mode, public key encryption mode, *etc.*), restore the idea of stateless cookies, and protect the Initiator's (instead of the Responder's) identity from an active attacker. It also suggests some other features, such as one-way authentication (similar to what is common practice when using SSL [14,15] on the web). These major modifications would bring the IKE protocol closer to JFK, although they would not completely address the DoS issues.

A measure of the complexity of IKE can be found in the analyses done in [16, 17]. No less than 13 different sub-protocols are identified in IKE, making understanding, implementation, and analysis of IKE challenging. While the analysis did not reveal any attacks that would compromise the security of the protocol, it did identify various potential attacks (DoS and otherwise) that are possible under some *valid* interpretations of the specification and implementation decisions.

4.2 Other Protocols

The predecessor to IKE, Photuris [2], first introduced the concept of cookies to counter "blind" denial of service attacks. The protocol itself is n 6-message variation of the Station to Station protocol. It is similar to IKE in the message layout and purpose, except that the SA information has been moved to the third message. For re-keying, a two-message exchange can be used to request a unidirectional SPI (thus, to completely re-key, 4 messages are needed). Photuris is vulnerable to the same computation-based DoS attack as IKE, mentioned above.

SKEME [18] shares many of the requirements for JFK, and many aspects of its design were adopted in IKE. It serves more as a set of protocol building blocks, rather than a specific protocol instance. Depending on the specific requirements for the key management protocol, these blocks could be combined in several ways. As a result of this modularization, both the number of round-trips and

[4] One case where protecting the Responder's identity can be more useful is in peer-to-peer scenarios.

the optional payloads and exchanges is quite high. The latter has a direct impact on the implementation complexity (as seen in IKE itself). Another interesting aspect of SKEME is its avoidance of digital signatures; public key encryption is used instead, to provide authentication assurances. The reason behind this was to allow both parties of the protocol to be able to repudiate the exchange.

SKIP [19] was an early proposal for an IPsec key management mechanism. It uses long-term Diffie-Hellman public keys to derive long-term shared keys between parties, which is used to distribute session keys between the two parties. The distribution of the session key occurs in-band, *i.e.,* the session key is encrypted with the long-term key and is injected in the encrypted packet header. While this scheme has good synchronization properties in terms of re-keying, it lacks any provision for PFS. Furthermore, there is no identity protection provided, since the certificates used to verify the Diffie-Hellman public keys are (by design) publicly available, and the source/destination master identities are contained in each packet (so a receiver can retrieve the sender's Diffie-Hellman certificate). The latter can be used to mount a DoS attack on a receiver, by forcing them to retrieve and verify a Diffie-Hellman certificate, and then compute the Diffie-Hellman shared secret.

References

1. Harkins, D., Carrel, D.: The Internet Key Exchange (IKE). Request for Comments (Proposed Standard) 2409, Internet Engineering Task Force (1998)
2. Karn, P., Simpson, W.: Photuris: Session-key management protocol. Request for Comments 2522, Internet Engineering Task Force (1999)
3. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: keyed-hashing for message authentication. Request for Comments 2104, Internet Engineering Task Force (1997)
4. Sheffer, Y., Krawczyk, H., Aboba, B.: PIC, a pre-IKE credential provisioning protocol. Internet Draft, Internet Engineering Task Force (2001) Work in progress.
5. Arsenault, A., Farrell, S.: Securely available credentials – requirements. Request for Comments 3157, Internet Engineering Task Force (2001)
6. Gustafson, D., Just, M., Nystrom, M.: Securely available credentials - credential server framework. Internet Draft, Internet Engineering Task Force (2001) Work in progress.
7. Maughan, D., Schertler, M., Schneider, M., Turner, J.: Internet security association and key management protocol (ISAKMP). Request for Comments (Proposed Standard) 2408, Internet Engineering Task Force (1998)
8. Miller, S.P., Neuman, B.C., Schiller, J.I., Saltzer, J.H.: Kerberos Authentication and Authorization System. Technical report, MIT (1987)
9. Simpson, W.A.: IKE/ISAKMP Considered Harmful. USENIX ;login: (1999)
10. Heberlein, L., Bishop, M.: Attack Class: Address Spoofing. In: Proceedings of the 19th National Information Systems Security Conference. (1996) 371–377
11. CERT: Advisory CA-96.21: TCP SYN Flooding. ftp://info.cert.org/pub/cert_advisories/CA-96.21.tcp_syn_flooding (1996)
12. Schuba, C., Krsul, I., Kuhn, M., Spafford, E., Sundaram, A., Zamboni, D.: Analysis of a denial of service attack on tcp. In: IEEE Security and Privacy Conference. (1997) 208–223

13. Kaufman, C., et al.: Code-preserving Simplifications and Improvements to IKE. Internet Draft, Internet Engineering Task Force (2001) Work in progress.
14. Hickman, K.: Secure Socket Library (SSL). http://home.netscape.com/security/techbriefs/ssl.html (1995)
15. Dierks, T., Allen, C.: The TLS protocol version 1.0. Request for Comments (Proposed Standard) 2246, Internet Engineering Task Force (1999)
16. Meadows, C.: Analysis of the Internet Key Exchange protocol using the NRL protocol analyzer. In: Proc. of the 1999 IEEE Symposium on Security and Privacy. (1999) 216–231
17. Meadows, C.: Open issues in formal methods for cryptographic protocol analysis. In: Proc. of DARPA Information Survivability Conference and Exposition (DISCEX 2000), IEEE Computer Society Press (2000) 237–250
18. Krawczyk, H.: SKEME: A Versatile Secure Key Exchange Mechanism for Internet. In: Proc. of Network and Distributed System Security Symposium (NDSS). (1996)
19. Aziz, A., Patterson, M.: Simple Key Management for Internet Protocols (SKIP. In: Proc. of the 1995 INET conference. (1995)

Efficient, DoS-Resistant, Secure Key Exchange for Internet Protocols (Transcript of Discussion)

Matt Blaze

AT&T Labs Research

This talk is political, but not in quite the same political domain as its title may suggest. This is joint work with my colleagues Steve Bellovin and John Ioannidis at AT&T Labs and Angelos Keromytis at the University of Pennsylvania. We are just about done with a first cut of a protocol that I'd like to talk about it, get feedback on, and discuss the requirements for such a protocol.

We're concerned with the problem of a secure key exchange protocol, particularly aimed at IPSec, but for Internet applications in general. The currently standardised protocol is called IKE, which stands for Internet Key Exchange. We decided this was an American-centric pun on the name Ike, which was the nickname of President Eisenhower, who had the slogan "I like Ike". We don't like IKE, so we'd like to see a successor to IKE. We call our protocol JFK, which we claim stands for "Just Fast Keying", but is also the initials of a president who succeeded Eisenhower for some amount of time. We're hoping not to ever discuss the protocol in Dallas. If there's ever an IETF in Dallas again, we're not going to mention our protocol at all there.

So, what's the problem? Fundamentally, IKE is an evil, foul smelling abomination against nature and all that is good and pure. In particular, it is complex and expensive. The complexity is to a point where years after its specification was published, there are still no interoperable implementations of the complete protocol. It's expensive in the sense that there are gratuitous exponentiations required. It forces the responder, the server, to be vulnerable to particularly many denial of service attacks. One possible approach is to look at the specific problems with IKE and try to fix them. We could potentially find the various denial of service attacks and, for example, add mechanisms to make those specific attacks less attractive, but that would invariably make the complexity aspects of IKE worse. We'd like to start from scratch, have a clear sense of what the requirements for such a protocol are, and base our design on the requirements, rather than following IKE's design process, identifying new requirements as we go along and patching them. We think this is where a lot of the complexity came from.

So we have our new protocol, or at least our new set of requirements for a protocol, that we call Just Fast Keying. The goal of Just Fast Keying is secure key establishment based on a Diffie-Hellman like underlying security mechanism, that is particularly suited to IPSec, but can be used for various Internet protocols.

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 40–48, 2002.

The requirements that we would like to see in include: Security — the key should only be known to the initiator and responder at the end of the protocol. That's the obvious straightforward security requirement.

Initiator privacy. Initiator privacy requires a little bit of discussion. We would like the initiator's identity to be kept secret from anyone other than the responder that the initiator intended to communicate with. Because IPSec keys might not be bound specifically to the IP address, there may not be information revealed about who the participants are just by virtue of the fact that they're communicating, and we'd like the key agreement protocol not to gratuitously leak new information about who the initiator is. It turns out that IKE can be configured in a way that preserves initiator privacy, but we have some other desirable properties that we'd like to see.

We'd like it to be simple enough that a person can understand what it does both for the purpose of having confidence in its security and so forth, and also for the purpose of producing a correct implementation.

We would like the protocol to be efficient, particularly with respect to limiting the number of round trips required to initiate communication. To shoehorn it into the theme of this workshop, in wireless applications where networks are unreliable, one of the most important communication costs that one tries reduce is the number of round trips before you can do something. When messages get lost, the more round trips you have the more opportunities you have to be forced to wait for a time-out and so on. We'd like no gratuitous computation required. We'd like not to force people to exponentiate six times every time they want to create a unique key.

We'd like to resist denial of service attacks, or at least not make denial of service attacks on the responder particularly easy. We'd like not to force the responder to create state easily; we'd like there not to be attacks that allow somebody to send a single packet with no valid address at the beginning that causes some state that sticks around for an arbitrarily long time at the responder, and we'd like not to be able to send arbitrary forged packets that induce the responder to do expensive computations that we can then bog them down with.

We'd like the protocol not to include a requirement for elaborate negotiation of parameters and so forth as a necessary part of every exchange, we'd like parameter negotiation to be an implicit part of running the protocol rather than requiring multiple rounds of negotiation. This goes back to the efficiency and simplicity aspects of this. We don't want to include a Swiss army knife approach[1] when that's not necessary.

This might be a bit controversial and strange — we'd like perfect forward secrecy, but it doesn't have to be perfect! We'd like forward secrecy that meets our requirements, I'll discuss what that means a bit later.

Bruce Christianson: Matt, just remind us what you mean by forward secrecy.

Reply: Well that's an excellent question, and again that comes down to the reasonableness. Forward secrecy here means that at the end of the key exchange

[1] LNCS 2133, pp 1–4

protocol neither party has enough information to recreate an old session key except for having the session key itself. We are willing to relax that very strict requirement in a way that we can control and quantify, whereas in IKE there's perfect forward secrecy or not, and you either have no forward secrecy or very expensive.

The basic characteristics of our first cut design are: We're based on Diffie Hellman. We require a total of two round trips, whereas IKE even in the best case requires three. The responder is in complete control of the acceptable parameters, so it is up to the responder to decide whether or not the parameters are acceptable, the initiator can always abort and try again with new ones and the responder might perhaps send hints on what's acceptable. In practice this is precisely the model that people use anyway. The responder need do no hard computation, such as performing an exponentiation, nor create any state, until after the first round trip. In fact the state doesn't have to be created until after you know that you're communicating with somebody who you are, in fact, interested in creating state with. Only the responder learns who the initiator is.

So here's a very preliminary basic idea of how one might go about doing such a thing. I've taken out the (mod p)s here just to make it simpler. Alice, the initiator, sends g^a to Bob, Bob responds with g^b, Alice then responds with her certificate and an authenticator that is bound to g^{ab}. For example, I send you my certificate and a message containing g^{ab}, signed with the certified key. Bob then does the same thing and at the end we have a key that we know belongs to Alice and Bob, with the usual concerns about man in the middle attacks. This is the preliminary basic idea.

The first thing that's wrong with this is we don't want just anyone to be able to see that it's Alice, but in this version of the protocol we're sending her certificate and her identity in the clear. An obvious first refinement on this is to send Alice's identifier encrypted under the session key that's been created, so that now only Bob, who knows b, can figure it out. For good measure we'll only do this after Bob is authenticated, so we'll force Bob to send his ID first. Bob as the responder has to give up privacy to be a responder but Alice the initiator needn't. So that would lead us to a simple refinement. Alice sends g^a to start out. Bob responds not just with g^b but also with his certificate and an authenticator bound to the g^b that's been chosen. Alice now responds with her identifying information encrypted under a secret key derived from g^{ab}, and Bob sends back an OK message if in fact he's willing to communicate under those circumstances.

So this improves it with respect to initiator privacy, and we're still at two round trips. But what about denial of service against Bob? We've effectively engineered a very useful service for conducting denial of service attacks against responders, because the attacker just sends a fake message number 1, a fake g^a, and this forces Bob to do at least two exponentiations, three if he's choosing a new g^b every time. Bob has to compute g^{ab}, he has to choose his g^b and he has to sign the g^b. The attacker, masquerading as Alice, need not even do this from

a real address. Every time Bob receives one of these message 1s, he has to do all of these computations in order to respond with message 2 intelligently.

We can get around this by having Bob pre-compute a g^b at various intervals. Every minute or so, as often as Bob can conveniently do this, Bob calculates a new g^b. Each exchange also includes a unique nonce from each side to make sure that the session key that's ultimately chosen is unique. So Bob pre-computes a g^b, but every time he receives a g^a he responds with his fixed g^b and a unique nonce that's a random number or may be derived from a counter and a hash. We don't have Bob actually perform any exponentiations until after the first round trip has occurred, so that Bob is now convinced that he's actually talking to a real initiator, a real Alice.

What about forward secrecy? Both Alice and Bob can select their exponentials whenever they want. Selecting an exponential costs you two exponentiations — you have to calculate the exponent and then sign it in order to have the authenticator. Until you do that you don't get any forward secrecy — you have to remember your previous exponential, and therefore you can compute past session keys. You can do this selection at fixed intervals so you can make the cost of doing this constant, regardless of how much traffic you're exchanging. If you choose the forward secrecy interval to be less than the amount of time the keys are in fact expected to be used, then you haven't lost any effective forward secrecy — you always know that you're going to throw out your exponential before you throw out the session key that's derived from it. This parameter can be tuned and selected but needn't be published. It needn't be a negotiated parameter, it's simply an implementation parameter that every site can choose for itself.

To protect Bob from denial of service we have this slightly refined version of the protocol. Alice sends out a g^a, Bob responds with his fixed g^b, his signature of g^b, a nonce, which we'll get to in a moment, and an authenticator that is essentially a keyed hash (using a key selected at the same time the exponent b is selected) of the g^a that Alice ostensibly sent, the g^b that is in current use and the nonce that was sent out. Now Alice responds in the third message with a nonce that she selects and an echoing back of this authenticator. What that does is prove to Bob that they are in round trip communication and that Alice not only was the one who sent the g^a but received this message 2. If Alice is satisfied that this g^b really came from the Bob that she wishes to communicate with, she calculates a signature on her exponent bound to this session key, creates a session key based on g^{ab} and the nonces sent her, encrypts her authenticated information under that and sends it on to Bob. Bob now checks message 3 using his secret hash key. If the hash matches he knows that the round trip has occurred and he can safely do exponentiation in the knowledge that he is doing this with a real correspondent. If Alice is somebody Bob is willing to communicate with, then he can then create the state that stores the session key, and sign something based on the nonces and g^{ab}.

So that is what we propose as a useful Internet key exchange protocol that just does fast keying. Now let's be a little bit more ambitious. One of the objec-

tions often raised to IKE and to IPSec in general is that it is very often the case that parties that have communicated once will want to communicate again with one another in the future, but with protocols like IKE every time you want to set up a new key you have to go through at least some exponentiations to set up a key exchange. This may be wasteful. For example, if you're a handset and you want to communicate back with a base station, there's no point in being forced to do a new Diffie-Hellman with the base station every time you register on the network if the mobile unit has remembered a key.

So, if Alice and Bob can remember an old g^{ab} they should be able to create new session keys without doing new exponentiations just by adding new nonces, but the problem with that is that the requirement for both Alice and Bob, both the initiator and the responder, to remember g^{ab} may be not a realistic one. Servers (responders) often can't afford to remember old exponents, and in fact that might not be a desirable property from the forward secrecy point of view. So we propose a possible extension in which each side sends a *blob* (we call it a blob because we're not completely sure what should be in it, although in the paper we suggest a possibility) that contains an encrypted copy of the old g^{ab} along with some authentication information, encrypted using a secret key known to each site that changes at intervals that are frequent enough to allow for reasonable forward secrecy but infrequent enough that the blobs might actually be able to be decrypted in the future.

Now if the client remembers the server's blob, it can re-key as long as the server maintains the key used to decrypt the blob that it had sent. So we might refine the protocol further to include re-key blobs. Alice sends her blob in the third message once she is satisfied that it's really Bob she's speaking with. Bob in the last message can include his blob, that Alice can use. This might lead to the protocol we haven't defined yet, which allows us to use the blobs to re-key. We somewhat jokingly call this RFK for Re-keying Fast Keys — more American-centric political humour. Two of my co-authors are not Americans but they like the names too. In the RFK protocol, instead of message 1, instead of sending a fresh exponential, you first try and send the blob from a previous exchange and a new nonce, and you authenticate all of this under the old key. The server, the responder, upon seeing one of these messages can simply decrypt it. If it's satisfied that this old g^{ab} is in fact one that it sent, and if it still remembers the decryption key, then it can then apply the new nonce and create a new key and acknowledge it, and we can do this without any new exponentiation. Now there are issues of replay and so forth, but it should be doable. It's not completely in the paper.

So there are a number of open issues. Does this meet the requirements? Are the requirements themselves reasonable ones? Is anything else wrong with it? For example, is this as secure as Diffie Hellman? Can we prove anything about it? Finally, if in fact this protocol is better than IKE, is it possible, politically feasible, and technically reasonable to kill off IKE?

Stewart Lee: A very small point. You keep talking about a unique nonce. Is that a tautology, or is it some special kind of nonce, or what? A nonce is, after all, an unpredictable collection of bits.

Reply: In this protocol, we're using nonces for less strong purposes than one normally uses nonces for. We're using nonces on the off chance that both sides are using the same g^{ab} that they've used in the past. We want to ensure that the derived session key is in fact unique.

Stewart Lee: You are taking many things mod p. How do you agree p?

Reply: Yes, I left that out. In general we think of the server as being in control of the group. We're also assuming that all of these messages contain some self-identifying information about what group was chosen, so it identifies what p and g are. These should be known to the server already. If message 1 contains a group that isn't acceptable to the responder it should respond with a message identifying what groups are acceptable. That's implicit negotiation.

Markus Kuhn: In the third message you hash g^a, g^b and N_b. What is the advantage of making a hash as opposed to just repeating N_b?

Reply: This is a hash including a secret known to the responder, so that Alice can't manufacture her own.

Markus Kuhn: Is this so you don't have to store state?

Reply: Yes, that's right, we don't want to have to store nonce N_b or Alice's g^a. On receipt of message 1, Bob doesn't want to have to create any state.

Markus Kuhn: Because g^b lives for a couple of seconds I can bombard you with the the same $H(g^a, g^b, N_b)$ for a couple of seconds.

Reply: That's right, but at most I'll create one bit of state as a result of that, and as long as that state's alive I can recognise the same message and refuse to create new state. So once I've created state, I won't create more state based on the same nonce. We're assuming the model that secret key computations are cheap and what we want to avoid are public key computations.

Bruce Christianson: The use of a blob is the standard technique for making a file server stateless. So what you're trying to do is look for a mechanism for getting write-once semantics?

Reply: Yes, that's right, the difference is in a file system you do this for performance reasons and you can tell if the performance improves. Here we're doing it for security reasons and that's more subtle.

Tuomas Aura: Is there some reason for having g^a in the first message instead of say N_a?

Reply: It could just as well be anything included in and bound to the first message. We chose g^a because that's the essential element of the first message that we want Alice to commit to. I don't see any fundamental reason why it couldn't be something else. You'd have more stuff included in the message, but there's no reason why we couldn't include a nonce in there.

Michael Roe: You took out the N_b from the signed portion of message 2 because otherwise Bob has to resign it for each message from Alice. But that means there's nothing to prevent someone from pretending to be Bob and replaying it.

There ought to be a date in there to say this is the g^b that was generated at this time and it's not a replay.

Reply: Alice can be fooled into sending her traffic using an old exponential from Bob, but because Alice chose a new nonce, even if she's using the same g^a we know that the session key will be unique. If it isn't Bob's current key then no-one will be able to make sense of the traffic.

Michael Roe: I'm worried about an attacker who has stolen an old b.

Reply: Right, if the attacker has stolen an old b, unless the authenticator includes the date, you would be able to do that. We see that as an issue that's outside of the protocol itself. We'd rather push that into the policy with respect to the certificate and authenticator data, which is answering the question, do Alice and Bob accept the credentials that are presented for this key? We think this properly belongs outside of the key agreement protocol itself, but that might be a layering taste issue that you can cut anywhere you want.

Pekka Nikander: I'm not quite sure if you mentioned it already, but can't you just sign the current date with the g^b so that the signature includes the date?

Reply: Yes, as Michael pointed out, for most applications you would want to include things that limit the use of this in some way.

Wenbo Mao: I realise you have a periodical pre-computation to reduce the amount of exponentiation. If the same server is talking to many different Alices, will this create a huge complicated state that we've got to maintain?

Reply: We need only maintain the current g^b of the minute or of the hour of what have you, and once we create a new one we can throw out the old one. So the amount of state we have is bounded regardless of the number of clients involved. Now you do need to create state for each initiator that you want to communicate with, but that's kind of a fundamental property.

Stewart Lee: You anticipate keeping g^b for a while and then changing it. Surely this means that the responder has to keep the new and the old version, and the version before that, and so on?

Reply: One thing you could do is simply throw out the old one and then anybody who was relying on it will fail and get the new one.

Stewart Lee: Doesn't the communication that resulted from the older version depend upon remembering it?

Reply: Right, but it will simply fail and the cost of failure is that you restart. Now if you want to optimise, then a simple windowing protocol where you remember some number of old g^bs and identify which one it is in the message would do. But it's not clear that over-optimising is that helpful. You never *need* to store more than one of these g^bs. If you choose your forward secrecy interval to be at least the expected amount of time of these two round trips then all you have to worry about is the boundary condition in which right here Bob has chosen a new g^b. So if you remember the old g^b and either try both or have them self-identified in some way, then you'll know you'll never have to do new communication as a result of that.

Bruce Christianson: The crucial point there is that each party has the ability to protect itself and control its own position. If Alice is worried about it, then she can update her g^a regularly. Alice has the power to update her g^a as often as she wishes, and that ensures that failure on the part of somebody else to update their credentials cannot destroy Alice's communication with anyone else.

Virgil Gligor: A minor difference from Oakley is the fact that you are caching these g^bs. So consequently you keep a little more state and need a little more protection in the server, whereas in Oakley and IKE you just generate a g^b and then you forget it.

Reply: That's right, we think generating a g^b is far more expensive than remembering a single g^b.

Virgil Gligor: I'm not disputing the performance, clearly you get performance! But you have to be careful that those g^bs don't get out, so you have to protect them a little more than before. That's one difference. The second difference is with perfect forward secrecy — you relaxed it a little bit. I think those are the only two fundamental differences between what you do here and what they did before, because they also have, for example, quick mode.

Reply: The original Oakley didn't have privacy for the initiator.

Virgil Gligor: Oakley had a variant which had privacy, They even had a variant with perfect forward secrecy.

Reply: That's right, although it cost you an extra round trip to get it.

Michael Roe: There's a robustness objection I might have to the protocol. I've often observed when you've got many different implementations of the same protocol, then at least one of them does something that's simpler to implement and looks externally the same. Here the obvious way that can occur is an implementation that keeps g^b the same forever. I would like something Alice can do that gives Alice assurance that this is a new g^b, as long as Bob is merely lazy rather than malicious.

Reply: Right, well one thing Alice can do unilaterally is look and see whether this is the same g^b over a long period of time, or look at the date that's been included in the authentication information. But Alice can ensure that the session key is new simply by including a new nonce. Alice can unilaterally tell that Bob is failing if she sees the same g^b with the same nonce because Bob should never be sending the same nonce twice to a different g^a.

Michael Roe: You could have an agreement that the prime p changes every day, and then to be externally, visibly, conforming to the protocol you have to use the prime of the day and you can't use yesterday's.

Reply: The mechanism for publishing the prime of the day is in message two. Yes, that's a reasonable way to do it. In practise people tend not to choose new parameters that often because it's so expensive. The least often you choose a new group, more often you choose new exponentials, and with every session you choose new nonces.

Wenbo Mao: Then in that case you have a message zero that selects the group.

Reply: No, the message zero is obtained by sending a g^a that may be nonsense and looking at the message that you get back. If you are correct about what the group was then you'll get back a message 2, if you are incorrect you get back the information you need to create a proper message 1.

Thwarting Timing Attacks Using ATM Networks

Geraint Price

University of Cambridge – Computer Laboratory,
England
gp200@cl.cam.ac.uk

Abstract. In this paper, we describe a method of using Asynchronous Transfer Mode (ATM) network technology to defeat attacks that rely on the opponent's ability disrupt the timely delivery of messages within a cryptographic protocol. Our method centres on ATM technology's ability to provide guarantees associated with the bandwidth and delay characteristics over a given connection. We manipulate these mechanisms to provide timing guarantees on a cryptographic protocol message, which can be used to monitor for foul play in the message delivery process. We also describe how this can be used to detect a denial of service attack.

1 Introduction

One of the many types of replication checks mentioned in the fault tolerant literature are *timing checks* [1, pages 123–124]. They are quite flexible in that they can be used in both software and hardware environments. In this paper, we describe a method of using Asynchronous Transfer Mode (ATM) networks in order to defeat or detect various timing and replay attacks on cryptographic protocols, by using such timing checks.

Attacks on cryptographic protocols that disrupt the timing of message delivery within a given protocol run are well known in the literature. Work by various authors [6,7,16,9,2,12,15] show the potential for such attacks on certain protocols. Each of these attacks has its different form, requiring a differing amount of knowledge about the messages and their content in order to be successful. What these techniques have in common is that they disrupt the flow of the messages within the protocol run to some degree. Gong [6] makes it clear that the main reason this is possible is down to the inherently unreliable networks that are used in the majority of distributed environments. This allows an attacker to interfere with packet transmission with impunity.

O'Connell and Patel [15] give a model whereby the trip latency on messages between two hosts is measured, this is then used to try and detect message delay. Their work – although not requiring synchronised clocks – relies on the measuring of real time, and reasonable measurement of clock drift between the two hosts.

By removing this unpredictability from the underlying communications infrastructure, we are able to reduce an attacker's ability to launch such attacks, also allowing a legitimate client to detect and respond to a potential attack.

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 49–58, 2002.

We use ATM as an example network architecture to counter this problem because of the bandwidth and delay guarantees that can be obtained on a connection.

2 ATM Functionality

It is a feature of ATM [11] networks that they allow a connection (called a Virtual Circuit or V.C.) to have certain upper and lower bound characteristics with regard to bandwidth and delay.

If an application is communicating via an ATM network, it can request guaranteed bandwidth and delay on a V.C. that it wishes to set up, which can be accepted or rejected by the network depending on current resource allocation.

2.1 Bandwidth Division

By giving an application that requires the use of a secure protocol the ability to request a V.C. with associated guarantees on bandwidth and delay, we can accurately predict the delay experienced by any message sent. Any deviation from this delay indicates the possibility of an attack. This does not always indicate an active attack – as no guarantee is absolute – but we can make the probability arbitrarily high.

We abstract this as dedicating a portion of the bandwidth for cryptographic protocols. This division should not disrupt the throughput for the majority of traffic, as once the cryptographic protocol is complete, the guaranteed bandwidth held by the principal is dropped.

By taking advantage of a channel which has its bandwidth and delay characteristics regulated, we are able to stop many timing attacks. Making use of this *secure* channel does not address any other issues in terms of security; it does not prevent eavesdropping, and all checks for integrity and authenticity are still be carried out by the cryptographic functions used by the application.

2.2 Cell Delay Variation and Discard Control

Information is physically sent using *cells*, the information distribution packet used on a V.C. Because of the queueing involved along the route that a cell travels from source to destination, the cells suffer from what is termed "Cell Delay Variation" (CDV, also called *jitter*) [5]. On a "Constant Bit Rate" (CBR) connection – which we use for the *secure* connection – the cells are sent out at regular intervals. Some cells will experience a shorter delay than preceding ones (causing *clumping*), and others will experience larger delay (causing *dispersion*). It is the statistical representation of these variations from cell to cell that defines the jitter. As the jitter on a given channel is likely to change gradually through time, any radical change in CDV around a *secure* connection will indicate the likelihood of an attack to the receiving end.

An advantage of measuring CDV over straight message delay is that cells should arrive at a roughly even distribution, subsequently any massive clumping (e.g., the whole message in successive cells) could signal an attack. This forces an attacker to carry out an attack without knowing the message content, because of the need to not disrupt the delay characteristics.

3 Changing the Protocol Stack

A simple representation of the protocol stack used in ATM is shown in figure 1 (from [11]). The proposed inclusion of a secure layer implementing the desired functionality has been added to the diagram. This could either be a new layer or be incorporated into an existing layer. For the purpose of our discussion, we will assume it resides above the AAL-x layer.

We describe an *initiator* of a secure V.C. to be the principal that requests the creation of a V.C. in order to transmit a cryptographic message[1]. A *receiver* is the recipient of the message.

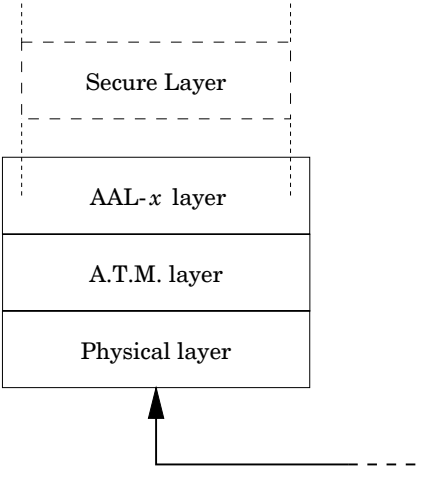


Fig. 1. Current (and proposed) ATM layers in stack

We restrict the number of *secure* V.C.s that are allowed to be open at any one time at a given host. This gives us an upper bound on the round trip delay for a message and its reply, because processing time at the receiving end will be – amongst other things – a function of the number of connections open.

The required functionality this layer should exhibit is characterised below:

[1] We use the term cryptographic message to describe a single message sent from one principal to another, that is part of a larger protocol (e.g., $[A \rightarrow S : A, B, N_a]$ constitutes a single message).

1. Initiator Side
   - Negotiate a secure V.C. from *initiator* to *receiver* with the underlying AAL-x layer.
   - Notify the initiator of the connection time on a V.C. carrying a message that has no response (such as the final message in the protocol), and whether this delay is acceptable.
   - Notify the client of the round trip delay of a message that is followed by a response (this includes both the time spent in both in-bound and out-bound connections and the processing time for the server). A breakdown of V.C. connection time and processing time should also be available.
   - If a request for a secure connection is refused, then back off and re-issue the request. This should be carried out a pre-set number of times before giving up and signalling an exception to the calling application.
2. Receiver Side
   - Notify the *receiver* process that an incoming V.C. is a secure request.
   - Notify the *receiver* if the delay on the incoming message falls outside the negotiated window.
   - Negotiate the response V.C. on a message in a protocol that requires a response (e.g., the second message in a handshake).
   - Notify the underlying AAL-x layer if the receiver already has the maximum number of V.C.s open, in order that a request for a new *secure* V.C. should be denied.

When negotiating for a *secure* connection, all parameters should be within a constant window (i.e., for a given network, all *secure* connections will have bandwidth and delay characteristics within a pre-defined envelope). This ensures that all participants in a cryptographic protocol are able to judge the outcome of their requests accurately.

3.1 Function Calls

Figure 2 shows an enumerated set of function calls to implement the scheme, with their descriptions given below:

1. Initiator calls
   1 : open_link. This is a call made by the application to the secure layer, it includes the address of the server to send the request to and includes the message content.
   2 : close_link. This is the successful request return made by an application. It includes a flag of whether the secure layer believes the request has been delayed or not.
   3 : link_fail. Informs the client that a connection for a V.C. was denied more times than a pre-set threshold.
   4 : link_req. The secure layer uses this to request a V.C. with the required bandwidth.
   5 : ref_req. A return signal to the secure layer indicating that the request has been refused.

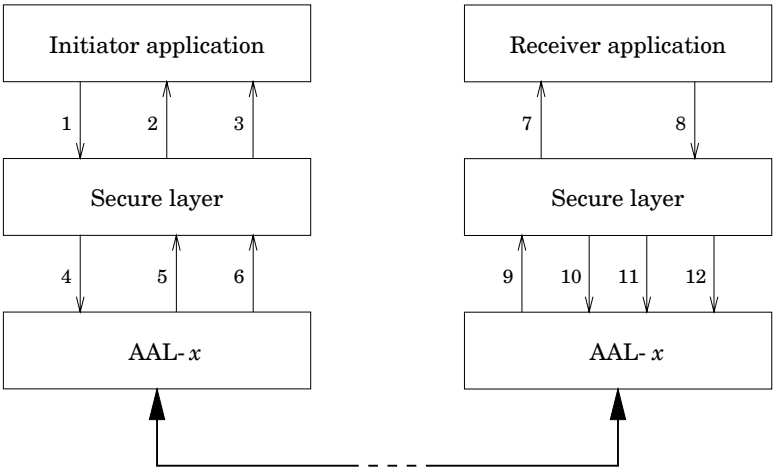


Fig. 2. Enumerated diagram for secure layer function calls

6 : return_msg This is a return message from the *receiver* to the *initiator* (if a response is part of the protocol). This is passed up to the application as a return message.

2. Receiver calls

7 : deliver_msg. Delivery of the message body from the *initiator* and flagged to indicate that it is a secure V.C. and if the delay is acceptable for the negotiated connection.

8 : msg_out. Outgoing message from the *receiver* to *initiator*.

9 : conn_req. Signal for a request of a secure V.C. from an *initiator*.

10 : conn_ref. Return signal refusing the request for a secure V.C. (due to a full incoming queue).

11 : conn_acc. Return signal accepting the request for a secure V.C.

12 : return_msg. Passing on the outgoing message from the server to the client.

When the secure layer receives a request from an application, it needs to negotiate a *secure* V.C. with the AAL-x layer. If the connection is accepted, then the secure layer starts a timer until the connection is closed by the *receiver* end of the V.C. It then checks if the actual connection time falls within the expected limits, flagging the result to the application.

Given that an attacker might try and give a false message to the *receiver* before the *initiator* sends the message, it should be up to the secure layer to wait for the whole duration that the V.C. is expected to be open before passing the message up to the application layer, given that the arrival of more than one message on a secure V.C. is itself likely to dignal an attack.

4 Examples of Usage

We now take a look at both replay attacks and denial of service attacks, evaluating how our scheme helps to prevent and detect these types of attacks.

4.1 Timing and Replay Attacks

First of all we take a look at some of the replay attacks described by Syverson [16]. In his paper he describes two separate taxonomies, an *Origination* taxonomy and a *Destination* taxonomy.

Taking the *interleaving attack* described by Syverson (the BAN-Yahalom protocol [3] and the attack on it are shown below), it is possible to see that A might raise the alarm in this situation. All principals involved in a cryptographic protocol can assume that all messages on a negotiated *secure* V.C. are sent and received. When A does not receive the message in round 3 of the BAN-Yahalom protocol, it is difficult for the attacker to cover the attack. If the attacker does not tear down the legitimate protocol, then B ends up receiving two versions of message 4 (one from A and one from ${E_a}^2$) and this will indicate the possibility of an attack.

The BAN-Yahalom Protocol
$(1) A \rightarrow B : A, N_a$
$(2) B \rightarrow S : B, N_b, \{A, N_a\}_{K_{bs}}$
$(3) S \rightarrow A : N_b, \{B, K_{ab}, N_a\}_{K_{as}}, \{A, K_{ab}, N_b\}_{K_{bs}}$
$(4) A \rightarrow B : \{A, K_{ab}, N_b\}_{K_{bs}}, \{N_b\}_{K_{ab}}$

Attack on the BAN-Yahalom Protocol
$(1) A \rightarrow B : A, N_a$
$(2) B \rightarrow S : B, N_b, \{A, N_a\}_{K_{bs}}$

$$(1') E_a \rightarrow B : A, (N_a, N_b)$$
$$(2') B \rightarrow E_s : B, N_b^{'}, \{A, N_a, N_b\}_{K_{bs}}$$

(3) Omitted.
$(4) E_a \rightarrow B : \{A, N_a (= K_{ab}), N_b\}_{K_{bs}}, \{N_b\}_{K_{ab}}$

Our system cannot defend against *classic replays* of the sort described by Denning and Sacco [4]. Their attack is mounted by a masquerader gaining access to an old session key, then initiating a fresh connection using a replay of part of the protocol in which that key was issued. We cannot defend against this type of attack, primarily because of the off-line nature of the attack [3]. Although our

[2] Where E_a is used to denote E masquerading as A.
[3] It is not really a goal of our system to defend against off-line attacks.

design could be used to augment systems such as those described by Lomas et al. [10] that do defend against such attacks.

In the case of *run internal attacks*, we have a similar defense to the interleaving attack described above. Due to the assurance that the principal has of the delivery of messages, it can regard a breakdown of a protocol run as suspicious.

Reflections and *deflections* can be detected if the attacker has to falsify a dropped protocol to interrupt the connection, similar to the case for interleaving.

In the case of *straight replays*, they are similar in concept to the *suppress-replay* attack discussed by Gong [6], where any message that is delayed is going to be picked up as a potential attack by our design.

An interesting point to note, is that if one connection shows signs of being tampered with, then it could be an indication of that connection being used as an oracle in attack on another connection in the system (as demonstrated from the BAN-Yahalom attack described above). The difficulty rasied by this question is trying to reconcile the two instances in order to determine the actual point of attack. Trying to design a robust method of achieving this goal would be an interesting point for further research.

4.2 Protocol Suitability

We take a brief look at how our scheme lends itself to different structures of protocol design.

$(1) A \rightarrow S : A, B, N_a$
$(2) S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
$(3) A \rightarrow B : \{K_{ab}, A\}_{K_{bs}}$
$(4) B \rightarrow A : \{N_b\}_{K_{ab}}$
$(5) A \rightarrow B : \{N_b - 1\}_{K_{ab}}$

By taking the enumerated description of Needham-Schroeder [14] given above, we can see that each client is able to encapsulate messages as pairs to and from another party in this protocol (e.g., A sees messages 1 & 2 as a pair to S). This means that they can make an accurate estimate of the round trip delay of this small part of the protocol that they view. In a protocol where clients are not involved in consecutive rounds of the protocol, it is more difficult for them to accurately predict the total delay expected until they receive their next message [4].

If we contrast Needham-Schroeder and the BAN-Yahalom protocol used in the *interleaving* attack in the previous section – where there are no symetric message pairs – we rely to a greater extent on the reliability of the communication

[4] It could be a policy that all connections through intermediaries are delayed for a pre-defined time (which would be a function of maximum message delays on the network), which would have the effect of re-synchronising the protocol at each step, although this does leave itself open to hijacking if the intermediaries are not fully trusted.

in the case of the protocol studied in the *interleaving attack*. We observe that our method is more adept at monitoring a protocol where there is symmetry in the message structure of the protocol (i.e., a protocol where a principal sends a message and receives a response directly from the principal that the first message is sent to). If, as part of a protocol, a client needs to send a message to another client, and does not receive the reply for a number of rounds within the protocol, then some inaccuracy in the calculation of the total round trip delay might be experienced, allowing more room for leverage from the attacker's point of view.

4.3 Denial of Service Attacks

It is possible to use the methods that we describe in order to detect a denial of service attack mounted against a cryptographic protocol. Indeed, in the example given by Needham [13] of the vault communicating with an alarm company, it is noted that the communication path between the two entities should have a guaranteed bandwidth.

As our design is more general, it is not possible to guarantee bandwidth to all clients at all times, which leaves us with the need to montior the use of *secure* channels. If the server is refusing another connection (e.g., if its input queue was already full) then as noted previously, the secure layer in the protocol stack would back-off for a short period, then re-issue the request. If the client end has to back-off more than a certain threshold, it would be reasonable to assume that the client or server was coming under a denial of service attack.

5 Conclusions

In this paper we present a complementary method of countering timing and replay attacks, by using the timing of the communication channel between the principals involved in the protocol. We view our design as augmenting current mechanisms for defending against timing attacks.

Our work has two distinct advantages when compared to that of O'Connell and Patel [15]. Firstly, continued delivery can be measured by using CDV to monitor cell by cell arrival, which has a distinct advantage over single message timing. Also, messages are guaranteed to be able to get through within a short time, independent of bandwidth usage by other applications for *non-secure* V.C.s.

We analysed how our design would detect the attacks mentioned in a taxonomy by Syverson [16], and show it handles *on-line* attacks (such as an *interleaving attack*) effectively. We also discuss our design's ability to deal with denial of service attacks. We conclude that it would handle this very efficiently. Any denial of service attack mounted against a specific client or server would be difficult to hide, given that our mechanisms monitor requests for service at either end of a V.C. and are not primarily concerned with bandwidth consumption across an entire network.

Policing of this mechanism for any abuse by applications would have to be a concern. The security module within each host on the network could be used to monitor for suspicious behaviour from any given application, raising an alarm with the network manager if need be, although pursiuing this more fully would provide interesting research.

Further work could be oriented towards the precise characterization of the properties required by our design. These definitions could then be used in the implementation of future networking standards, rather than shoe-horning our requirements into exisiting mechanisms. It is a feature of most implementations that they are designed with inadequate security features in mind, and a set of clear design goals for channel security could be one method of changing this.

To highlight the potential usefullness of our mechanism, we outline two scenarios where we believe our mechanism could be used effectively.

The first scenario is where maintaining the physical security of the LAN could be prohibitively expensive. Our mechanism could provide for a cheap and cheerful means of monitoring for potenial foul play on the network.

The second scenario is where several LANs are connected via an ATM backbone to generate an intranet. The propsal for this type of network topology is increasing [8]. The physical security of the two LANs might be easily guaranteed, but the backbone is a shared media with other untrusted entities. This type of topology might be common in a commerical environment where the company has many sites and wishes to provide a company wide network, without incurring the cost of a the dedicated hardware to provide this. Commerical espionage in this environment might provide a lucrative target.

References

1. T. Anderson and P.A. Lee. *Fault Tolerance: Principles and Practice*. Prentice-Hall International, 1981.
2. Tuomas Aura. Strategies against replay attacks. In *Proceedings of the* 10^{th} *Computer Security Foundations Workshop*, pages 59–68, June 1997.
3. M. Burrows, M. Abadi, and R.M. Needham. A logic of authentication. *Proceedings of the Royal Society of London*, 246:233–271, 1989.
4. Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Communications of the A.C.M.*, 24(8):533–536, August 1981.
5. Annie Gravey and Soren Blaabjerg, editors. *Cell Delay Variation in ATM Networks*. COST 242 Mid-term Seminar Interim Report, August 1994. Sections 1 & 2.
6. Li Gong. A security risk of depending on synchronized clocks. *A.C.M. Operating Systems Review*, 26(1):49–53, January 1992.
7. Li Gong. Variations on the themes of message freshness and replay. In *Proceedings of the Computer Security Foundations Workshop VI*, pages 131–136, June 1993.
8. Tim Hale. Integrating ATM backbones with high-speed Ethernet at the edge. `<http://www.3com.com/technology/tech_net/white_papers/500668.html>`. 3com White Paper.
9. Shyh-Wei Luan and Virgil D. Gligor. On replay detection in distributed systems. In *Proceedings of the* 10^{th} *International Conference on Distributed Computing*, pages 188–195, May 1990.

10. T. Mark A. Lomas, Li Gong, Jerome H. Saltzer, and Roger M. Needham. Reducing risks from poorly chosen keys. In *Proceedings of the* 12^{th} *A.C.M. Symposium on Operating Systems Principles*, December 1989. Published as *A.C.M. Operating Systems Review*, 23(5):14–18.
11. Ian M. Leslie and Derek R. McAuley. ATM: Theory and practice. University of Cambridge Computer Laboratory lecture notes, 1995.
12. C. Mitchell. Limitations of challenge-response entity authentication. *Electronics Letters*, 25(17):1195–1196, August 1989.
13. Roger M. Needham. Denial of service: An example. *Communications of the A.C.M.*, 37(11):42–46, November 1994.
14. Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the A.C.M.*, 21(12):993–999, December 1978.
15. S. O'Connell and A. Patel. A model for the detection of the message stream delay attack. In Sokratis K. Katsikas and Dimitris Gritzalis, editors, *Proceedings of the I.F.I.P* 12^{th} *International Conference on Information Security*, pages 438–451, May 1996.
16. Paul Syverson. A taxonomy of replay attacks. In *Proceedings of Computer Security Foundations Workshop VII*, pages 187–191, 1994.

Thwarting Timing Attacks Using ATM Networks (Transcript of Discussion)

Geraint Price

Center for Communications Systems Research

A quick overview of what I'm going to go through: the background of timing attacks; current means that have been used to overcome the attacks; why we're going to use ATM networks to overcome them; what changes we think ought to be put into the ATM structure; how this would provide some means of detecting and hopefully overcoming denial of service attacks as well; and then some conclusions at the end.

Timing attacks go back to literature in the early 90's looking at ways that protocols could be manipulated while they were running in order to find out session keys or convince the responder in an authentication protocol that you are Alice when you're actually Eve, given the capability to disrupt the protocol run in some way. Most of these come down to problems with parts of the authentication protocol being able to be disrupted with impunity in networks.

In Gong's paper looking at the problem that comes up in the Kerberos protocol, he says that authentication tends to break down especially on networks such as Ethernet that have an unreliable nature, so if a packet gets dropped or a message goes missing, nobody's going to worry about it too much. You're going to run another session of the protocol and not worry too much about what happened.

Bruce Christianson: The thing about Ethernet is that the collision recovery time is non-deterministic.

Reply: Yes, that's probably a more accurate description. Rather than just being unreliable, the delay is not bounded.

How have people gone about overcoming these attacks in the past? A lot of the research that went into formal proofs of protocols in the 90's was trying to identify what you could change within the protocol messages themselves in order to stop this from happening, so even if you tore down a legitimate protocol what you received or were able to overhear or were able to repeat in the future wouldn't allow you to circumvent the security of the protocol itself. But formally proving protocols has proved to be quite a difficult task. So we make sure that people can't put timing attacks into place.

O'Connell and Patel in 1996 proposed a means of using timing checks on message transmission. They way they went about it was to time from when the message was initially sent to when the message was finally received. The point of this was to reduce the unpredictability which networks have, and give you some guarantee that the message had been delivered straight through from the sender to the receiver.

So why do we pick ATM? It's designed so that all the traffic doesn't have to suffer under high load. Virtual circuits on the network can have upper and

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 59–62, 2002.

lower bound characteristics on the delay and bandwidth. That provides us with some form of guarantee that the message is going to be received at the receiver within a given time, and that the message itself is going to actually arrive at the other end with high probability. So you're not going to start losing messages, and if you start losing messages, you're going to wonder why. This doesn't prevent many of the security concerns that authentication protocols address, such as eavesdropping. We're just looking at providing a means of overcoming some attacks where an attacker needs to be able to disrupt the message transfer.

The type of virtual circuits we used are called Constant Bit Rate connections. Cells are sent out at definite intervals to use up the bandwidth that you've asked for. Our improvement on the paper by O'Connell and Patel is that the receiver can look at the dispersion or clumping that happens on the cells, called Cell Delay Variation. From the Cell Delay Variation you can tell if its likely that the message is being tampered with as it's travelling through the network. The switches should be passing cells on at consistent rates through the network, so if anybody's trying to replace parts of the message, or the whole message, then they're going to have difficulty doing so without affecting the characteristics of a given connection.

We're splitting the bandwidth of the network into ordinary bandwidth and then something that's reserved for a secure connection on behalf of an application during an authentication protocol. You'd want to have an upper bound on the number of secure virtual circuits that are available at a particular host at a given time, so that it can respond to each of them in time.

There's a paper published by Syverson that looks at a taxonomy of different types of replay attacks and timing attacks. He branches them into different categories. I won't go through them all. There are interleaving attacks (he gives an example of the Yahalom protocol) where an eavesdropper can replay a message and claim to be in the protocol itself. This attack relies on the attacker being able to tear down the communications protocol and replay the initial message. It's these sort of attacks that we're looking to try and defeat.

There are what he calls the classic replay attacks (these are the type that Denning and Sacco talk about in their paper) where you can recover old session keys and use them to initiate new communications with legitimate participants of old communications and get them to believe that it's a fresh communication. We can't defend against this because it's an off-line attack. We're trying to defend against on-line attacks where disruption of the communication needs to be carried out.

As a side issue of this, we found that it provides some sort of protection against denial of service attacks. In Needham's 1994 paper where he looks at problems with people trying to disrupt the communication from a bank vault to a security company, he mentions that providing some sort of guaranteed bandwidth would make the job of an attacker harder. So providing some form of guaranteed communications using ATM gives us one means of overcoming denial of service attacks.

Conclusions. We don't see this as a means of letting everybody get away with having sloppy authentication protocols, it's just a means of providing some engineering advantage in problems that are difficult to solve by cryptographic means. The reason we chose ATM is because you can measure the jitter on the network, and that to us is a better means of monitoring the communication than end to end message delay. It also provides some denial of service protection. Policing these mechanisms might be an issue. If secure virtual circuits are the only type of circuits on the network that are allowed to have strict guarantees, applications might be tempted to abuse them as a means of trying to do other things. I don't know how you would go about trying to police them so you don't waste bandwidth that should be used for secure virtual circuits on more frivolous things.

I'd like to pose a question. I've used ATM guarantees on delay and bandwidth here as a means of trying to augment security facilities. What other physical properties do networks have that we can use to augment security?

John Ioannidis: Unless you're constraining the computers on either side to be running hard real time operating systems with bounded response times guaranteed for the computations they're going to perform, how can you tell that something's failing in the network versus something failing in the computer? How do you know that the high variance in response time you're getting is not due to the computer? If the attacker can attack an ATM switch, presume he can attack sufficiently so that he can reroute the traffic on a particular connection and still pretend that it's come from the right virtual circuit. It adds delay, but you don't know whether its delay in the ATM network or delay in the the server.

Reply: As you say, you'd have to have some sort of hard guarantee on the host itself.

Markus Kuhn: Aren't you just moving the interest of an attacker away from the actual switched channel to the connection establishment of the virtual circuit? If I have control of a rogue router, during the negotiation of the virtual circuit I just say that you can have this connection but there will be a three second delay. In these three seconds I can either pass you through a FIFO or I can pass you through my attack machine.

Reply: But when the network is set up, all the delays are bounded within system parameters that are set. So three seconds may be way beyond what the the end hosts are told is a reasonable negotiation.

Markus Kuhn: But that reduces your robustness, because there might be a sea cable failure and for a day we have to reroute everything via two geostationary satellites. Then you have three seconds extra delay. You exclude a lot of backup routes if you have a fixed bound.

John Ioannidis: You exclude all real networks if you rely on ATM.

Reply: I'm assuming here that the system administrator would be aware of it. You'd get a message from your system administrator saying because of problems with the transatlantic link expect a certain amount of delay, and that can then be configured in.

Markus Kuhn: There could be an intelligence agency co-operating with a telecomms provider — it's happened before — shutting down your access to an undersea line for the specific purpose of getting the additional 200 milliseconds they calculated they need to do the attack.

Bruce Christianson: A more likely scenario is that the intelligence agency is routing a satellite line through a land line at great expense to themselves to get the extra processing time that they need to carry out the attack.

Reply: All right, so you get extra time, but some of these attacks actually rely on the communication failing completely.

Towards a Survivable Security Architecture for Ad-Hoc Networks

Tuomas Aura[1] and Silja Mäki[2]

[1] Microsoft Research Ltd.
7 J J Thomson Avenue, Cambridge, CB3 0FB, UK
tuomaura@microsoft.com
[2] Helsinki University of Technology, Laboratory for Theoretical Computer Science
P.O.Box 5400, FIN-02015 HUT, Finland
Silja.Maki@hut.fi

Abstract. We present a security architecture for access control in ad-hoc networks of mobile electronic devices. Ad-hoc networks are formed on demand without support from pre-existing infrastructure such as central servers, security associations or CAs. Our architecture is fully distributed and based on groups and public-key certification. The goal is a survivable system that functions well even when network nodes fail and connections are only occasional. We identify some open problems in the optimal use of unreliable communications for security management.

1 Introduction

Consider the digital appliances, such as desktop PC, mobile phone, MP3 player, palmtop computer, and car computer, that belong to one person. While each one of these devices is useful alone, their full potential will only be realized when they are connected into a network to share information and services with each other. The equipment must also be able to connect to outside services such as a printer or an email server. Much work is currently done on this kind of ad-hoc networking of smart devices with wireless connections [5,11].

Ad-hoc networks differ considerably from the fixed communications networks. The nodes and connections are inherently unreliable. The networks are created on demand, without support from fixed infrastructure such as central servers and routers. The networks, in particular, security in these environments must be managed without the help of pre-existing trusted servers, naming schemes, certification authorities (CAs), or security associations.

The organization of ad-hoc networks is often based on *groups* of nodes [7,15]. In this paper, we describe an architecture for securely creating groups, managing their membership, and proving group membership. The design is fully

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 63–73, 2002.

distributed, based on key-oriented key certificates, and *survivable in situations where nodes fail and communication is only occasional.*

The primary use of our architecture is in access control but it can also be used for authentication in various group key exchange protocols, e.g. [3,2,1]. In addition to the personal networks of electronics, the protocol may find applications in other systems with similar characteristic, including office and home appliances and mobile rescue and military operations.

Much of the complexity in our design comes from the fact that on-line verification of access rights and predictable distribution of revocation lists cannot be implemented with the completely unreliable communication. We present a best-effort solution that seems like a reasonable compromise and identify open theoretical problems in optimization of the revocation procedure.

2 Distributed Group Membership Management

With a group, we mean a set of members, persons or other physical or logical entities that may collectively be given access rights. Some of the members are leaders and have the right to make decisions about group membership. Groups may also have subgroups.

The membership management is based on public-key certificates. Our view is key-oriented. As in the SPKI [10,9], SDSI [14] and KeyNote [4] infrastructures, the group is represented by a public key, the group key, and its members are identified by their respective public keys.

2.1 The Purpose of Groups

The primary purpose of a group is that it may be granted access rights, which any member of the group is entitled to use. For example, the group members could be allowed to use a printer or to access a database. The group membership, as we define it, never imposes any obligations onto the members. It follows that the members of a group (e.g. printer users) do not need to trust each other in any respect. In fact, they might not even know about each other.

However, a common use for a group is co-operation and mutual authentication inside the group. For example, the electronic appliances owned by a single person may be configured to have full access to each other. The group members may create a shared encryption key for protecting communication between themselves. Every time a member communicates with another one and transmits or receives a message encrypted or authenticated with the shared key, it makes the implicit decision to trust the group with its secrets or its integrity. The group members

may also decide to provide each other services like message routing and network management. In every case, the members must have a reason, such as an administrative decision, for trusting the specific group with respect to the purpose for which they use it. It is a prudent practice to write the original purpose of the group explicitly on all the member certificates.

2.2 The Basic Group Structure

A new group is created by generating a fresh signature key pair. The new key KG is called the *group key*. The public signature key is used as the group identifier. The group key is also used to certify new members and to verify membership. The owner of the group key is called the *leader*. Anyone capable of generating a fresh signature key may start a new group and become its leader. This way, groups can be created ad hoc.

In the beginning, the leader is the only member of the group. The leader may admit other members by issuing *member certificates* to their public keys (*member keys*). The certificates are signed with the group key KG. A member certificate contains the group identifier (i.e. the public group key), the public key of the member, an optional validity period and a signature signed with the group key.

To protect against the loss or disconnection of the leader, the leader may distribute its powers by appointing other leaders. It does this by issuing *leader certificates* to their public keys (*leader keys*). The leader certificates have the same structure as the member certificates. All leaders of the group are also members of it.

All the leaders have an equivalent authority with the original leader and each one may act independently. They all may admit new members and new leaders to the group by signing certificates with their own keys. All the certificates will have the same group identifier (the public group key or its secure hash value) on them.

The multiple independent leaders make the group survivable. If some leaders are removed from the system either temporarily or permanently, the remaining leaders can still continue operation. A disadvantage is that no single leader is guaranteed to have a whole picture of the group membership at a given moment. The leaders should have a protocol for sharing information about the members with each other whenever communication is possible.

The protocol allows the leaders to act independently because any mechanical restrictions will make the system less reliable. In practice, however, the leaders should follow some agreed policy in making their decisions or in taking over leadership when connection to another leader is lost. For instance, in our example of personal appliances, the preferences of the user operating the equipment will in a natural way create such a policy.

We note that while *threshold schemes* [8,16] can theoretically improve the security of mobile networks where the nodes are exposed to physical danger, they can easily lead to denial of service and create frustrated users. Hence, they are not suitable for distributed systems where the device owners from consumers to military leaders want to implement their decisions instantly without waiting for approval from several possibly unreachable peers. Therefore, our architecture allows every authorized leader to act independently.

When a leader certifies other leaders or members, it passes along all the certificates that prove its own status as a leader in the group. This way, a *certificate chain* is formed from the group key to each member key. A member can prove its membership to another group member or to an outsider by presenting its certificates and by using its own private key.

Since the group key is the group identifier, everyone doing business with the group will automatically know it. Services, like a printer, will have the group key in an access control list (ACL). The group key from the ACL is used to verify the first certificate in the chain. Each of the following certificates is verified with the leader key certified in the previous certificate. No names or certification authorities are needed in the verification process, and only a single key per group is needed in any ACL. Moreover, the verification of access rights can be implemented so that the verifier uses only a constant amount of memory.

Fig. 1 illustrates a group with multiple leaders. (For simplicity, the validity periods of the certificates are not shown in the figure.) When Member 5 of Fig. 1 joined the group, it received the membership certificate signed by Leader 2 and the leader certificate issued to Leader 2, which is signed by KG. The latter certificate proves the authority of Leader 2. When Member 5 wants to prove its membership, it presents both certificates and proves its possession of the key $K5$.

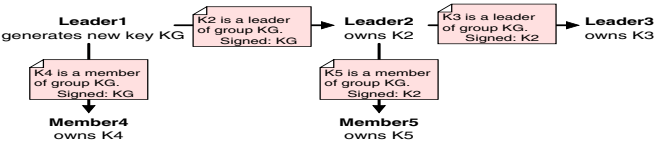


Fig. 1. A group with multiple leaders

2.3 Subgroups

Sometimes a leader may want to admit all members of another group to its own group. For this purpose, we have defined a *subgroup certificate*, which any leader can issue to the group key of the subgroup. The subgroup certificate has the

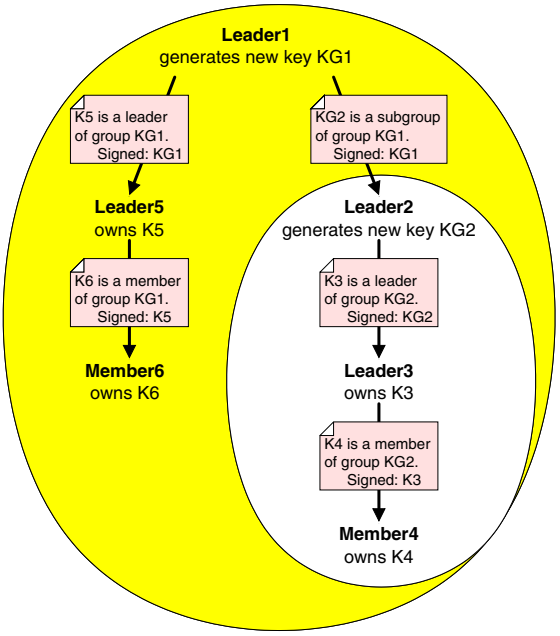


Fig. 2. Subgroups

same structure as a member or leader certificate. It contains the *supergroup* and subgroup identifiers (i.e. keys), a validity period, and a signature.

All the members of a subgroup are also members of the supergroup. The leaders of a subgroup may continue to admit new members to the subgroup. Any changes in the subgroup affect also the membership of the supergroup. Otherwise, the subgroup and supergroup are managed independently. The subgroup leaders are not leaders in the supergroup or vice versa, unless they are separately certified as such. The aim is to keep the membership management functions as simple and local as possible.

Figure 2 shows how the group *KG2* becomes a subgroup of the group *KG1*. The leaders of the groups *KG1* and *KG2* can continue to admit new members, but not directly to each other's groups.

The membership of a group is transitive in a hierarchy of subgroups but applications should keep the depth of this hierarchy to the minimum. In fact, we suggest avoiding the complications of subgroups altogether except when they simplify the system management significantly. A typical situation where subgroups are useful is the management of a small hierarchical organization. For example, the electronic appliances of a household include shared items and the personal items of each family member.

3 Revoking Membership

When a member voluntarily leaves a group, it should discard the member certificate and, if possible, erase its signature key. In many applications, the members can be trusted to do this. For example, when an electrical device is sold or given away, the owner can be expected to reset its memory.

Nevertheless, there are situations where the members may not be co-operative. If a member stops paying for a service, a device is stolen, or there is a sudden suspicion that a remote computer has been hacked, some preemptive means is needed to remove them from the group.

3.1 Simple Methods

The most effective way to get rid of unwanted members or leaders is *group reconstitution*, replacing the group key with a new one and reissuing all certificates. This guarantees that no unwanted members remain in the group. The problem is that the certificates must be distributed and the authentic new group key must be conveyed to all the places where membership is verified. This costs time and may be impossible when communication is unreliable. In practice, the new group has to replace the old one gradually so that any nodes unaware of the new group can still use the old keys and certificates.

Another secure way to get rid of unwanted certificates is to have *short validity times* and to require the members to renew their membership regularly. This means that all the members must communicate regularly with leaders to obtain new member certificates. It may, however, be impractical to let the leader certificates expire because the renewed certificates must be distributed to all the members certified by the recertified leaders.

3.2 Best-Effort Revocation

The methods described above work well only when time is not critical and communication is guaranteed. Therefore, we have to resort to the distribution of *revocation lists*, which is much faster, albeit less reliable.

In our protocol, a leader may revoke the membership of any members or leaders of the same group. Note that we revoke membership, i.e. $\langle Groupkey, Memberkey \rangle$ pairs, and not keys (as in X.509 or PGP [6,17]) or certificates (as in SPKI). (Leaders are not allowed to remove members from other groups, so universal revocation of a key cannot be allowed, and it is more convenient to revoke all the redundant certificate chains at once than to revoke each certificate separately.) The revocation entries have the same structure as the membership certificates.

Like the certificates, they must be signed by a leader of the group and they can only be accepted when accompanied by the certificate chain that proves the leadership.

Additionally, we allow a key to revoke its own membership in all groups at once. The revocation information may be collected to revocation lists if there is a lot of it. However, group reconstitution and agreed maximum validity times help to reduce the amount of revocation data.

3.3 Minimizing the Side Effects

A certificate chain is only valid as long as all certificates in it are. If a leader's membership is revoked, that makes invalid all certificates signed by that leader. Consequently, all members that depend on such certificate must obtain new ones. Our architecture provides two methods for protecting the members against such effects: redundant certificate chains and erasing the private group key.

The members may obtain multiple certificates from independent leaders. If one chain becomes invalid, another one can still be used to prove the membership. Verification of the certificates is also made faster by using the shortest valid chains.

Redundant certificates will, however, not help if the group key itself is revoked from the group. Therefore, it pays to protect the private group key particularly well. The most effective way to prevent the compromise of a private key is, simply, to erase it. After a new group key is generated, it can be used to sign a few leader certificates and then destroyed. The certificates signed with the erased key will remain valid and can still be verified with the public key. That way, it is never necessary to revoke the group key.

In Fig. 3, if either leader loses its authority due to revocation, no other member is affected. Note that the redundant certificates do not complicate the revocation in any way. In fact, the leader revoking a member need not know which certificates have been issued to it. This is why we have chosen to revoke membership and not single certificates.

4 Optimizing Revocation

Revocation with unreliable communication is inherently unreliable. Conveying revocation information to all the verifiers may take an arbitrarily long time and some nodes may never be reached. But even when the revocation cannot be guaranteed to succeed, care should be taken to minimize the window of opportunity for the attacker.

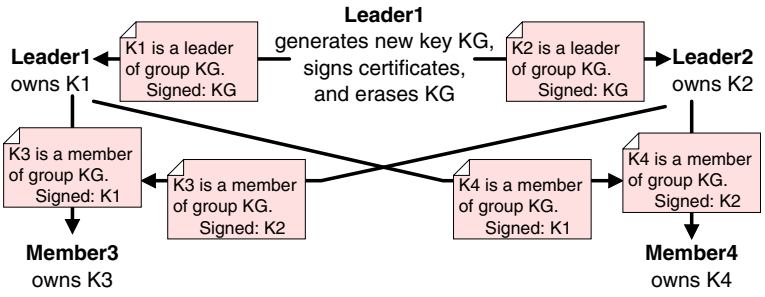


Fig. 3. Redundant certificates and erased group key

We will first describe how revocation information is handled in our architecture and then discuss potential improvements.

4.1 Propagation inside a Group

The revocation data needs to be conveyed to all the entities that may verify the membership. These include both outside services and group members.

The primary method of distributing revocation lists in our architecture is propagation from member to member. When two members of the same group come to contact, they should propagate revocation information to each other. This way, the data that one member has eventually spread to all those members that are connected by regular communication.

In order to avoid denial of service by flooding with bogus revocation entries, a node is only required to accept revocation data for the groups in which it knows itself to be a member. This data is easy to recognize because it mentions the group name, it is signed by a leader of the group, and it is accompanied by a certificate chain that proves the authority of that leader. Other data is stored and forwarded only when there is excess capacity.

Clearly, a member should be revoked only when there is a reason to suspect that the private key has been compromised and there is an urgent need to react. Revocation should not be used for scheduled key changes or for replacing accidentally deleted keys. Moreover, fixed maximum validity periods for certificates or regular reconstitution of the group starting from a new group key should be used to keep the revocation lists short.

4.2 Registration of outside Services

Since we limit the propagation of revocation information to the group members themselves, some other mechanism must be provided to notify outside services that also may verify group membership. For example, when a group has the right to access a printer, the printer may want to know about removed members.

For this purpose, we borrow a solution from classic revocation literature: the service must register its contact information with some group members, which will then inform the service. These members are often group leaders.

The outside services, like group members themselves, may have limited storage capacity. If a service is not willing to store revocation information and a member of the client group if revoked, the group should unsubscribe the service (have the group key removed from the ACL) until the group has been reconstituted by replacing the group key.

4.3 Subgroups and Revocation

When it comes to revocation, we have decided to treat supergroups in the same way as external services. If the supergroup leaders want to receive revocation data from a group, they must register with appropriate members. After receiving a revocation message, the leader may issue a revocation message of the same key also in the supergroup.

This choice is by no means obvious. It means that revocation of a subgroup member will not be passed directly to the supergroup members even if they come to contact with the subgroup before their leaders do. The reason for this is that the supergroup members do not necessarily know about the subgroups and cannot recognize the revocation data as relevant to them. In order to avoid denial-of-service attacks, they cannot be required to store and distribute the data. Non-leader members of the supergroup may also register as recipients of the subgroup revocation data if they know it to be useful.

Information that comes from the supergroup to the subgroup is less problematic. The subgroup members either know about the supergroup or they do not. Only those that are aware or their membership in the supergroup will propagate revocation data for that group. This is natural because the nodes that do not even know about the supergroup certainly do not need to know about revocations.

4.4 Optimal Revocation?

While the policies outlined above seem to create a reasonable balance between effectiveness and cost of the revocation for the applications that we have in mind, they are nor guaranteed to be optimal in any respect.

In a theoretical setting, we could aim for information theoretically maximal distribution of revocation lists. We could simply require all nodes to store all revocation data they have ever received and to pass it to all the nodes that they communicate with. In the synchronous model of communication that we have implicitly assumed above, any two or more nodes that establish a communications link should synchronize their revocation lists. In an asynchronous model, the data should be appended to all messages ever sent. Such requirements are clearly unreasonable for real-world systems with limited memory and bandwidth. The threat of denial-of-service attacks also means that the nodes cannot be required to accept any information unless they can verify that it is authentic.

It is an open question how to optimally propagate revocation information in an distributed system where communication between nodes is only occasional and the nodes have capacity limitations. To solve this problem, one should also provide some exact measure for comparing the efficiency of such systems. The theory of gossip-based probabilistic event propagation might provide some tools for the comparison [12]. Since these questions are hard to answer, we have taken a practical approach in this paper and suggested a first suboptimal solution, which hopefully can be improved.

5 Conclusions

We described techniques for secure group membership management in highly distributed systems where nodes are mobile and prone to loss or failure, and communication links between them exist only occasionally. In our architecture, a group can be created ad hoc by generating a new public signature key, which will be used as the group identifier and to certify the public keys of other group leaders and members. The leaders may act independently in certifying further leaders and members, and in revoking them. The optimal propagation of revocation lists with given resources and unreliable communication was identified as an open problem.

Acknowledgments. This research was funded by Defence Forces Research Institute of Technology, Finland, and supported by Helsinki Graduate School in Computer Science and Engineering (HeCSE) and Academy of Finland grant #47754. The work was done mostly while Tuomas Aura was at HUT. The group architecture was originally presented in Nordsec2000 [13].

References

1. N. Asokan and Philip Ginzboorg. Key-argeement in ad-hoc networks. *Elsevier Preprint*, 2000. To appear.

2. Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. New multiparty authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communications*, 18(4):628–640, April 2000.
3. Mike Burmester and Yvo Desmedt. A secure and efficient conference key distribution system. In *Advances in Cryptology – EUROCRYPT '94*, volume 950 of *LNCS*, pages 275–286, Perugia, Italy, May 1994. Springer.
4. Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos Keromytis. The KeyNote trust-management system version 2. RFC 2704, IETF Network Working Group, September 1999.
5. Specification of the Bluetooth system, version 1.0b, 1999.
6. CCITT. *Recommendation X.509, The Directory – Authentication Framework*, volume VIII of *CCITT Blue Book*, pages 48–81. 1988.
7. Wenli Chen, Nitin Jain, and Suresh Singh. ANMP: ad hoc network management protocol. *IEEE Journal on Selected Areas in Communication*, 17(8):1506–1531, August 1999.
8. Yvo G. Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, July–August 1994.
9. Carl Ellison, Bill Franz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylönen. Simple public key certificate. Internet draft, July 1999. Work in progress.
10. Carl Ellison, Bill Franz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylönen. SPKI certificate theory. RFC 2693, IETF Network Working Group, September 1999.
11. IEEE Standards Board. 802 part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, 1997.
12. Anne-Marie Kermarrec, Laurent Massoulie, and Ayalvadi J. Ganesh. Reliable probabilistic communication in large-scale information dissemination systems. Technical Report MMSR-TR-2000-105, Microsoft Research, Cambridge, UK, October 2000.
13. Silja Mäki, Tuomas Aura, and Maarit Hietalahti. Robust membership management for ad-hoc groups. In *Proc. 5th Nordic Workshop on Secure IT Systems (NORDSEC 2000)*, Reykjavik, Iceland, October 2000.
14. Ronald L. Rivest and Butler Lampson. SDSI – a simple distributed security infrastucture. Technical report, April 1996.
15. Gert Roelofsen. TETRA security – the fundament of a high performance system. In *Proc. TETRA Conference 1997*, 1997.
16. Lidong Zhou and Zygmunt J. Haas. Securing ad hoc networks. *IEEE Network Magazine*, 13(6), November-December 1999.
17. Philip Zimmermann. *The Official PGP User's Guide*. MIT Press, June 1995.

Towards a Survivable Security Architecture for Ad-Hoc Networks (Transcript of Discussion)

Silja Mäki

Helsinki University of Technology

I'm going to talk about secure group management in ad-hoc networks. This is joint work with Tuomas Aura and it is partly funded by Finnish defence forces. Just when things started to look better, for example, in access control, we are now running into troubles again. The point is that in traditional security models, access control usually relied on on-line servers which needed fast and reliable global communication. That was becoming true in the Internet and other networks but at the same time we are developing new networks that again have limited capacity, poor coverage, unreliable links and nodes.

An example of such networks could be a network of personal electronics of one person. There could be the devices at the office or at home, the car, the laptop computer of one person, his or her palmtop PC, mobile phone, and so on, and all these devices are getting smarter all the time, but more importantly they are also going to communicate as home networks with each other. Now the devices have consumer quality and probably operate over radio links although there may be also some fixed connections at times, and mobile nodes would probably have batteries that they work on. All these things make the communication unreliable again, and also the nodes have limited capacity and may run out of energy.

The previous example was a typical ad-hoc network. Ad-hoc networks generally are created on demand. They don't rely on any permanent infrastructure such as servers or routers, they neither have CAs, trusted parties or keys, this is because they usually also they also lack administrational structure. The communication with the nodes is unreliable and actually the communication may be only occasional. Because ad-hoc networks typically don't have any hierarchy between the nodes a more natural organisation is groups of nodes. In these groups all the nodes are equal and they together, or separately, take part in the network management security issues and routing. The nodes of a group could, for example, create a group key to share secret information between each other. These groups can also be used as a basis for access control in the network. When a group is used for access control then the access rights are not granted for individual nodes but rather they are granted to groups of nodes and the nodes can use these rights by demonstrating their membership in the group.

So I will now have a few words about how to create these groups and manage them and then speak about the security issues and some problems that are related with these groups. Our idea was based on the use of public keys and public key certificates. All the members of the group have public key certificates that demonstrate their membership, and the keys are used as the identity of the

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 74–79, 2002.

members. Also the group has an identity which is the public key of the group. Using this key-oriented approach, no names, no CAs are needed, only the keys are used and our proposal has this ad-hoc nature, so that anyone may start the group just by generating the group key.

To come back to this example of personal electronics, the office PC generates a new group key, a private and a public key pair, and all these devices generate their own public key pairs which present their identities. The devices are added to the group so that the office PC, which is the leader and the generator of this group, signs membership certificates to the devices.

The leadership is further distributed. This is because the nodes in an ad-hoc network generally have less physical security than in traditional networks and so they can be lost permanently. Also because the links are unreliable, they can be temporarily unreachable. So in order for the group to function, and also when the generator is not available, there need to be more leaders, and the leaders are introduced with leader certificates. The original leader of the group gives leadership to others that can go on acting independently and equally with the original one. So, for example, the owner of the mobile phone, laptop PC, palmtop computer could have another leader for the group at home, the home PC, and these two leaders can work independently, they can sign certificates to new members and also certify new leaders when needed.

Subgroup certificates admit a whole group as members to the other group. In the previous example this could be used, for example, when the other electronics of the home make one group, and then the electronics of each of the members of the family have their own subgroups for this bigger group. When a subgroup certificate is issued, all the members of the subgroup become members of a bigger group, but the subgroup and the supergroup go about their management independently of each other.

So what new does this approach bring to security? Nowadays groups can be used as a basis for access control, for example a group of these personal appliances could have access to a printer, and now the printer stores the group key in its access control list and when a device from this group wants to use the printer it presents the certificate chain that binds it to the group key and a signed service request. In traditional models, the access control lists were usually used with identities of the users, and then there was this option of certificates which directly show the authority to use a service, this is a combination of those.

Another application of these groups is group key exchange, now the members of the group can authenticate each other with the help of the membership certificates and generate a secret group key for confidential communication. The distributed leadership makes a protocol more survivable, it's not dependent on any single leader, but even if some leaders are removed, the others can go on managing the group. The disadvantage of distributed leadership is that nobody may know about all the members.

We didn't choose any threshold schemes for the group management, this is because it is often needed to be able to make quick decisions and if the commu-

nication is not reliable it could be very painful to wait for agreement from other members of the group.

I will now have a few words about key compromises and the question, can the groups get secure? The keys are probably not compromised so often, but they are suspected more often. This happens again because nodes in an ad-hoc network may not have so good physical security. Three kinds of solutions could be used to keep the group secure, these are group reconstitution, certificate expiration, and best effort membership revocation.

Group reconstitution simply means that periodically or on demand somebody creates a new group key pair, signs new membership certificates and distributes them to the trusted members. This may take some time because the links are not so stable, and if it is done in a hurry it may happen that some trusted members are left out of the group.

Certificates have a validity period and when they expire they either can be re-signed or left without, so a member who is not trusted anymore may not have a new certificate. This also cannot be used when there is an immediate suspicion about a compromised key, it's only useful to keep up some kind of security awareness and to periodically renew the certificates of the trusted members.

Membership revocation is needed when there are some immediate reactions that want to be done, for example, if the mobile phone is stolen then it's reasonable to revoke it from the group so that it cannot be used for these access rights that the group has. Membership revocation cannot be done completely reliably because the links are not stable, and therefore we have to just settle for a best efforts revocation method. And this is the open problem, how to do this most efficiently.

To revoke a membership the leader signs a group key member key pair and these revocation messages are propagated throughout the network. The leader of the group may revoke anyone in its group and a key may also revoke itself. Revocation is only needed when there is a suspected compromise of keys, if someone only wants to leave the group then it's enough if he erases his keys.

So how to do a best effort revocation? If it were possible that all the nodes collected all the revocation data that they have, and every time they need each other, tell everything they know to each other, then probably all the nodes would sooner or later know about all the revoked members. However, it's not possible that they store and distribute all the revocation data that goes on in the network, because their capacity and the memory is limited and this would easily lead to denial of service. So, if we assume that the nodes store some amount of revocation data and distribute it to some other nodes in the group then we have a gossip-based probabilistic distribution inside the group. We assume that nodes in the same group always tell each other about changes or revocation inside the group, the outside services would know about revocations when they have registered with the group to receive revocation messages that concern them.

The sub- and supergroups are treated as outsiders, so for example, when the group key is added to the printer's access control list the printer may also ask to know about the revocations that are done in the group. If any server doesn't

want to know about the revocation done in the group, then it's the group's responsibility alone to take care of the security, and basically this means to refine the service and constitute a new group. Some further applications for the groups could be found in mobile military networks, in rescue and law enforcement operations and in office equipment.

To summarise, we presented a group membership protocol for ad-hoc groups that works with unreliable communications and unreliable nodes. It's survivable because it's fully distributed and it's only based on key oriented public key certificates and doesn't need any centralised services to function. Best effort propagation of revocation information is an open research topic.

Markus Kuhn: Have you any idea how convenient this is for end users who are not computer scientists to use, whether a normal buyer of a laser printer will actually understand this?

Reply: In my opinion it's pretty intuitive. I skipped the technical slides on generating the keys and distributing the certificates, but I'll show them now. Certificates here are only signed messages, and now, for example, if this member 5 wants to prove that it's a member of the group, it somehow proves that it owns the private key and then shows the two certificates that bind it to the original identity of the group, the key K_G. Well I don't know how intuitive users find this.

Tuomas Aura: Well, given any particular implementation, the user doesn't really need to know there are certificates at all, the user will just know that they have a group of devices and that some of the devices are linked. Is there something un-intuitive about the key?

Markus Kuhn: Why does the user have to be aware of the concept of who the leader is?

Tuomas Aura: Because we don't have to make everyone a leader, so the user needs to know that with some devices he can control this membership and manage that so as to add a new member or to revoke one.

Babak Sadighi: But who decides who is the leader? If you think of military applications then you should have some kind of protocol.

Tuomas Aura: In fact it comes to what Roger was talking about earlier[1], if you have only one leader — like in the military — and only this leader is allowed to make the decisions, then what happens when this leader fails — like gets shot? The next in the line will take over the position and continue right from there without thinking for a minute.

Bruce Christianson: Yes, leadership is a rôle, and there is some global state that says who has this rôle at a particular time.

Tuomas Aura: But if you enforce this kind of thing in the computer, if you build technical access controls that prevent the other one — whoever happens to be there — from taking over, then it doesn't work well. It seems to us that actually, if you think of this military thing, you should just give the leadership to all the senior officers in this area, or in the case of systems, to all the technicians in this end-network, or something like that. Then they will have this verbal

[1] Keynote Address, these proceedings.

agreement of who is doing the work at the moment, and then they can use their own judgement when to take over. So just because you have the rights to be a leader, it doesn't mean you actually use them, that's not necessary.

Pekka Nikander: What happens when one of the leaders gets captured by the enemy?

Tuomas Aura: That's why you need the basis for revocation. If I see that a man was caught, so he might still be alive, and his equipment might still be running — because he didn't have time to press Ctrl-Alt-Del or something like that — then that's when I want to tell everyone whom I need to, that they should suspect this person.

Bruce Christianson: But you say this quite clearly in the paper: that you are distinguishing between power and permission[2], and you are giving a mechanism to referee who has the *power* to do what. That's not to say that they mightn't have a protocol amongst themselves that says who will actually do it, and who is the backup and who is the primary. But your mechanism simply addresses the issue of power. It's a separate problem to prevent the backup from acting as the primary when they don't have *permission* to do so.

Reply: That's true.

Tuomas Aura: And we definitely don't want to make the access control system too restrictive, even for the case of consumer goods. If you think that you only have one leader in your system, and your PC breaks down, then suddenly you can't do anything with your other devices.

Bruce Christianson: This form of leadership is more analogous to apostolic succession perhaps than to military hierarchy [laughter].

Roger Needham: That's an arcane remark even coming from you, Bruce.

Mark Lomas: Could one approach to the revocation problem be to introduce the idea of a quorum within the group, so you say unless the majority of members of the group believe a key to be valid, then by definition it isn't. It's invalid.

Bruce Christianson: What is the difference between a quorum and a threshold? We've already rejected thresholds.

Tuomas Aura: I mean, isn't there anyone who really likes thresholds? [laughter]. They're theoretically very nice but in practice we found, especially speaking with the military people (Silja used to work for the military) that they wouldn't hear about something that prevents them from making a decision. And at this point they would have to get an approval from someone else.

Bruce Christianson: More to the point, your limiting factor is the ability to propagate information about changed beliefs, and having a threshold scheme doesn't change that problem. You've still got the difficulty that I have changed my belief about whether something is valid, but I have no sure way of telling the others about this change.

Mark Lomas: I think we may be talking about different things. I'm not suggesting that you have to communicate, I'm saying that your lack of communication can be taken as an indication that you might not believe the key.

[2] See also LNCS 1796, pp 48–59.

Bruce Christianson: But that doesn't help, because you then have a denial of service problem. You can trade revocation for mandatory refreshment, depending on which threat you would prefer to avoid and which you would be prepared to live with, but you can't avoid both.

Mark Lomas: It depends on your security policy which you prefer.

Bruce Christianson: Exactly. The military particularly don't like the idea that they *can't* push the button[3].

Tuomas Aura: All the so-called superpowers have a complex system of access controls inside the military in peacetime, but this certainly does not exist on the battlefield.

Bruce Christianson: Yes, just as Roger said earlier: as soon as war breaks out you turn the security off [laughter].

Tuomas Aura: For example in the heat of combat, you're not interested in the technical situation, you're not even interested in spies, and whether they might be counter spies and double-double agents and all this.

Bruce Christianson: To some extent the decision about whether to have a revocation list or freshness certificates is an arbitrary one. If the security policy were different you could just change your mechanism to match, is that not so?

Tuomas Aura: Well, it's not an arbitrary decision, because we're making it clear that there are choices but if there's no information then we trust. In the case of a freshness certificate, when refreshing the certificate you would have to make the decision that if there's no information then you don't trust.

Bruce Christianson: Yes, but that's a *policy* issue. Your *mechanism* still works perfectly well in either case.

Tuomas Aura: Yes, true. You could do that.

[3] The possible existence of a state of affairs in which a button *ought* to be pressed but *cannot* be is a denial of the "Ought implies Can" axiom familiar from deontic logic. In the present, rather Diodorean, system context such an axiom (*e.g.* COpFp in Prior notation) can be regarded as a form of liveness property. The imposition of a corresponding safety property is more problematic: for example naïve assumption of "Ought Not implies Cannot" apparently commits us to "Ought Not implies Is Not" (CONpNFp=CFpPp, whence CpPp=CONpNp since CpFp). Of course even at the meta-level the admission that a particular application system morally ought to possess a particular safety or liveness property does not entail that the system can in fact be so constructed (although it *ought to imply* that it can!), nor conversely.

PIM Security

Dieter Gollmann

Microsoft Research, Cambridge

Today, one frequently encounters application scenarios where users equipped with multiple mobile devices want access to information wherever they are, whenever the information is required. One particular aspect of this vision are Personal Information Management (PIM) systems providing access to a user's personal information. Personal information may relate to a variety of domains, like employment, membership in an organisation, and the user's private sphere, each with its own security requirements.

The basic systems architecture for such a service consists of servers storing the information, of mobile devices carried by the user, and communications protocols that give access to the information. Superficially, the security problems may seem the same as in any other client-server architecture: Protection of the data in transmission and access control to the data on the server. PIM security (PIMmS) is, however, crucially different from standard client-server security. Looking at the differences between access control within an organisation and access control in a 'public' service may help to highlight the specific features of PIMmS. In the first case, the players are the *members* of the organisation and its *management*. In PIM, the players are *data owners* and the PIM *service provider*.

Privacy is one distinguishing factor. In an organisation, the data stored on the server usually belongs to the organisation. The organisation has by necessity access to some personal information of its members, and may even have a duty to know what its members are storing on the server. In PIM, there is no necessity for the service provider to keep information about customers beyond, say subscription numbers and subscription payments. For privacy reasons, service providers may even wish to be in a position where they cannot access customer data. (Of course, service providers may have other business reasons to claim ownership of their customers' data.)

A second distinguishing factor is security management. Within organisations it is possible to organize security management, allocate responsibilities to properly qualified staff, and take actions against individuals who fail to fulfill their responsibilities. In a public service, the end users have to take on responsibilities for security management (e.g. guard passwords or private keys, access sensitive services only from trusted devices, set their own policies) independent of their qualifications and the service provider is not in a strong position to sanction end users. After all they are the paying customers. (The music distributors may show how long one can annoy the people whose money one wants until they walk off.) Careless users may bring a service into disrepute and scare away more security conscious customers.

Next, PIM security policies can be truly 'discretionary'. The identifiers used when setting access control policies may be defined by each data owner. Data

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 80–81, 2002.

owners do not have to reveal to the PIM server the identities of the partiers they are granting access to their own data. The PIM server has to know about data owners and their policies but need not identify and authenticate third parties requesting access. To gain access, it suffices to be the holder of a valid *authorisation certificate*. However, such a solution poses again security management challenges. When data owners hand out authorisation certificates to the wrong parties, it is entirely their own problem. The PIM server does not know and cannot help. When data owners store their data in encrypted form and keep their keys, they lose access to their own data when they lose their keys. The PIM server does not know and cannot help.

If the user's policy is expressed using identities defined by the service (PIM identities), the service can authenticate parties requesting access, but one has to guarantee to the data owner that a given PIM identity corresponds to the person the user has in mind. If a person has more than one PIM identity, either all of them have to be entered in the relevant access control structure or that person has to make sure the appropriate identity is used when accessing data. This can be done, but can easily become a nuisance. If for some reason a PIM identity changes 'ownership', the user's policy would be changed.

To address these management problems, a PIM service could try to help users when setting their policies, but gain access to privileged information at the same time. A PIM service having access to private information may advertise a privacy policy to inform its customers about the way their data will be used. In such a situation, audit is an excellent tool for users to monitor usage of their data. It is also a necessary tool for the management of a service provider to check that their organisation adheres to its advertised privacy policy.

In summary, PIMmS accentuates the trade-off between responsibilities and privacy guarantees. When data owners take full responsibility for their data, they are in charge of their own privacy vis-à-vis the PIM service, which just needs to guarantee the availability of protected information. When data owners want more support from the PIM service, the PIM service becomes responsible for protecting the private information of its clients. Clients no longer have technical guarantees that the PIM service cannot misuse their information and rely on procedural controls and business practices at the PIM service provider. There exist of course intermediate solutions like key recovery services, where a number of parties have to collude to compromise the technical privacy protection mechanisms.

PIM Security (Transcript of Discussion)

Dieter Gollmann

Microsoft Research

This is not the first time I've come across the problem I'm going to talk about in personal information management systems. It's one further example of a trend I'm seeing developing and I want to talk about the security problems I see this trend will create. It's about open questions rather than about solutions. The main conclusion as ever is that it's an issue of security management, and you have to think hard about the type of security management you're letting yourself into.

So, there is a story. The story starts with users that have all sorts of wonderful mobile devices that they carry with them. I don't possess a single one of those and I'm still happy, but that is the future and I'm still lagging behind. So, you have this variety of mobile devices. You want access to your personal information whenever you need access and wherever you are, so you need a system that allows you to access your personal information. That's why they're called personal information management systems. Part of the security issue here is that what is deemed personal information might relate to my employment, might relate to organisations that I'm a member of, might be my own private information, private credit card and so on and so forth, and I don't necessarily want to give everything to the same entity and apply the same set of access rules. So I have different ideas for what to do with different pieces of my personal information.

So that is the starting point setting the scope. In the obvious architecture, you have a server, the server stores the information somehow, you have the mobile devices, and you have various protocols to access your information when you need it, where you need it. So superficially you have a typical client-server scenario. My mobile device is the client, there is a server, and I want secure access from my client to the server. You might think of access control protection of data on the server and communications security protection of data during transmissions. I'm claiming that the situation is not really the same in the scenario of personal information management systems, and the way I'm trying to explain the story to myself is that you have to distinguish between access control within an organisation and access control in what I call a public service.

When I'm talking about an organisation like Microsoft, there are the members of the organisation and there is very clearly an entity called management, and there are certain relations between the members and the management. In personal information management I'd rather talk about data owners, the users with their personal information, and a service provider. As I'll try to explain later on, there are differences between these two scenarios. As a footnote, people round here talk about a different aspect of personal security — personal security assistants where you have all your relevant personal information in the personal

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 82–86, 2002.

device you're carrying, and use that private information to access services. I'm talking about a different scenario, where I want to have this information still on the server.

So, one area where there are differences is privacy. My employer has to know certain information about me, where I live, how much I earn, hopefully they get it right, sometimes they have problems, and so on and so forth. The organisation might also have legal requirements to know what I'm doing on their server. It might be deemed that they have to ensure I'm not downloading pornography and so on. In personal information management there is no necessity from the way I've described the service for the service provider to know anything about me. The service provider somehow has to keep the information, ideally without having access to it. Ideally from the privacy point of view, but not in general, service providers may wish to be unable to have access to personal information. I once spoke to a company in Germany which thought about offering an electronic archive as a service. They would keep all your electronic information for you, and guarantee that it would never be lost, and whenever you wanted access you'd get it. They were well aware of the privacy issues so they thought at one stage, why don't you the user encrypt all the data? We store it for you; if you want it we give it back to you encrypted and you can decrypt it. I'll come back to this story later.

There are further differences with respect to security management. In theory, at least, within an organisation the management of the organisation can hire properly qualified people to do the security management, have security policies in place, check that people are using proper passwords, discipline people who are not following the rules, even throw out employees who constantly violate security policies. It's again the relation between the members of an organisation and the management. In a public service users must participate in certain security management activities like they have to choose a good password, remember it and not disclose it, if they have private keys they have to protect them, not disclose them, they have to apply good judgement from where to access the private information — maybe not from an Internet café in a dodgy place. You probably can think of more examples. You're now moving to a situation where you have a security unaware user and you're setting up a system where that security unaware user is taking on a rôle in a distributed security management structure. That is a change, and your ability to discipline the users are somewhat limited — they are your paying customers. Where do you draw the line? When do you tell a user we terminate your service because you are constantly bringing our service into disrepute?

A very different situation is the music industry and their ideas about controlling what their users are doing and what they are sharing. The question is, how long will this work? How long can a service provider inconvenience its paying customers before the paying customers do something else.

Tuomas Aura: There is something similar with credit rating agencies. Customers who misbehave are kicked out and then when they try to join another service they are told no, you have bad security, go away.

Reply: Hmm. . . a security ratings agency? For the purpose of organising this security workshop I had to put my credit rating on the line by helping to open a bank account. That's a similar story.

Another difference is in terms of access control. I have particularly the Hailstorm example in mind, which offered you a means of defining a group of friends who would get access to certain information.

In theory, access control policies in a personal information management system could be properly purely discretionary, in the sense that I declare my own policy using my own identifiers. I don't have to use prescribed system identifiers; I don't have to reveal to the service provider who my friends are; I only have to give the right tokens or authorisation certificates to my friends and they will get access. But now I have to trust my friends that they don't give the authorisation certificates away. That is a change. The PIM server would not need to know about my friends, the PIM server only has to enforce my policy.

So, as I said before, if data owners hand out authorisation certificates to the wrong people, that's their problem. The service cannot help, the service doesn't know. Coming back to the example I had before, where the service provider gives users the opportunity to store their data encrypted and the user keeps the key, if the user loses the key then the user no longer has access to the data and the service cannot help. So at that stage the German company decided that using cryptography and leaving the keys with the user was not the solution to the problem and they would have to think of something else. After all, they were providing a service. If you now go to a solution where you allow the service to take a greater rôle, for example, by expressing security policies with service identifiers, PIM Ids as I've called them here, then you need guarantees that the service user corresponds to the person the user has in mind. In the Hailstorm scenario, you would use a Hailstorm identifier. How do I know that an identifier really corresponds to the person I'm thinking of? Somehow the service has to take on the rôle of a certification authority. At the policy level I will say Tuomas is allowed to read this data. If a person has more than one identity at the service level, I don't want to say that these five identifiers corresponding to him are allowed to read the data. If I only know about one then the problem is on his side that whenever he wants to read my data he has to remember which identifier to use. There are all sorts of nasty management problems around here.

What I see is a trade off. If the service tries to help users to set the policies, to enforce the policies, to authenticate third parties trying to access personal information then it is fairly frequently the case that you give private information to the service. You could then say, right, advertise a privacy policy, there are various schemes around. If you do that you might think of whether an audit could help you check that the service provider is adhering to its policies. That would also be a good way for the management of the service provider to check that they're doing what they're claiming they're doing in terms of adhering to privacy policies.

If I as a data owner want to take on full responsibility for my data, then I can have total privacy vis-à-vis the PIM service, but I have taken on the respon-

sibility. If the service wants to give more support, takes on more management responsibilities, I have to release part of my privacy. I don't see how one can extract oneself from this situation. As far as privacy guarantees are concerned, I don't see that there is much scope for technical guarantees that the service will use the data only the way I want the data to be used. It's more about procedural control and business practices. Sometimes intermediate concepts like threshold schemes are mentioned. Why don't you distribute the keys to different parties in a threshold scheme so only a certain number has to behave according to the rules, and you can cope with a certain amount of compromise? It works in theory. Has anyone a clue about whether that's backed up by a real business case?

And the final question, can you really be in control of your information? Can there be technical guarantees that the party holding your information will enforce your access control policy on data they are holding? Are you competent to manage your own security?

Pekka Nikander: If I go back to the very beginning I was just wondering why would you want to store your personal information on a single server? Certainly I wouldn't want to do that. I came up with two reasons. One is maybe you want to share some of that information with some other people, and the other reason might be that you are scared that you'll lose your equipment and you won't risk all your data. But I could use something like a PIM Napster — distribute all my personal information with everybody else having these personal information devices, and maybe striping it in the process so that nobody has all my personal information but just one bit of it.

Reply: Yes, but how much relevant personal information have you got that you can distribute it that easily, that's the one question.

Tuomas Aura: With personal information like your mail box, you want to be able to look at it from this device and that device and everywhere, wherever you are. I'm not sure how quickly Napster and Freenet respond to your request. How fast do you get the information back?

Pekka Nikander: I have got something like 200G of disk. I've got enough storage space to store all my personal information a hundred times over.

Tuomas Aura: So why do people actually keep their mailboxes on servers, and not in their pocket?

Roger Needham: I don't think there's any question of saying you put all your information on a server. Correct me if I'm wrong, but I think it's saying you're offering a service which is much more heavily backed-up than the user's home PC.

Pekka Nikander: It's the same if I share my data with my best friends on the Internet.

Reply: To come back to your question, one of the business cases I have seen made was an article in Communications of the ACM[1] about personalizing web pages. You go to a website, they require subscription information. The argument

[1] "Personalization on the Net using Web Mining", Communications of the ACM, Volume 43, Number 8, August 2000.

is the usual electronic commerce argument — electronic commerce is held back because different websites have different formats so the poor customer cannot cope with all this variety, therefore you need automated support. They look at the web page, they see what's required, they have access to your personal information, and fill out the page for you. I have to stress that the article was not anything to do with Microsoft. It was comparing different solutions to this problem. So seemingly people think there is a market for that type of service.

The other business case came from Ericsson. The idea is you have your mobile phone and now you get your e-mail on your mobile phone and your mobile phone will not hold all your gigabytes of e-mail.

Tuomas Aura: Is it comparable to single-signon?

John Ioannidis: Single-signon has a reason for existing, but it is not a panacea. It only makes sense in tightly controlled environments such as within a company.

Mark Lomas: You said a service provider could abuse access to the information. I would actually divide that into two aspects. One is the institution providing that service chooses to abuse your information, and the other is a rogue individual within the organisation and I think that's where the threshold system is actually valuable. For instance, let's say Microsoft provides the service. They'd be able to say, no one individual within our organisation can look at your personal information; however, if you forget your password we can get through to people who are scattered between, say, three different offices who will recover your data for you.

Matt Blaze: In the first information hiding workshop I wrote a paper called Oblivious Key Escrow[2] in which you distribute keys essentially at random on the net that could be put together using a threshold scheme. Anyone can recover them, but it's very hard to do it in secret. Perhaps that's the right solution.

Reply: Which gets you back to my last question here. Are your users competent to use oblivious key threshold schemes?

[2] "Oblivious Key Escrow", in "Information Hiding: First International Workshop", Springer Lecture Notes in Computer Science number 1174.

Merkle Puzzles Revisited – Finding Matching Elements between Lists

Bruce Christianson[1] and David Wheeler[2]

[1] Computer Science Department, University of Hertfordshire, Hatfield
[2] Computer Laboratory, University of Cambridge
England, Europe

Abstract. Consider the following problem. A and B each have a N-element set of bit-strings. They wish to find all collisions, in other words to find the common strings of their sets or to establish that there are none. How much data must A and B exchange to do this?

Problems of this type arise in the context of Merkle puzzles, for example where A and B propose to use the collision between two randomly constructed lists to construct a cryptographic key.

Here we give a protocol for finding all the collisions. Provided the number of collisions is small relative to $N/\log_2 N$ the protocol requires on the order of $\log_2 N$ messages and the total amount of data which A and B need exchange is about $4.5N$ bits. The collision set can also be determined in three messages containing a total of at most $9N$ bits provided $N < 2^{1023}$.

1 Introduction

Suppose that A and B each have an N-element set of fixed-length bit-strings. By first hashing with an appropriate random hash function if necessary, we can assume that the bit-strings appear to be uniformly distributed. A and B arrange their respective strings in lexical order, and form a table. They may use a fixed part of each string rather than the entire string, provided the parts are sufficiently long to distinguish all pairs of strings.

A *collision* is a string which is in both tables. It is assumed in what follows that the number of collisions is small relative to $N/\log_2 N$. It may be known *a priori* that there is exactly one collision, or some other small number, or the precise number may be unknown, and may include zero. A and B wish to find all collisions between their tables, by exchanging only a small amount of data per table entry.

In the next few sections, we discuss the properties of some protocols to do this. It turns out that the amount of data which the partners are required to exchange can be made less than five bits per list element, regardless of N. Following this, we give an application of our solution to cryptography, in the context of a Merkle puzzle.

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 87–90, 2002.

2 Basic Protocol

The basic protocol is a recursive construction. Imagine a list containing, in lexical order, all the 2^n possible table entries (not just the actual ones). We divide this list into a number of intervals of nearly equal size. Initially this number is chosen to be $M = N \log_2 e \approx 3N/2$ where N is the number of entries actually in each table. We say that an interval is *occupied* for A if it contains an entry which occurs in A's table, otherwise it is *empty* for A. The number of intervals M is chosen so that the probability of each interval being occupied is about $1/2$ for each participant, and all these probabilities are nearly independent.

Each party informs the other which intervals are occupied and which are empty. This communication requires one bit per interval, and the coding of the information is optimal. On average half the intervals will be empty for each participant. Only intervals which are occupied for both participants survive to the next round.

Eliminating the intervals which A regards as empty obviously eliminates no entries from A's table, although it does eliminate about half the entries from B's table. However the information from B will eliminate roughly half of the intervals which A regards as occupied, as well as half the intervals which A regards as empty, since their selections are uncorrelated. For each participant, on average half the table entries and one quarter of the intervals thus survive to the next round.

The next round begins by dividing each surviving interval in half. This restores the ratio $N/M = \log_2 e$, so that the probability of occupation is still one half, but the size of the problem is now halved.

3 Message and Bit Counts

The protocol continues for about $\log_2 N$ rounds. The i-th round exchanges two messages (one in each direction), each containing $M2^{1-i}$ bits of information.

However, if B waits to receive A's messages before responding, the length of B's messages can be halved.

The number of messages can also be reduced if A and B take it in turns to send the first message of a round combined with the second message of the previous round giving $\log_2 N$ messages.

In this case the first message (from A) contains M bits, and the i-th message (from A or B according as i is odd or even) contains about $M2^{2-i}$ bits for $1 < i \leq \log_2 N$.

The total bit-length of all messages is thus less than the geometric sum $3M$. (A's total is $M + 2M/3$ bits. B's total is $4M/3$.) This is less than $4.5N$, ie less than five bits per table entry.

In all cases to eliminate chance we send extra bits of the candidate collision strings to verify the collision. This requires at most $\log_2 N$ bits per collision, which is negligible since the number of collisions is assumed small relative to $N/\log_2 N$. Each collision also causes an interval to be occupied throughout the

protocol: this adds one bit per collision per round, which is also negligible by hypothesis on the number of collisions.

4 Effect of Statistical Deviations

At each round, small statistical deviations from the expected ratios occur. Although the relative size of the deviation introduced at each round increases as the protocol proceeds, the effects are not cumulative: deviations from the previous rounds tend to be systematically damped out.

To see this, assume that at the i-th round there are M_i occupied intervals and N_i^A entries in A's table. Let $\lambda_i^A = N_i^A/M_i, \epsilon_i^A = \lambda_i^A - \log_e 2$. To first order, the proportion of occupied intervals for A is $1 - \exp -\lambda_i^A = (1 + \epsilon_i^A)/2$. Neglecting the statistical errors introduced during the i-th round, we therefore have to first order $\epsilon_{i+1}^A = (1 - \log_e 2)\epsilon_i^A \approx \epsilon_i^A/3$ and similarly for B.

5 Reducing the Number of Rounds

We may wish to restrict the number of rounds. This can be done by pipelining several rounds into the same pair of messages, at the cost of a modest increase in the total number of bits required per initial table entry.

If A places 16 rounds in the first message, then B can produce a second message of almost equal length containing 2^{16} rounds. If the table contains more than 2^{65536} entries, then A can reply with a third message containing $2 \uparrow (2^{16})$ rounds, and so on.

In practice a reasonable trade-off might be for A to place two rounds in the first message, and B to place ten rounds in the second message. Provided $N < 2^{1023}$, a total of three long messages suffices, with a combined total $6M < 9N$ bits. This is twice the minimum value required by the basic protocol. A fourth short message is required if both parties wish to know the locations of the collisions.

6 Application to a Merkle Puzzle

Suppose that A and B wish to agree a 40-bit shared secret. They publicly agree a prefix c, and then each generates a few (k) million 40-bit random values r_i. Each then forms a table (in lexical order) of the values $h(c|r_i)$, where h is a randomizing hash function.

By the birthday paradox, the tables are likely to contain about k^2 collisions, and these can be found by the participants using the protocol described above, exchanging on the order of $10k$ million bits of data. The value of r corresponding to the first collision is the shared secret, or alternatively the unused bits of the collision. The work required by an attacker to find r is of the order of 10^{12} rather than 10^6.

If no collisions are found then the exchange is completely repeated or else N is increased, until a solution is found. The chance that no collisions are found is about $\exp -k^2$.

7 Conclusion

We have shown that Merkle-type puzzles can be solved at a total communications cost which is less than five bits per table entry, regardless of the of the number N of table entries, provided that the number of messages exchanged is allowed to increase to $\log_2 N$.

The usual protocols for a Merkle puzzle require one message in each direction, effectively forcing one of the participants to transmit their table to the other, at a communications cost which increases nonlinearly with the number of table entries. The precise cost under this message restriction depends upon the details of the protocol used. But even here, a modified form of our protocol requires on the order of $1.5 \log_2 N$ bits per table entry, which is considerably fewer than Merkle's original algorithm.

For example, for a 40 bit key with $\log_2 N \approx 22$ Merkle requires A to send about 70 bits per table entry, whereas the one-shot version of our protocol requires less than half this amount.

Alternatively, both the bits per table entry and the number of messages can be held to $\deg_2 N$, where $\deg_2 1 = 0, \deg_2 N = 1 + \deg_2 [\log_2 N]$.

Merkle Puzzles Revisited (Transcript of Discussion)

Bruce Christianson

University of Hertfordshire

I'm going to break with tradition and talk about an actual security protocol. This is a small part of a problem which arose in the context of something that David Wheeler and I were looking at, and we didn't think anyone else had written about it.

The setting for the problem is that A and B each have a table of bit strings. There don't have to be the same number of strings in each table, but it's simpler to explain in the case where they do. The bit strings don't all have to be the same length either, but again it's conceptually easier if they do. So N is the number of bit-strings in each table, and each bit-string is n bits long. And there is a certain number k of *collisions* between the tables. By a collision I mean a bit string that is in both tables, although it may be in a different place in each table.

Now A and B want to discover all the collisions between their tables. It may be that they know *a priori* that there is exactly one collision and they just want to find out what it is. It may be that it's possible that there are no collisions, and they want to know that there aren't any. (This is sometimes the case with the original Merkle key agreement protocol.) Or it may be that there are lots of collisions and they want to find all of them. (If you're doing digital money and you're trying to put watermarks in digital money in such a way that no-one can cheat, then very often you know that there is a number of collisions that's reasonably large, about of the order of the square route of the number of things in the table.) For technical reasons which will become apparent later, we assume that the proportion of collisions is small, that k is small relative to $N/\log_2 N$. But it suffices for this if k is of order $\sqrt{N}$.

We assume that the bit strings are randomly distributed. If they're not, then just put them through a random hash function and use the values of the random hash function instead of the original bit strings. And we assume that the bit strings are long enough that different bit strings can actually be distinguished. So $n > \log_2 N$. Again, that's a technical assumption that you can get rid of if you want, by using encoding rules.

The classic application of this is the Merkle key agreement protocol. Here you have some value c that's publicly agreed but that can't be predicted in advance by an adversary. Then A and B each pick a few thousand random values r_i and form a table of the values $h(c|r_i)$. They take the (lexically) first collision in their table as being the key that's agreed. The adversary will have to do on the order of a million computations rather than on the order of a few thousand to obtain the key. You can scale this up so that you're talking about 2^{30} and 2^{60}

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 91–94, 2002.

instead of a thousand and a million. And, as David explains in his talk[1], you can build protocols which enable you to be reasonably confident that the beacon c is available to you and your partner but not to the adversary. So in this case you're simply putting the extra bits in your keys in order to ensure that the adversary cannot subsequently verify a guess about the beacon. This way you can get keys that really are long enough for all practicable purposes. As I said, you can embed them as watermarks in digital money, and so forth, and reveal them a bit at a time[2].

It's basically the birthday paradox. If you've got 40 bits of random number and you each pick p random numbers, then there's about $k = 2^{-40}p^2$ collisions, and your failure probability is about $\exp(-k^2)$, which is exponentially decaying as k gets large.

Now the question which this talk addresses is, how much data do A and B have to exchange to find the collisions? We know how much computation they have to do, the question is how many bits do they have to send to each other. The answer is, they have to send about 4.5 bits per table entry irrespective of the size of the table. If A just sends her entire table to B, and B compares the two tables and picks out the entries that match, then you're sending something that's of order $\log_2 N$ bits per table entry, and 4.5 is much smaller than that for large N.

The basic construction is a simple recursion. Consider a table C consisting of all possible n-bit patterns — not just the ones that we've actually got but all 2^n that there could possibly be — arranged in some order. This doesn't have to be lexical order but it's the simplest so assume that it is. Define M to be the nearest integer to $N/\log_e 2 \approx 1.5N$, and divide C into M consecutive *intervals*, *i.e.* groups of contiguous entries of as nearly as possible equal size. For each interval, the probability that that interval contains at least one actual element from A's table is almost exactly a half. So A sends to B a series of M bits, one for each interval, 0 means not occupied, 1 means occupied. The bits are independent and occur with probably a half, so the coding is Shannon-optimal. This eliminates about half of the intervals from C, and eliminates about half of the entries from B's table. Similarly half of the intervals from C are not occupied by an actual entry from B's table, and these two factors are independent. So once B has sent a message in the opposite direction, about a quarter of the intervals and a half of the entries in each table survive to the next round. A and B then divide each interval into two, thus restoring the ratio 1.5 between the number of intervals and the number of table entries. Now they have a problem that's almost exactly half the size of the previous one.

A and B each sent M bits, which is about 3N bits in total, and this halved the size of the problem. So you have about half that number of bits exchanged in the next round, and so on. It's a geometric sum, that adds up to 6N. If A and B are willing to take turns, in other words, if B is willing to wait until he gets A's first message before he sends his first message, then all the lengths of

[1] David Wheeler, these proceedings.

[2] as suggested by Rivest and Shamir, LNCS 1189, p 85.

B's messages are halved, and that gives 4.5N bits in total, which was the result that I asserted at the beginning.

Each collision leads to an interval being perpetually occupied, since an interval with a collision in it will survive through every round. But by assumption $k \log_2 N << N$, so the perpetual occupancy rate is small enough that this comes to a negligible proportion of a bit per table entry. A purist analysis says, the number of rounds you need is logarithmic in N because it halves N at each round and so therefore after $\log_2 N$ rounds you're down to 1. Now of course that isn't exactly true because the number of intervals won't exactly halve at each round, you will get small statistical deviations from the expected ratio of a half. What you can show is that those deviations are not amplified as the rounds proceed, in fact they're damped by a factor of 3, which is nice. What that means is that by the time you get down to the point where the number of table entries is significantly different to what it should be, the number of entries you're left with is sufficiently small that you can simply blurt out what's left of your table and this won't significantly increase your bit count per initial table entry. The details are in the paper.

But $\log_2 N$ is rather a lot of rounds. You might not want to wait for that many messages. In that case you can pipeline the rounds. A doesn't have to wait for B to respond to the first round before sending the information for the second round. In fact A can send the information for both rounds in the same message. The cost of that is that A will send some unnecessary bits. A will be sending bits corresponding to intervals which, if A had waited for B's message, A would have been able to eliminate and save the corresponding bandwidth. However the interesting thing is, even if you send one message — or alternatively one long message and one short one, if both parties want to know what the collision is — this only requires $1.5 \log_2 N$ bits per table entry. This is still a lot better than Merkle's original algorithm, and for the kind of jelly-bean numbers that you have to use to beat the NSA now, it really is quite a lot better.

Alternatively, if A places 16 rounds in the first message, then B can place 2^{16} rounds in a second message of the same length. A can then place 2^{65536} rounds in a third message of the same length, and that's enough for all practical purposes, given that the number of rounds is the logarithm of the number of table entries. As a compromise, A can place say 2 rounds in the first message, then B places 10 rounds in the second message, there's a total of 3 messages — or 3 long messages and one short message if they both want to know all the numbers — and that's 9 bits per table entry, twice 4.5. And that works provided N is less than 2^{1023} which it probably is. Again, the point is that 9 bits per table entry is still a lot shorter than what you would otherwise get.

We've shown you can have constant bandwidth per table entry, provided you're willing to have, in principle, something that's logarithmic in the number of rounds. You can have a constant number of rounds, you can have just one round provided you're willing to accept a number of bits per table entry that's logarithmic in the table size. What happens if you want equal rounds and bandwidth? The answer is this. Define the *degree* function as follows: the degree of

1 is zero, the degree of N is one more than the degree of the integer ceiling of $\log_2 N$. So, for example, the degree of 2 is 1, the degree of 4 is 2, the degree of 16 is 3, the degree of 65,536 is 4, and the degree of $2^{65,536}$ is 5. Now you can get the number of rounds and the bandwidth both equal to the degree. The application of this is reasonably obvious.

Markus Kuhn: Is there a simple solution for the dual problem, where we have two huge pools of strings and I have a few strings that you don't have and you have a few strings that I don't have, how can you find out which ones are in the symmetric difference.

Reply: Yes, that's an interesting problem[3]. There are a number of other problems to which generalisations of this apply. For instance, if we have databases of people with bad credit records and we want to know if there are people who are in one database but not in the other, but we are prohibited from revealing information about people that we don't have a reason for sharing information about. Or we're investigating two crimes and we have a suspect and we want to know if they're the same suspect relative to a particular database and so forth. I'm confident that there are variations of this that apply to these related problems.

The original application that we had was a Merkle-like puzzle that was embedded in a protocol we were designing and that's closely related to a problem in designing digital currency with watermarks which you can reveal as the month goes on and which makes it hard for a corrupt individual in the mint to forge money.

[3] Divide each table into blocks and transmit the checksum of each block. Eliminate blocks with the same checksum and split those that remain.

Encapsulating Rules of Prudent Security Engineering (Position Paper)

Jan Jürjens*

Computing Laboratory, University of Oxford, GB
jan@comlab.ox.ac.uk – http://www.jurjens.de/jan

Abstract. In practice, security of computer systems is compromised most often not by breaking dedicated mechanisms (such as security protocols), but by exploiting vulnerabilities in the way they are employed. Towards a solution of this problem we aim to encapsulate rules of prudent security engineering in such a way that a system specification formulated in (a formal core of) the Unified Modeling Language (UML, the industry-standard in object-oriented modelling) can be evaluated wrt. these rules, violations be indicated and suggestions for modifications be derived.

1 Introduction

In the context of computer security, "an expansive view of the problem is most appropriate to help ensure that no gaps appear in the strategy" [20].

However, it has remained true over the last 25 years that "no complete method applicable to the construction of large general-purpose systems exists yet" [20] that would ensure security, inspite of very active research and many useful results addressing particular subgoals [19].

Thus problems often arise from design limitations and implementation errors rather than defects in security mechanisms [3,4]. Therefore research in avoiding design and implementation errors is one of the main challenges in computer security research [19].

For instance, in the case of GSM security [21], some examples for security weaknesses arising in this way are

- the failure to acknowledge limitations of the underlying physical security (misplaced trust in terminal identity; false base stations),
- an inadequate degree of flexibility to upgrade security functions over time and
- lack in the user interface wrt. communicating security-critical information (no indication to the user that encryption is on).

* Supported by the Studienstiftung des deutschen Volkes and the Computing Laboratory.

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 95–101, 2002.

We aim to draw attention to such design limitations during the design phase, before a system is actually implemented.

We use a formal core of the Unified Modeling Language (UML [18], the industry-standard in object-oriented modelling) to encapsulate knowledge on prudent security engineering and thereby make it available to developers of security-critical systems [13,11,12,10]. More precisely, we use a fragment of UML together with a formal semantics (which helps us formulate our concepts). So far, there exists no universally agreed-upon formal semantics for all of UML, but even if there will never be one, one can use our approach (by incorporating the security checks into a tool and explaining them informally, just as programming languages are usually used without a formal semantics, however unsatisfactory this may be).

Currently a large part of effort both in verifying and in implementing specifications is wasted since these are often formulated imprecisely and unintelligibly [16]. Being able to express security-relevant information in a widely used design notation helps alleviate this problem (especially if this allows designers to reuse concepts and results formulated in this notation). Additionally, this may reduce misunderstandings that may arise between different parties involved in designing and evaluating security-critical systems (cf. e.g. [6]).

Since we are using a simplified formal fragment of UML, we may reason formally, showing e.g. that a given system is as secure as certain components of it. Furthermore one may go beyond formal verification and make use of techniques more feasible in practice, such as specification-based testing (e.g. following ideas in [15]).

In this position paper, we explain our approach by showing how it relates to the principles of security engineering set out in [20], using examples from earlier work (for which the details have to be omitted and can be found in the respective references given). We also mention briefly how one might apply our approach to investigate security protocols in the system context.

2 Design Principles for Secure Systems

We demonstrate by examples how our approach relates to the rules stated in [20].

Economy of mechanism. Our approach addresses this "meta-property" by providing developers (possibly without background knowledge in security) with guidance on the employment of security mechanisms who might otherwise be tempted to employ more complicated mechanisms since these may seem more secure.

Fail-safe defaults. One may verify that a system is fail-safe by showing that certain security-relevant invariants are ensured throughout the execution of the system, i.e. in particular if the execution is interrupted at some point (possibly due to malicious intent of one of the parties involved). An example is secure log-keeping for audit control.

As an example for modelling audit control with our approach we give a part of the specification of the unlinked load transaction of the smart-card based Common Electronic Purse Specifications (CEPS) [5] given in [11]. The simplified behaviour of the card is given in Figure 1 (since it just serves as an illustration for our approach we omit the explanation which can be found in [11]). We use

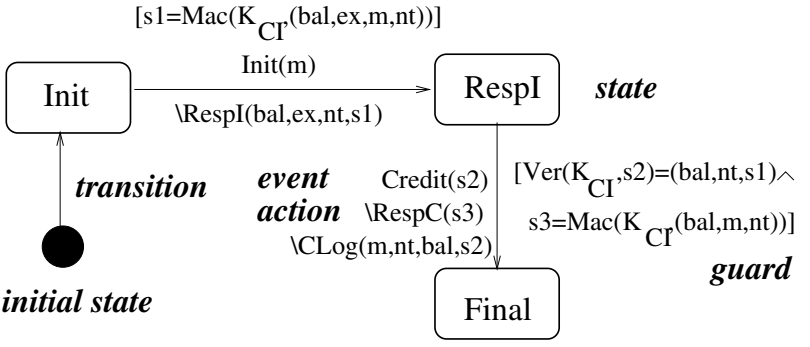


Fig. 1. Statechart for card

a UML statechart diagram, which is a notation for state machines where, intuitively, a label \Msg(args) on a transition means to output the message Msg with arguments args, Msg(args) means to trigger the transition on input of Msg whose arguments are assigned to the variable args and [condition] means to trigger the transition only if condition is fulfilled.

In the context of this load transaction, one aspect of audit security is that the cardholder should only be lead to believe (e.g. when checking the card with a portable card reader after the transaction) that a certain amount has been correctly loaded if she is later able to *prove* this using the card – otherwise the load acquirer could first credit the card with the correct amount, but later in the settlement process claim that the cardholder tries to fake the transaction. Thus we have to check the following audit security condition on the attributes of the audit log object CardLog (again this is just for illustration; details can be found in [11]).

Correct amount: $s2$ and $s1$ verify correctly (say $\mathsf{Ver}(K_{CI}, \mathsf{CardLog}.s2) = (bal', nt', s1')$ and $\mathsf{Verify}(K_{CI}, s1') = (bal'', ex'', m'', nt'')$), and additionally we have $\mathsf{CardLog}.m = m''$ (i.e. the correct amount is logged).

This way one can e.g. uncover unstated assumptions on the trust relations between the protocol participants on which its security relies (e.g. that audit security relies on the fact that the load acquirer trusts the card issuer; details cf. [11]).

Complete mediation. This principle can be enforced e.g. in Java by using guarded objects [8]. Their use however is not entirely straightforward [7]. We demonstrate how one can ensure proper use of guards to enforce this principle can be enforced with an example from [12].

Suppose that a certain micropayment signature key may only be used by applets originating at and signed by the site Finance (e.g. to purchase stock rate information on behalf of the user), but this access should only be granted five times a week.

The guard object can be specified as in Figure 2 (where ThisWeek counts the number of accesses in a given week and weeklimit is true if the limit has not been reached yet). One can then prove (informally or using a formal logic such as [2]) that certain access control requirements are enforced by the guards.

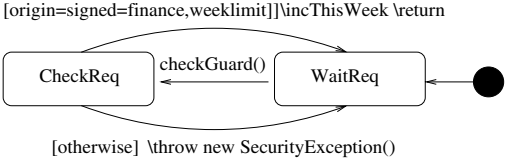


Fig. 2. Statechart MicGd

In this situation, a specification satisfies complete mediation if the access to every object is guarded by a guard object. More feasibly, one can specify a set of sensitive objects and say that a specification satisfies mediation wrt. these objects if they are guarded. One may then give a general policy that defines which access restrictions the guard objects should enforce.

Open design. Our approach aims to contribute to the development of a system whose security does not rely on the secrecy of its design.

Separation of privilege. As an example for an instance of this principle, one may easily modify the guard in Figure 2 to require signatures from two different principals on the applet requesting access to the guarded object.

In this context, a specification satisfies separation of privilege wrt. a certain privilege p if there are two or more principals whose signature is required to be granted p, at every point of the execution.

More generally, one can formulate such requirements on a more abstract level using UML activity diagrams and verify behavioural specifications (such as the ones given above) wrt. these requirements.

Least privilege. Given functionality requirements on a system, a system specification satisfies the principle of least privilege if it satisfies these requirements and if every proper diminishing of privileges of the entities in the system leads to

a system that does not satisfy the requirements. This can be formalised within our specification framework and the condition can be checked.

An example application for this rule are the access control rules enforced by firewalls, e.g. considered in [15].

Least common mechanism. Since we follow an object-oriented approach, this principle is automatically enforced in so far as data is encapsulated in objects and the sharing of data between different parts of a system is thus well-defined and can be kept at the minimum of what is necessary. Note that on the programming language level there may be further subtleties (e.g. one should not design public fields or variables that can be accessed directly [8]). It is intended to address these in our specification framework in future work.

Psychological acceptability. Wrt. the development process, this principle is addressed by our approach in so far as it aims for ease of use in the development of security-critical systems, and thus for the psychological acceptability of security issues on the side of the developers.

To consider psychological acceptability of security mechanisms on the side of the *users* of the system one may make use of work using UML for performance engineering (e.g. [17]) to evaluate acceptability of the performance impact of security mechanisms. One may also modify this approach to measure the user interaction required by such mechanisms (e.g. for authentication).

3 Protocol Contexts

It has been suggested (e.g. in [1]) to investigate the way security mechanisms (such as protocols) are employed in the system context, which in practice offers more vulnerabilities than the mechanisms themselves [4].

As an example, the security of CEPS transactions depends on the fact that in the immediately envisaged scenario (use of the card for purchases in shops) it is not feasible for the attacker to act as a relay between an attacked card (in a modified terminal) and an attacked terminal. However, this is not explicitely stated, and it is furthermore planned to use CEPS over the Internet [5, Bus.Req.], where an attacker could easily act as such a relay (this is investigated in [14]).

Sometimes such assumptions are actually made explicit, if rather informally (such as "R_1 should never leave the LSAM in the clear." [5, Tech. Sp. p.187]).

Being able to formulate precisely the assumptions on the protocol context, to reason about protocol security wrt. them, and to be able to communicate them to developers and clients would thus be useful.

In our UML-based approach, security protocols can be specified using message sequence charts. An example from [14] is given in Figure 3. Assumptions on the underlying physical layer (such as physical security of communication links) can be expressed in implementation diagrams (cf. [13]), and the behaviour of the system context surrounding the protocol can be stated using statecharts and reasoned about as indicated above.

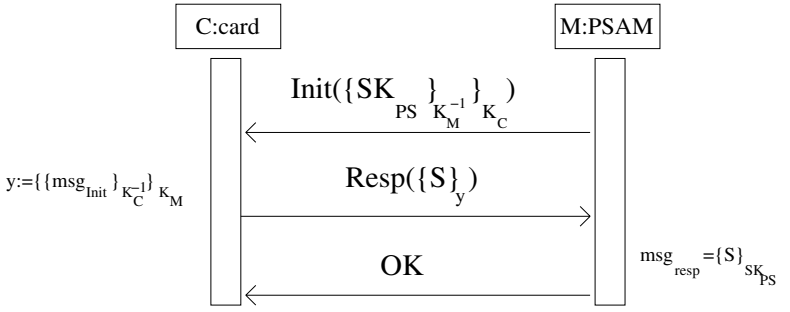


Fig. 3. MSC for CEPS purchase transaction

4 Conclusion

In this position paper, we used the computer security design principles put forward in [20] to illustrate our approach towards encapsulating rules of prudent security engineering using a formal core of the object-oriented design notation UML (the current industry standard in object-oriented modelling). Using this approach one may evaluate system specifications wrt. these rules, and obtain suggestions for modifications in case of violations. We also mentioned briefly how to use our approach to consider the security of protocols in the system context.

With this work, we aim to provide an "expansive view" of computer security and provide a method for development of large security-critical general-purpose systems, as requested in [20].

The research on using UML to develop security-critical systems is still in a very early stage; current work addresses the formal foundations of the approach as well as tool-support and application in case-studies (e.g. in the development of an Internet auction system in an MSc project currently in preparation).

Acknowledgements. The idea for this line of work arose when doing security consulting for a project during a research visit with M. Abadi at Bell Labs (Lucent Tech.), Palo Alto, whose hospitality is gratefully acknowledged. It has also benefitted from discussions with D. Gollmann, A. Pfitzmann, B. Pfitzmann and others.

References

1. M. Abadi. Security protocols and their properties. In F. Bauer and R. Steinbrueggen, editors, *Foundations of Secure Computation*. IOS Press, 2000.
2. M. Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, 1993.

3. R. Anderson. Why cryptosystems fail. *Communications of the ACM*, 37(11):32–40, November 1994.
4. R. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2001.
5. CEPSCO. Common Electronic Purse Specifications, 2001. Business Requirements vers. 7.0, Functional Requirements vers. 6.3, Technical Specification vers. 2.3, available from http://www.cepsco.com.
6. Dieter Gollmann. On the verification of cryptographic protocols – a tale of two committees. In *Workshop on Security Architectures and Information Flow*, volume 32 of *Electronical Notes in Theoretical Computer Science*, 2000.
7. Li Gong. Java^{TM} Security Architecture (JDK1.2). http://java.sun.com/products/jdk/1.2/docs/guide/security/spec/security-spec.doc.html, October 2 1998.
8. Li Gong. *Inside Java 2 Platform Security – Architecture, API Design, and Implementation*. Addison-Wesley, 1999.
9. H. Hußmann, editor. *Fundamental Approaches to Software Engineering (FASE/ETAPS, International Conference)*, volume 2029 of *LNCS*. Springer, 2001.
10. Jan Jürjens. Developing secure systems with UMLsec — from business processes to implementation. In *VIS 2001*. Vieweg-Verlag, 2001. To appear.
11. Jan Jürjens. Modelling audit security for smart-card payment schemes with UMLsec. In P. Paradinas, editor, *IFIP/SEC 2001 – 16th International Conference on Information Security*. Kluwer, 2001.
12. Jan Jürjens. Secure Java development with UMLsec. 2001. Submitted.
13. Jan Jürjens. Towards development of secure systems using UMLsec. In *[9]*, 2001.
14. Jan Jürjens and Guido Wimmel. Security modelling for electronic commerce: The Common Electronic Purse Specifications. In *First IFIP conference on e-commerce, e-business, and e-government (I3E)*. Kluwer, 2001.
15. Jan Jürjens and Guido Wimmel. Specification-based testing of firewalls. In *Andrei Ershov 4th International Conference "Perspectives of System Informatics" (PSI'01)*, LNCS. Springer, 2001. To be published.
16. L. Paulson. Inductive analysis of the Internet protocol TLS (transcript of discussion). In B. Christianson, B. Crispo, W.S. Harbison, and M. Roe, editors, *Security Protocols – 6th International Workshop*, number 1550 in LNCS, page 13 ff., Cambridge, UK, April 1998.
17. R. Pooley and P. King. The unified modeling language and performance engineering. *IEE Proceedings – Software*, 146(1):2–10, 1999.
18. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.
19. F. Schneider, editor. *Trust in Cyberspace*. National Academy Press, 1999.
20. J. Saltzer and M. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.
21. M. Walker. On the security of 3GPP networks. In *Advances in Cryptology – EUROCRYPT*, volume 1807 of *LNCS*. Springer, 2000.

Encapsulating Rules of Prudent Security Engineering (Transcript of Discussion)

Jan Jürjens

Computing Laboratory, University of Oxford

I am going to give an account of some work towards a method for designing, developing, and implementing secure systems, with a particular focus on the deployment of security protocols in the system context.

So I start out with the very classic design rules by Saltzer and Schroeder and in this talk I'm going to illustrate the approach by showing exemplarily how one can check or try to implement these design rules in the design process.

I will start with two comments by Saltzer and Schroeder in their paper of 1975, of a general nature. Firstly they argue that one needs an expansive view of the problem of computer security in order to address the various difficulties that are related with this subject. And secondly they argue that (back then in 1975) there was no complete method available which would allow one to construct secure systems in a reliable and correct way. I would argue that both of these statements are to a large degree still valid today, 25 years after they were issued. Of course the first one, since it's a statement of truth, it's not surprising, but the second one maybe is a bit more disappointing. Even though there has been a lot of work on very specific issues and a lot of successful work, this has not really led to a complete and general method for designing secure systems.

So what are problems in security design? Ross Anderson[1] pointed out that in practice attacks are often directed not against vulnerabilities in the cryptography that is being used in a system, nor against the security protocols themselves, but rather against the way these protocols are used in the system context, and maybe more general design limitations in the system. It is these problems I would like to address. What I'm using as a notation is a formal core of the unified modelling language (UML); this is the current *de facto* standard in object oriented modelling in industry. I would like to use this to encapsulate how to enforce these design principles by Saltzer and Schroeder, and more generally I would like to use an extension of this notation to encapsulate knowledge on security engineering so as to make it available to designers and developers who may not have a strong background in security themselves. So this is maybe part of my personal motivation why I thought this work would be useful.

Last summer I was visiting Bell Labs where there were some people working on a project developing wireless embedded-systems infrastructure. Obviously there are some problems with security but the people who were working on that at that time did not themselves have a strong background in security, so they

[1] Ross Anderson: Why Cryptosystems Fail, Communications of the ACM, November, 1994

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 102–106, 2002.

asked me to look at their design and see what might be wrong with it in that respect. That led me to the conclusion that security is increasingly an issue in commercial applications, and developers are not yet sufficiently trained to deal with this situation.

When I say using a formal core of UML, those of you who know UML may wonder what I exactly mean by that. UML altogether is a rather huge design notation and some of the parts are only meant to be used in a very early design phase, and they are intentionally not formal and probably will never be formal; so that's why I only use a fragment of this language and I use it together with a formal semantics and for that one should know that an official formal semantics, even for these parts, does not exist at this point but there are people working on that and even if it does never exist, for whatever reasons, then this approach would work because we are also happily using programming languages that don't have a formal semantics. Of course then the notions of security that I'm considering here would need to be implemented in tools so that people can use this approach without having a formal semantics. Then one should of course also explain what these security notions mean in a necessarily informal way, but this is really not the problem that I'm addressing in this talk today.

Part of the motivation is also that there is a lot of effort being wasted in designing systems and implementing and maybe verifying them or testing them because specifications, if they exist, are not formulated very precisely because people may not have the time or the motivation to do so. So part of the motivation here is that if I have a method that gives people some useful help in designing their secure systems then this may actually motivate them to specify them in a widely known design language in the first place. UML is the candidate here because it is rather well known in the industry. Apart from this more general motivation, one can then use specifications written in this notation to firstly reason formally (as has been done using other notations for a long time now). But in industry in practice, it may turn out to be very expensive to prove the things correct, and people may not be willing to pay for that. So one might like to look for more less expensive methods of gaining some confidence that the system design is correct and secure and one way of doing this is specification based testing where you derive test sequences from your specification in a automatic way and this is also what I'm working on.

So I would start out looking at these design rules now and see how to relate my approach to these rules.

Firstly the rather general design principle, economy of mechanism. This is addressed indirectly by my method, since if I give guidance on how to employ security mechanisms and how to design a secure system then hopefully developers will be kept from using the most complicated security mechanism available simply because this may seem more secure psychologically than simpler ones. So in this way I indirectly address this principle.

Secondly, the principle of having fail-safe defaults. This could mean more specifically that one would like to ensure security-relevant invariants throughout the execution of a system, meaning that whenever a failure occurs, the system

fails in a secure state. As part of a case study regarding the Common Electronic Purse Specifications (CEPS[2]) that are currently developed by Visa International and other companies, I considered this principle specifically addressing the log-keeping for audit control in this electronic purse scheme. Using a simple state chart (one of the possible kinds of diagrams in UML) I can specify part of this protocol used in these electronic purse specifications, and in my work I was able to uncover unstated assumptions on the protocol and on the environments in which it is supposed to be used. I think this is actually an important result, even though this is not really a bug in the protocol (which is what one is usually looking for if one tries to verify protocols) because maybe, as I've pointed out earlier, in practice the problems arise often from the way protocols are used in the context of a system. The problems arise exactly because it is not clear what assumptions have to be made on the context of the system. If these assumptions are not even stated informally in the specifications, well, it cannot be clear. So the useful result of this approach was that I exemplified some assumptions that were made here without being made explicit in any way.

The next principle is that of complete mediation. In another case study I considered the Java security architecture and the jdk1.2, and here one can enforce this principle of mediation, using guarded objects. In the diagram I give the specification of such an guard object, using these specifications one can then verify that actually the access control principles that I would like to have enforced in my system are indeed enforced. This is also, in practice, less simple than it may appear and there are many examples of people simply forgetting guards or restrictive restrictions for access control, and by simply forgetting a guard for an object one may threaten the security of the whole system. So if I have a way of mechanically verifying that my access control principles are actually enforced by the objects this will be very useful.

Next the more general principle of open design. This is again addressed more indirectly using this method since the aim is to make it possible to develop systems whose security does not rely on the secrecy of the design: nothing new here really.

Next the separation of privilege. In the context of the previous example, this could mean that two or more principals are required to sign an object in order to be granted access. So again this could be formulated with some other kind of diagram and these behaviour specifications that I showed earlier can be verified back to these more abstract requirements.

And the principle of least privilege could be formalised by stating that whenever I reduce the privileges of different entities in the system, then the system is not anymore able to satisfy the functionality requirements and similarly this can be formalised and verified using this approach but I don't have the time to point this out in more detail.

Then the principle of least common mechanism. This is addressed insofar as the notation I'm using here follows the object-oriented approach where one has data encapsulation and in particular the sharing of data is well defined and I

[2] http://www.visa.com/pd/cash/ceps.html

can keep it at the minimum of what is necessary, so in this sense least common mechanism.

And then psychological acceptability. Firstly in the context of the development process itself, I'm trying to make it easier to develop secure systems by giving help and guidance for all developers to do so. Of course what psychological acceptability of security is supposed to mean is more addressing the use of the system rather than the development. For that, maybe also more indirectly, one interesting line of research would be to make use of work that has been extending UML to perform performance evaluation. This way I could measure the acceptability of security in mechanisms with respect, for example, to the performance impact.

After these design principles something else I would like to point out, even though it is at this point still a work in progress, is addressing more closely this problem of considering security protocols in contexts. For example, also in this setting where I considered the common electronic purse specifications, this situation came up that a part of the security of the system relies on the fact that the communication between, for example, the card and the load application module, cannot be simply relayed by an attacker which would be the case of course in the Internet. Although in the first instance it is not intended to use this on the Internet, but it is already pointed out in various parts of this specification that one intends to do so in the future, so it is important then to keep in mind that one would actually have to change the protocol in order to do this and it may not be clear whether people are aware of that or at least it should be made clear in the specification and this is not done at this point. So, another example why it is important to consider the contexts of the protocol and this is, as I said, also what I'm trying to address with this approach.

And so the conclusion. Following what Saltzer and Schroeder stated, I'm trying to work towards an expansive view of computer security in the design, development and verification of systems, and I'm trying to work towards a general method for constructing secure systems. Specifically what I showed in this talk was these design rules by Saltzer and Schroeder and how one can evaluate system design with respect to them, and also I pointed out how to move towards considering protocols in contexts.

Tuomas Aura: Do you have any arguments that this list of design principles covers everything in security?

Reply: At this point, not yet. This is still ongoing research, I mean there is still a lot to do and this is at a rather early point of this, setting up this whole approach. The motivation why I considered these design principles is simply that they are classic and recognised design principles in computer security. Of course it would be very interesting to go further and show to what degree these principles that were proposed really give me security. But of course since these are rather general design principles it may be difficult to quantify this. Also they pointed out in the paper these are simply principles, so you don't have to follow them in every respect and may still get a secure system, on the other hand if you simply blindly apply them then there's no guarantee that your system will

be secure in the end. So this is giving you some useful progress in the design and implementation of systems, but no perfect security, I mean as elsewhere you don't have anyway. But, yes, it would be very interesting using this approach also to evaluate the scope of these design principles.

David Wheeler: How far do your principles go in recording intrusion or failures so that you can trap or otherwise detect failures?

Reply: In these design principles by Saltzer and Schroeder, I guess the closest one of that that would apply is this one of failsafe default.

David Wheeler: Ah yes, but it's beyond failsafe. Failsafe merely says you can recover whereas what I'm asking is, is there any provision for detecting and thence trapping (or not) security breaks.

Reply: Yes, that would be also an interesting further point. This was a further point they also mentioned in their paper, but they don't really make this a part of their list of principles because, at least in their way of argument maybe, this is a principle which in the physical world may be easy to do. The example they give is of a locker which itself is not very secure but with a strong lock, and the point here is that, people may get to the content of this locker by simply breaking it open but then you will see that this was being done because it is not possible to simply open the lock without having the key, yes. At least they argue that in the physical world, this works, straightforward in this example, but it may be more difficult to see how exactly this example would work in the world of computer security. It would be very interesting to see how one can formulate general principles to do this.

Virgil Gligor: Some of these principles apply quite directly to access control mechanisms and policies, but I was wondering how you'd apply them to other security aspects such as authentication. For example, how would you apply the least privilege principle to authentication, have you given any thought to that?

Reply: In these design rules, I guess their first goal is to apply to a system as a rule. What I am trying to do here is to look at how security protocols are being used in the system, as I mentioned. Now since you mentioned authentication, this presumably would more be addressed by design principles for security protocols, rather than the complete system[3], of which there have been some proposed, as you are probably aware. The first paper, I think, was by Abadi and Needham followed by Anderson and Needham in the mid 90s and this would be also a very interesting further line of research in trying to formalise these design principles. But in the first instance, I looked at these more general design principles for systems rather than protocols because the method I'm using here has the possibility to address, at least in principle, the whole system rather than just a small component of the security protocol. It interesting further research to look at these principles for security protocols, but I've not really done that yet.

[3] Although for arguments against this view, see R.M.Needham, "Security Protocols and the Swiss Army Knife", LNCS 2133, pp 1–4.

A Multi-OS Approach to Trusted Computer Systems

Hiroshi Yoshiura, Kunihiko Miyazaki, Shinji Itoh, Kazuo Takaragi, and Ryoichi Sasaki

Hitachi, Ltd., Systems Development Laboratory,
292, Yoshida-cho, Totsuka-ku, Yokohama, 244-0817, Japan
{last name}@sdl.hitachi.co.jp

Abstract. Trusted computer systems are becoming increasingly important in private sectors such as banking and insurance, but those currently available are inconvenient in private sectors because they were developed for use by government agencies and the military. This paper describes how a trusted system can be made from an ordinary OS for daily use and a secure OS monitoring that OS. This configuration is implemented by using the multi-OS control technique. Analysis of an example system confirms the feasibility of the proposed method and even implies that a trusted system can be implemented by having two ordinary OSs that monitor each other.

1 Introduction

Information handled by computers is not secure unless the platform systems of those computers, as well as the cryptographic algorithms and application software running on those computers, are secure. Consequently, trusted computer systems have been studied since the early 1970s and have been used in security-sensitive fields, such as those dealt with by government agencies and the military [1].

Recently, however, private sectors such as banking and insurance and digital content delivery are also becoming security sensitive because various kinds of valuable data have been computerized and used in electronic commerce. This causes the following problems concerning the use of trusted computer systems in private sectors:

(1) Trusted computer systems are inconvenient for daily use because they are specialized and most of the application programs developed for ordinary computer systems cannot be used on them.
(2) Except for the security technology, the technology used in trusted computer systems is not necessarily the latest because the period of their model change is longer than that of ordinary computer systems.
(3) Trusted computer systems are often much more expensive than ordinary systems.

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 107–114, 2002.

Table 1. Security Classes defined in the Orange Book.

Class	Additional conditions
D	-
C1	(1) User can classify files into three types: accessible by only this user, accessible by members of the same group, and accessible by anyone. (2) System authenticates each user when that user logs in.
C2	(1) User can specify the accessibility of each file in terms of individual persons (e.g., John can read File 1 and Mary can write onto File 2). (2) System can monitor each user. (3) Data remains in memory and disks are not accidentally accessed.
B1	(1) A security label must be given to each file and user, and a user's access to a file is controlled according to those labels. Defining security labels is allowed to the system administrator and not to the users. This function is called "mandatory access control." (2) The system needs to be divided into security-control parts and other parts.
B2	(1) Mandatory access control must be applied to all the files and users as well as to all the peripheral equipment. (2) Interactions between system and user need to be certified by physical means.
B3	(1) The access control monitor needs to be protected from modification and circumvention.
A1	(1) Security functions need to be guaranteed by formal analyses and mathematical proof.

This paper proposes a multi-OS approach to solving these problems. This approach typically uses two operation systems. One, for daily use, is convenient and inexpensive but by itself is insecure. The other OS is for security and is isolated and used only by the system administrator. It is secure by itself and ensures the security of the first OS by monitoring it.

2 Trusted Computer Systems

Trusted computer systems are defined in several books, and the standard since 1983 has been the Orange Book [2]. For network-oriented systems, the Red Book complements the Orange Book [3]. Although the Orange Book is now being superseded by ISO/IEC 15408 [4], this paper uses the definitions in the Orange Book because they are well known and is easily understood.

The Orange Book defines seven classes of security (Table 1), and the conditions for a more secure class include the conditions for all the classes that are less secure. For the least secure (Class D) there are no security conditions, and right-hand side of the table lists only the additional conditions for each class. Note that these definitions mainly concern operating systems.

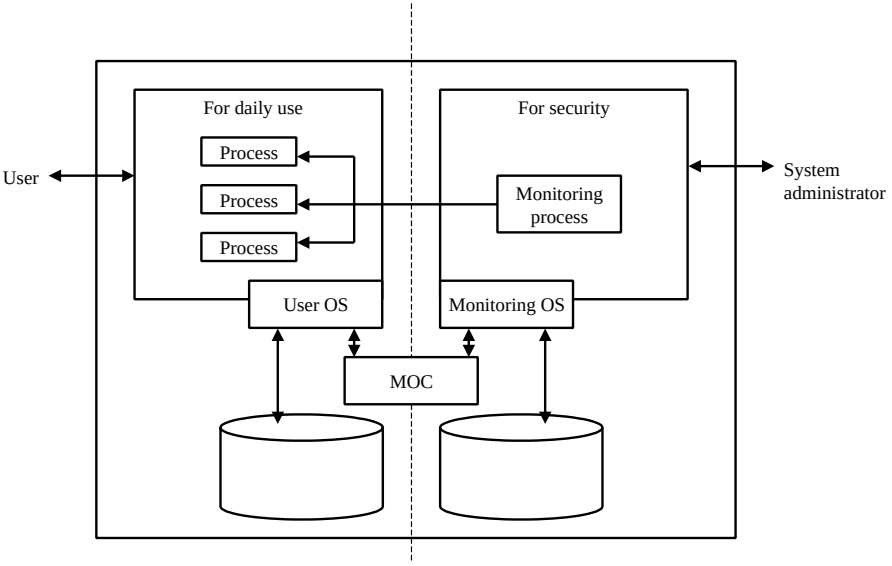


Fig. 1. Basic ideas.

3 Multi-OS Approach to Trusted Computer Systems

3.1 Multi-OS Controller [5]

Multi-OS control is a recently established technology using a multi-Os controller (MOC) in a layer below that of OSs. It enables multiple OSs to be used effectively in a single computer, and the typical functions of a MOC in a two-OS computer system are the following:

(1) In the default state the first OS is activated and the second OS is asleep.
(2) Conditions for the MOC to change OSs can be defined.
(3) According to the conditions and the system status, the MOC makes the first OS sleep and activates the second OS.
(4) The MOC also restores the default state according to the conditions and the system status.
(5) Processes and files are managed by either the first or second OS but not by both.
(6) The MOC can impose one-way accessibility on the two OSs (i.e., processes managed by one OS can access processes and files managed by the other OS, but the reverse is not allowed).

3.2 Goals and Ideas

Our goals are to solve the three problems mentioned in Section 1: trusted systems are inconvenient for daily use, their technologies other than security technologies are not necessarily latest, and they are expensive.

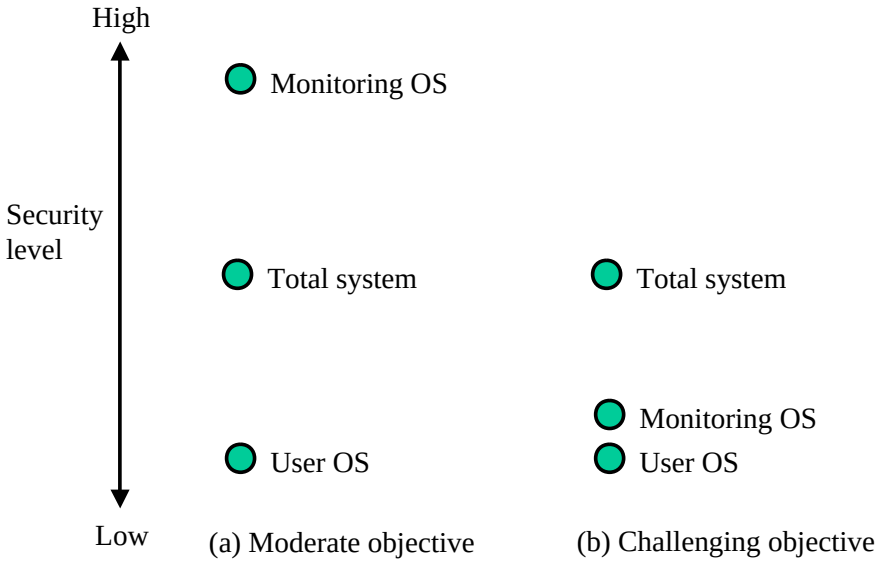


Fig. 2. Objectives of the proposed method.

Fig. 1 illustrates our basic ideas for solving these problems. The proposed method uses two OSs based on the MOC. The user's daily application programs run on the user OS, and the monitoring process assuring the security of the user OS runs on the monitoring OS. We have two objectives:

(1) to make the total system more secure than the user OS (Fig. 2 (a)).
(2) to make the total system more secure than any of the user and monitoring OSs (Fig.2 (b)).

The first objective is moderate: a trusted system that is not inconvenient and that can use the latest technologies. This is because the user OS can be an ordinary inexpensive insecure OS on which daily application programs run or it can be one that is state-of-the art with regard to aspects other than security. Such a system is not inexpensive, though, because it requires the use of two OSs and the monitoring OS may need to be an expensive trusted OS.

The second objective is more challenging: a state-of-the-art convenient system that is inexpensive because both the user and monitoring OSs can be inexpensive OSs less secure than the total system.

4 Example

4.1 Question to Be Addressed

This section describes an example of a system meeting the first objective. We assume the user OS to be of class C2. This assumption is appropriate because

ordinary OSs have a class C2 or lower. The monitoring OS is assumed to be more secure (for example, of class A1), and our aim is to make the user OS more secure than C2.

Most significant difference between class C2 and the higher classes is mandatory access control (MAC), the definition (1) of class B1 in Table 1: C2 does not have MAC, but B1 does. This section therefore focuses on the following question: "Can the user OS of class C2 have MAC by using the proposed method?"

4.2 Mandatory Access Control

Mandatory access control is defined in the Orange Book as follows (note that the following definitions are simplified for clarity):

(1) Security levels are defined. For example, the four levels Unclassified, Classified, Secret, and Top Secret.
(2) A security level is given to each file and user.
(3) A user can read a file if and only if the user's security level is equal to or higher than the file's security level.
(4) A user can write to a file if and only if the user's security level is equal to or lower than the file's security level.
(5) These security levels can be given only by the system administrator (i.e., not by the users).

4.3 Example System

(1) Structure

Fig. 3 illustrates the structure of the example system using the proposed method. In the user OS, which is loaded into the main memory, file accesses are executed by the file management process. The monitoring OS is also loaded into the main memory. Security levels of the files and users are defined by the system administrator and recorded in the Security level file managed by the monitoring OS. Access is controlled by an access control program, also loaded into the main memory.

The MOC is loaded and imposes one-way accessibility on the two OSs. That is, processes managed by the monitoring OS can access processes and files managed by the user OS, but the reverse is not allowed.

The access control program monitors all the file access commands processed by the file management process. It checks each file access command and suspends it if the access does not correspond to the Security level file.

(2) Monitoring

Because the two OSs cannot be active at once, it is impossible for the monitoring process to directly monitor the file management process. Thus, in the implementation, the MOC participates in the monitoring. The MOC recognizes the file access command and hooks it from the file management process. The MOC then stops the user OS, activates the monitoring OS, and passes the file

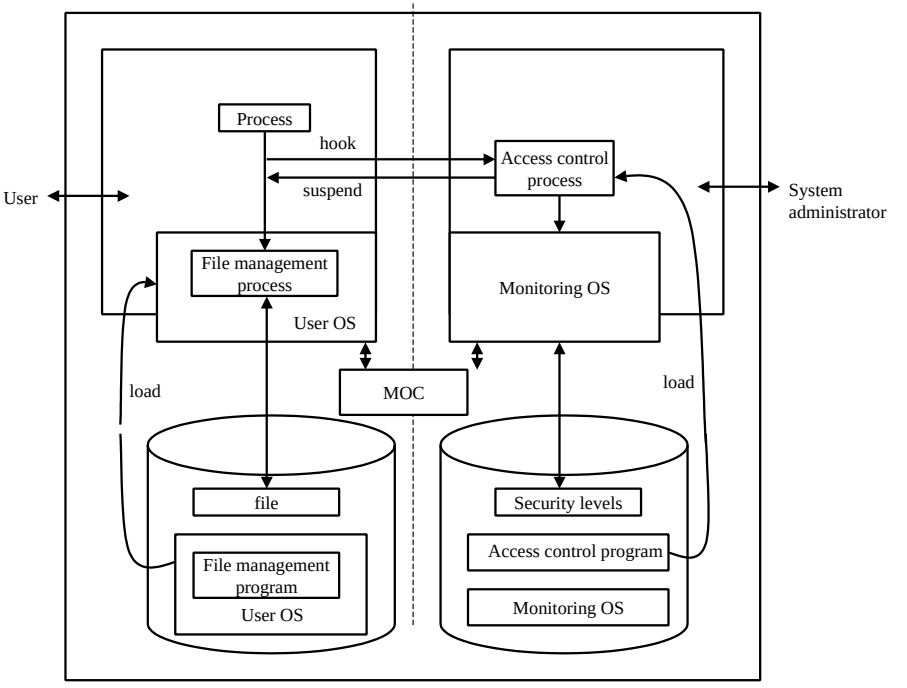


Fig. 3. Example system.

access command to the access control process. The access control process checks the file access command, determines the accessibility and informs the MOC of the decision. Finally, the MOC suspends the file access if the decision is "not accessible."

5 Security Considerations and Questions

5.1 Security against Online Attacks

Once all the programs and OSs and the MOC have been correctly loaded into the main memory, the consideration of MAC security can be reduced to consideration of the following four factors:

(1) Security against attacks to the user OS

Can file access commands executed in the user OS be made to circumvent the monitoring function of the MOC? We are currently analyzing this possibility in co-operation with OS specialists and expected to find that "It is impossible to circumvent the MOC's monitoring because all the file accesses are eventually executed by the file management process."

(2) Security against attacks to the monitoring OS

The access control process might be attacked so that it cannot work correctly. The security against these attacks is simply that of the monitoring OS. In this example, the monitoring OS is assumed to be sufficiently secure and we do not need to be anxious about this problem.

(3) Security against attacks to the MOC

The MOC might be attacked so that it cannot recognize or monitor the file access commands or activate the monitoring OS. The security against these attacks is simply that of the MOC and we are currently analyzing it in cooperation with OS specialists.

5.2 Security against Offline Attacks

The programs and OSs and the MOC need to be loaded correctly before they can work properly. The following problems therefore need to be considered:

(1) Security against attacks to the user OS

The user OS, especially the file management program, might be modified so that the file management process cannot be recognized by the MOC. This attack can be treated by using a tamper detection program. That is, a tamper detection program can be used on the monitoring OS to check the user OS by using digital signatures.

(2) Security against attacks to the monitoring OS

The monitoring OS, the access control program, and security level file might be modified so that the monitoring cannot be executed correctly. The tamper detection program might be also attacked so that the tampering to the user OS cannot be detected. The security against these attacks is that of the monitoring OS and here we do not need to be anxious about this problem.

(3) Security against attacks to the MOC

The MOC might be modified. The loading path of the MOC might be attacked so that a fake MOC is loaded. We are also analyzing this problem in cooperation with OS specialists.

5.3 Open Questions

As noted above, the main open question is about the security of the MOC:

(1) How secure is the MOC against on-line and off-line attacks?

A related question is about the effectiveness of the MOC with regard to non-security aspects:

(2) Can the MOC be generally used with any OSs or does it need to be customized for each OS? Can the MOC be inexpensive?

Two further questions are the following:

(3) Can the total system be made more secure than the user and monitoring OSs?

(4) Can the method be improved by using two mutually monitoring OSs or by using more than two OSs?

6 Conclusions

Trusted computer systems are becoming increasingly important in private sectors, and those currently available are inconvenient for use in the private sectors because they were developed for use by government agencies and the military.

This paper showed how a trusted system can be made by using an ordinary OS for daily use and a trusted OS for monitoring the ordinary one. The proposed method was modeled by using the multi-OS controller, and its feasibility was shown by the analysis of an example system. There are still open questions with regard to the security and effectiveness of the multi-OS controller and with regard to the possibility of making a trusted system from two ordinary OSs that monitor each other.

References

1. Deborah Russell, G.T. Gangemi Sr.: Computer Security Basics, O'Reilly & Associates, Inc., 1991.
2. Department of Defense Trusted Computer System Evaluation Criteria, Dept. of Defense, National Computer Security Center, DOD 5200.28-STD, 1985.
3. Trusted Network Interpretation of the TCSEC, National Computer Security Center, 1987.
4. ISO/IEC 15408 Information Technology – Security Technology – Evaluation Criteria for IT security, 1999.
5. T. Arai, T. Sekiguchi, M. Satoh, T. Inoue, T. Nakamura, H. Iwao: DARMA: Using Different OSs Concurrently based on Nano-Kernel Technology, Proc. 59th-Annual Convention of Information Processing Society of Japan, 1999.
6. Evaluated Product List, http://www.radium.ncsc.mil/tpep/epl/historical.html

A Multi-OS Approach to Trusted Computer Systems (Transcript of Discussion)

Hiroshi Yoshiura

Systems Development Laboratory, Hitachi Ltd.

I would like to talk about an idea, not results. Firstly I clarify the problems with previous trusted computer systems. Information security requires security of application system and security of platform, including hardware and operating system. Trusted computer systems or TCSs have been studied and used by the military since the early 1970s, but the requirement for TCSs increases because of computerisation in government, growth of economic commerce and passing of electronic signature laws. TCSs are now needed not only for the military but also for public and private sector, as in banking, insurance and digital content delivery. Public and private sector TCSs need not only to be secure but also to be easy to use for non-professional users, as they have to be used for security critical tasks as well as daily tasks. They also have to be cost effective. In meeting these new requirements, TCSs have two problems: they are hard to use and inconvenient for daily tasks, and they are much more expensive than ordinary systems. So I will propose a multi-OS approach to trusted computer systems.

Before proposing the solution I would like to explain an existing technology called multi-OS controller, or MOC. MOC is a nano-kernel, that is a system running between hardware and OSs. By using MOC, multiple OSs run independently on a single computer; each OS on a MOC cannot recognise other OSs running on the same computer and MOC partitions the computer resources and dedicates them to each OS. Communication between two OSs is only possible via MOC and each OS accesses dedicated devices.

MOC can be used to give accesses between two OSs in one direction, that is, a process on the secondary OS can access processes and resources of the primary OS, but the reverse is not allowed. Our basic proposal is to build a trusted computer system using MOC. We use two OSs on a single computer, and the computer memory is time divided into two virtual spaces. The user space is controlled by the user OS, and it is for daily use of the ordinary users; ordinary users recognise only this space. Administrator space is controlled by the administrator OS and it is for security and it is used only by security administrator; users do not recognise this space. Security processes, such as monitoring processes, are instituted in the administrator space to monitor processes in the user space and cryptographic process may be executed in the administrator space to protect the secret key. The essential advantage of using MOC which I expect, is that the administrator's space can be tamper resistant.

Using the MOC, security systems such as the monitoring process in the administrator space can use the large CPU and memory of user's computer and

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 115–118, 2002.

can provide robust and versatile computer security support to the total system. With the alternative method, using only ordinary OS, security systems can also use the large CPU and memory, but these security systems are easily attacked because they are running on an ordinary OS, so they can only provide limited security. Another alternative is to use small hardware such as smartcard. Security system in small hardware can only use the small CPU and memory of the hardware, so it can only provide limited security to the total system. So I optimistically believe that the proposed method is best.

There are two other reasons for using MOC. Our moderate objective is that the total system is more secure than the user OS, although it is less secure than the administrator OS. Previous TCSs are inconvenient, because they are specialised, and they are expensive. If our moderate objective can be achieved we can use an ordinary convenient OS for the user OS, and because user can only recognise the user OS, we just solved the first problem of TCSs being inconvenient for daily use. The more challenging objective is that the total system is more secure than both the administrator and the user OS. If this objective can be achieved, we can use ordinary convenient and inexpensive OSs for both the administrator and the user OS, and we have solved the first problem as well as the second problem of being expensive. I will show an example system and an analysis of its security. In the example problem, we assume that the administrator OS is sufficiently secure (at least we're not deeply anxious about it), but the user OS does not have a function for file access control. Our goal is to control users file access: to give a function for file access control to the user OS, and to protect this access control securely from attacks.

Here is a rough sketch of a system. An access control process is executed in administrator space to monitor and control the access from the user process to the user file. The access control process refers to the access control policy, to know how to control the access. We assume all user files are encrypted beforehand. An access from the user process goes through the file management process and (if the user OS has not been tampered with) that goes through a special part, so the access can be recognised by access control process and allowed or disallowed. If allowed, the encrypted user file is read out and decrypted and passed to the user process. This access control cannot be bypassed because decryption is necessary. If it is bypassed the user process cannot read the user files, so access control cannot be bypassed. And these processes of access control work correctly and this secret key cannot be stolen because the administrator space is tamper resistant. Now, we assumed that all user files are encrypted so we have remaining problem of how to read the files, we also assumed that the user also has not been tampered with, so another problem is how to detect the tampering.

We encrypt user files in initial installation of system that easily encrypts user files using a secret key, and store this secret key into another file which is controlled by the administrator OS. This means that this file is securely protected from attacks because we assume that the administrator OS is sufficiently secure and the stored secret key can be loaded when access control is executed.

We detect tampering in the system booting phase. The first step of system booting is to load security programs into the administrator space to establish the access control process and tampering protection process, then the tampering protection process checks the user OS to guarantee that it has not been tampered with. Multi-OS controller divides a computer system into two virtual systems, the user system controls the users and the administrator system here, behind the user system, provides a tamper resistant module upon which security processes run to keep the total system secure. This method makes the computer system both secure and convenient because the system can be built from a secure administrator system and convenient user system.

A future work is to build a trusted computer system from two insecure and inexpensive systems by for example two systems monitoring each other and by using minimal hardware. We also would like to use this method to prevent illegal copying of digital content by allowing a plain text of digital content to appear only in the administrator space.

Thank you for your attention, any questions.

Stephen Early: I think you've set yourself an extremely difficult task, and when you come to implement it you'll find a lot of problems. The biggest problem I can see is how you can tell what is going on in your standard operating system. You have a plausible story about spotting file accesses because if they don't go through the routes that you're expecting, then you just won't decrypt the files, that's fair enough. But how can you spot things like one process accessing the file on behalf of another process, possibly by a different user? I don't think it's possible for your monitoring program to understand everything that is going on in the monitored operating system. Assume that you don't have source code for the user operating system, it's just a commercial operating system which you've bought from somebody like Microsoft. How can your monitoring process understand what is going on in it, how can it spot all of the flows of data, for example? How can it know what user is logged into it, how can it know on whose behalf each process is running, and how can it tell what is one process and what is another process? Say your user operating system is running two processes, one owned by each user, how do you prevent one process from getting the other process to act on its behalf to access a file? A multi-level secure operating system would prevent this; this seems to be your goal, but I don't see how you could do it.

Reply: I did not explain the concrete method of activating this OS. In the ordinary situation only the user OS is active and the administrator OS is inactive because the two OSs are executed in time divided manner. We insert code into the file management process which adds an activating program. The activating program recognises the access and this program reports it to MOC and MOC activates the administrator OS and passed to the access control program the details of this access and then the result is done. But this is implementation details — the essential problem is that we can automatically catch all the accesses, this is a past problem, and access control can be solved if the OS has not been tampered with.

Stephen Early: I understand what you're doing with respect to file access but what about other flows of information? For example a program could read a file, as it is allowed to, and then spit it out a network port; how can you stop that kind of thing? Essentially how can you spot how the data is flowing within the user operating system and enforce your policy on that? If I want to leak information from a file which I have access to but I'm not supposed to be able to send elsewhere, how can you prevent me from doing that?

Michael Roe: Usually in architectures like that, the user OS is taken as completely untrusted. As far as the administrators rôle is concerned, the user OS is one big program and you don't care what flows are in it, all you care is that it only gets data that that OS as a whole is allowed to have access to.

Stephen Early: If that is the case then that's fair enough.

Stewart Lee: I think that's the intention.

Markus Kuhn: So once you have accessed top secret data you have to press control-alt-delete in order to publish unclassified data; you have to reboot regularly in order to get to a lower level with your entire operating system.

Stephen Early: Could you tell us what risk you are trying to protect against with this system. What's the goal?

Reply: We have one more barrier to attack because the attacker must intrude into this user OS and also he must get through the administrator OS. In the user OS the attacker can do everything. But intruding into the administrator OS is much more difficult because we shall use a very secure OS, and it is not open to ordinary users. Network systems such as the Internet are connected only to the user OS, so it is very difficult to have the privilege of the administrator OS.

Geraint Price: The thing with the goal is to make sure that the access control that goes in the file system could be rigidly enforced outside of the operating system of the users.

Stephen Early: But how can you have access control policy when you can't tell what the source of a request from user space is?

Geraint Price: The user operating system *is* the user space.

Roger Needham: The threat you're protecting against is that the user process *and* the operating system is all the enemy.

A Proof of Non-repudiation

Giampaolo Bella[1,2] and Lawrence C. Paulson[1]

[1] Computer Laboratory, University of Cambridge
Pembroke Street, Cambridge CB2 3QG (UK)
{gb221,lcp}@cl.cam.ac.uk

[2] Dipartimento di Matematica e Informatica, Università di Catania
Viale A. Doria 6, I-95125 Catania (ITALY)
giamp@dmi.unict.it

Abstract. The Inductive Approach for protocol verification, extended with a formalisation of message reception and agents' knowledge, is adopted here for reasoning about non-repudiation protocols. The fair non-repudiation protocol due to Zhou and Gollmann is modelled inductively in Isabelle/HOL. The verification focuses on the *validity of evidence* goal: that the evidence held by each peer at the end of a session suffices to refute the other's denial of participation. The proof strategies appear to be general thus far, but the *fairness* goal is yet to be studied.

1 Overview

Non-repudiation protocols aim at protecting the peers from each other's walking away from a transaction. *Non-repudiation of receipt* (NRR) signifies that the responder who receives a message can refute the initiator's denial of having sent the message. *Non-repudiation of origin* (NRO) allows the initiator who sends a message to refute the responder's denial of having received it. Each refutation is typically stipulated by a judge upon the peer's exhibition of specific pieces of evidence — i. e. cryptographic messages, specifically, digital signatures — which must be valid. Hence, *validity of evidence* is the primary goal of a non-repudiation protocol.

A number of non-repudiation protocols have been designed, but they are not yet deployed [5,6]; verifying them formally [10,11] might strengthen their credibility. Proving non-repudiation goals was in fact one of the the first author's purposes for extending the Inductive Approach with message reception [1] and agents' knowledge [2]. This paper shows that we have now achieved that purpose to some extent. We assume a basic familiarity with protocol verification using the Inductive Approach [9], which is mechanised in Isabelle/HOL [7].

Non-repudiation protocols can easily be modelled in an inductive definition. Each rule of the definition describes an event that may happen, but no event is forced to happen. When a peer is entitled to send a message, the model does not force him to: he is free to act unfairly. Similarly, a message that is sent will not necessarily be received. This captures the unreliability of the communications.

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 119–125, 2002.

In our model, messages cannot alter during transmission. We follow previous work by not including an external attacker.

To prove validity of evidence we need to establish that, if a peer holds some message M, then certain events concerning the other peer occurred. Our inductive protocol model is the set of possible traces of events, so an inductive proof of validity of evidence considers every way in which a trace could have been generated that contains the message M, and checks that M appeared after the claimed events in every case. A successful proof means that if a peer can exhibit the message, then the claimed events did occur. A failed proof indicates that the message alone forms insufficient evidence. We have developed a simple strategy to proving validity of evidence, based on two intermediate results.

First, we prove that if an agent B knows a message (of those forming the evidence), then an event ev_B whereby B learnt that message occurred — according to the current formalisation of events, ev_B is either "B sends a message to someone", or "B receives a message", or "B notes down a message" . This result belongs to a new hierarchy that involves reasoning on the knowledge of legitimate agents. Clearly, an agent *only* knows what she sends or receives during a protocol [2], and this produces longer case splits than reasoning on the attacker's knowledge does, because the latter includes *everything* that is sent or received by any-one. Previous work on authentication protocols [4] only required reasoning on the attacker's knowledge.

Second, we need to establish that, if ev_B occurs, then a prior event ev_A concerning B's peer A also occurred. Theorems of this form can be proved routinely, as they resemble authentication theorems [3]. Notice that the entire strategy must be applied for each of the messages forming the evidence that is needed to B to prove A's participation.

This paper presents the Zhou-Gollmann protocol as a case study (§2), its modelling (§3), and its verification (§4). Finally, it draws some conclusions (§5).

2 A Case Study: The Zhou-Gollmann Protocol

We demonstrate our strategy to proving validity of evidence on the Zhou-Gollmann protocol. Its *fair* variant [12] is given in Figure 1, though we have not yet studied the fairness goal.

Here, TTP is a trusted third party, M is the message that A intends to communicate to B, K is the key that A chooses to transmit M, C is M encrypted with K, and L is a unique label. Also, $f_\star$ are the non-repudiation flags, and sK_X is the private signature key of agent X (the corresponding public verification key being known to all), and $\{\!|m|\!\}_{sK_X}$ indicates the signature of message m with key sK_X.

The last two steps are interchangeable, as they stand for an *ftp get* operation from a publicly accessible directory of TTP. The protocol authors comment extensively on the protocol functioning [12]. For example, they observe that, even if the peers do not want to play fair, they must complete a session in order to get sufficient evidence to win any disputes with each other. For this purpose,

$$
\begin{aligned}
&1.\ A \to B \quad : \quad \mathsf{f}_{\mathsf{nro}}, B, L, C, \underbrace{\{\!|\mathsf{f}_{\mathsf{nro}}, B, L, C|\!\}_{sK_A}}_{NRO}\\
&2.\ B \to A \quad : \quad \mathsf{f}_{\mathsf{nrr}}, A, L, C, \underbrace{\{\!|\mathsf{f}_{\mathsf{nrr}}, A, L, C|\!\}_{sK_B}}_{NRR}\\
&3.\ A \to \mathsf{TTP} : \quad \mathsf{f}_{\mathsf{sub}}, B, L, K, \underbrace{\{\!|\mathsf{f}_{\mathsf{sub}}, B, L, K|\!\}_{sK_A}}_{sub\_K}\\
&4.\ B \leftrightarrow \mathsf{TTP} : \quad \mathsf{f}_{\mathsf{con}}, A, B, L, K, \underbrace{\{\!|\mathsf{f}_{\mathsf{con}}, A, B, L, K|\!\}_{sK_{\mathsf{TTP}}}}_{con\_K}\\
&5.\ A \leftrightarrow \mathsf{TTP} : \quad \mathsf{f}_{\mathsf{con}}, A, B, L, K, \underbrace{\{\!|\mathsf{f}_{\mathsf{con}}, A, B, L, K|\!\}_{sK_{\mathsf{TTP}}}}_{con\_K}
\end{aligned}
$$

Fig. 1. The Zhou-Gollmann Protocol

rather than the plaintext message M, A initially sends the commitment C, which B can decrypt only when he terminates the session.

Let us informally analyse how to resolve disputes. From B's standpoint, it appears that obtaining *con_K* signifies that A submitted K, bound to label L, to TTP; obtaining *NRO* should signify that A sent C as a commitment bound to label L. In consequence, message M, obtained decrypting C with K, should have originated with A. From A's standpoint, a similar reasoning seems feasible. If A holds *con_K*, this should guarantee that A lodged K and L with TTP, and so B would be able to get it via *ftp*. If A also holds *NRR*, it should be the case that B accepted commitment C. In consequence, B would be able to obtain M.

Our reasoning might resemble that of a judge who is provided with evidence *NRO*, *con_K*, M, C, K, L by agent B, or with a similar evidence (but *NRR* rather than *NRO*) by agent A. The question is whether such a reasoning is correct in a setting where the communication means is unreliable, and the protocol is executed by an unlimited number of agents, each entitled to initiate an unlimited number of interleaved sessions, but not forced to terminate them. In the following, we formally prove the answer to be YES.

3 Modelling the Case Study

Our work rests on the Isabelle theory Public [8] for cryptographic protocols based on asymmetric encryption. The theory models three kinds of network agents: a spy, which is not used here, a trusted server, which is renamed as TTP here, and an unlimited population of legitimate agents. Each agent X is endowed with a key pair priK X, which is only known to X, and pubK X, which is known to all. We will use them respectively as private signature key of X, and public verification key of X. The theory also provides a primitive for encryption, Crypt, which we use where the protocol requires a digital signature.

The protocol is modelled routinely until A sends her message to TTP. The next steps are modelled as shown in Figure 2 and are more interesting.

TTP_prepare_ftp

```
[| evsT ∈ zg;
   Gets TTP {|Number f_sub, Agent B, Nonce L, Key K, sub_K|} ∈ set evsT;
   sub_K = Crypt (priK A) {|Number f_sub, Agent B, Nonce L, Key K|};
   con_K = Crypt (priK TTP) {|Number f_con, Agent A, Agent B,
                              Nonce L, Key K|} |]
⟹ Notes TTP {|Number f_con, Agent A, Agent B, Nonce L, Key K, con_K|}
      # evsT ∈ zg
```

A_ftp

```
[| evsA ∈ zg;
   Notes TTP {|Number f_con, Agent A, Agent B, Nonce L, Key K, con_K|}
     ∈ set evsA |]
⟹ Notes A {|Number f_con, Agent A, Agent B, Nonce L, Key K, con_K|}
      # evsA ∈ zg
```

B_ftp

```
[| evsB ∈ zg;
   Notes TTP {|Number f_con, Agent A, Agent B, Nonce L, Key K, con_K|}
     ∈ set evsB |]
⟹ Notes B {|Number f_con, Agent A, Agent B, Nonce L, Key K, con_K|}
      # evsB ∈ zg
```

Reception

```
[| evsr ∈ zg; Says A B X ∈ set evsr |] ⟹ Gets B X # evsr ∈ zg
```

Fig. 2. Modelling the Zhou-Gollmann protocol (fragment)

Rule TTP_prepare_ftp models TTP's creation of the key confirmation in its publicly accessible directory. Note that TTP verifies the signature *sub_K* to learn the identities of the peers for K. All components necessary to verify that signature are available. The issuing of *con_K* is modelled by a Notes event [9], which indicates a change to TTP's internal state.

Rules A_ftp and B_ftp model the peers' retrieval of *con_K*. The two rules are not forced to fire simultaneously, since each peer decides independently whether to terminate the protocol. Rather than introducing a new event to express the *ftp get* operation, we again adopt Notes. Using a Gets event instead would violate the conventions of the message reception model, in which each Gets event must follow a matching Says event, as stated by rule Reception.

4 Verifying the Case Study

First, let us verify that, if B holds *NRO*, *con_K*, M, C, K, and L, then A cannot deny having sent M.

According to the general strategy given above, we prove that the only way for B to obtain *con_K* is to get it via *ftp*, namely to complete the protocol. Theorem 1 states this formally. Recall that the function knows yields the set of messages that a given agent learns from a given trace, namely the messages that the agent sends or receives or notes on the trace. The operator parts extracts all components from a set of messages by recursively breaking down concatenated messages and ciphertexts (preserving the encrypting keys, of course). So, the assumption *con_K* $\in$ parts(knows B *evs*) signifies that *con_K* is potentially accessible to B during the network history indicated by *evs*. In practice, B can verify the contents of *con_K* as he knows TTP's public verification key.

Theorem 1.

```
[| evs ∈ zg; con_K ∈ parts (knows B evs);
   con_K = Crypt (priK TTP) {|Number f_con, Agent A, Agent B,
                              Nonce L, Key K|} |]
⟹ Notes B {|Number f_con, Agent A, Agent B, Nonce L, Key K, con_K|}
     ∈ set evs
```

The proof is non-trivial in the Reception case, where Isabelle's simplifier leaves us with the possibility that B knows *con_K* because he has received it from the network (rather than noted it). In this sub-case, someone must have sent it by a Says event, and we appeal to a lemma stating that no agent can ever send *con_K* (because TTP's private signature key is safe).

Again following the general strategy, we can routinely prove Theorem 2, stating that if B has noted *con_K* then A indeed lodged K with TTP, bound to the label L.

Theorem 2.

```
[| evs ∈ zg;
   Notes B {|Number f_con, Agent A, Agent B, Nonce L, Key K, con_K|}
     ∈ set evs;
   con_K = Crypt (priK TTP) {|Number f_con, Agent A, Agent B,
                              Nonce L, Key K|};
   sub_K = Crypt (priK A) {|Number f_sub, Agent B, Nonce L, Key K|} |]
⟹ Says A TTP {|Number f_sub, Agent B, Nonce L, Key K, sub_K|} ∈ set evs
```

The proof initially deduces that TTP made *con_K* available, that is the event

$$\mathsf{Notes}\ \mathsf{TTP}\{\!|\mathsf{Number}\ \mathsf{f_{con}}, \mathsf{Agent}\ A, \mathsf{Agent}\ B, \mathsf{Nonce}\ L, \mathsf{Key}\ K, con\_K|\!\}$$

occurred on the trace, and then concludes that A sent *sub_K* on the same trace. Theorems 1 and 2 together state that, if B can access *con_K*, then A sent *sub_K*.

Some extra evidence is needed to B to refute a denial from A. The evidence is *NRO*, which B holds only if A sent it to him, as Theorem 3 states.

Theorem 3.

```
[| evs ∈ zg; NRO ∈ parts (knows B evs);
   NRO = Crypt (priK A) {|Number f_nro, Agent B, Nonce L, C|} |]
⟹ Says A B {|Number f_nro, Agent B, Nonce L, C, NRO|} ∈ set evs
```

The proof of Theorem 3 resembles that of Theorem 1. A lemma is formulated for the induction, whose premise refers to the parts operator. The statement shown above is a trivial corollary of the inductive lemma. The induction requires some care in the Reception case, where the simplifier tries to establish whether B might learn *NRO* by receiving it from the network. We prove it by appealing to another lemma, which confirms that *NRO* is only sent in the first step of the protocol.

These theorems show that a judge may safely conclude from the evidence submitted by B that A intended to transmit to B the message M obtainable decrypting the commitment C with key K. Analogous theorems confirm validity of the evidence submitted by A about B's receipt of M. For example, Theorem 4 establishes that B has received a message ratifying the association between C and the label L in case *NRR* is accessible to A.

Theorem 4.

```
[| evs ∈ zg; NRR ∈ parts (knows A evs);
   NRR = Crypt (priK B) {|Number f_nrr, Agent A, Nonce L, C|};
   NRO = Crypt (priK A) {|Number f_nro, Agent B, Nonce L, C|} |]
⟹ Gets B {|Number f_nro, Agent B, Nonce L, C, NRO|} ∈ set evs
```

5 Conclusions

The Inductive Approach to analysing cryptographic protocols works for non-repudiation protocols once message reception and agent's knowledge are defined. While demonstrating this claim on the Zhou-Gollmann protocol, we have developed general strategies for proving the validity of the non-repudiation evidence.

Zhou and Gollmann analysed their protocol using a logic of belief [11]. Their findings concern validity of evidence and are comparable to ours, although the philosophical differences between the two approaches are well known. Schneider analysed the protocol thoroughly [10] using the theory of Communicating Sequential Processes (CSP). A major difference with our work is that his model includes a judge and the protocol peers presenting him with the non-repudiation evidence. We have chosen not to model the judge because his functioning as well as the peers' interaction with him are external to the non-repudiation protocol. A peer's presenting some evidence to the judge is equivalent to the peer's holding the evidence, which can be concisely formalised in terms of our function knows.

Schneider also studies the fairness of the protocol, confirming that it is achieved. Precisely, he concludes that the initiator never is in a position to win a dispute over the responder, nor is the responder ever in a position to win a dispute over the initiator, once both of them have retrieved *con_K* from TTP. Although we have not yet investigated this goal, we expect to be able to prove

by induction that, if a peer holds some evidence, then the other peer holds the equivalent evidence, and vice versa.

Acknowledgements. Discussions with Dieter Gollmann were very useful. This work was funded by the EPSRC grant GR/R01156/01 *Verifying Electronic Commerce Protocols*.

References

1. G. Bella. Message Reception in the Inductive Approach. Research Report 460, University of Cambridge — Computer Laboratory, 1999.
2. G. Bella. Modelling Agents' Knowledge Inductively. In B. Christianson, B. Crispo, J. A. Malcolm, and R. Michael, editors, *Proc. of the 7th International Workshop on Security Protocols*, LNCS 1796, pages 85–94. Springer-Verlag, 1999.
3. G. Bella and L. C. Paulson. Kerberos Version IV: Inductive Analysis of the Secrecy Goals. In J.-J. Quisquater, Y. Deswarte, C. Meadows, and D. Gollmann, editors, *Proc. of the 5th European Symposium on Research in Computer Security (ESORICS'98)*, LNCS 1485, pages 361–375. Springer-Verlag, 1998.
4. G. Bella and L. C. Paulson. Mechanising BAN Kerberos by the Inductive Method. In A. J. Hu and M. Y. Vardi, editors, *Proc. of the International Conference on Computer-Aided Verification (CAV'98)*, LNCS 1427, pages 416–427. Springer-Verlag, 1998.
5. M. Ben-Or, O. Goldreich, S. Micali, and R. Rivest. A Fair Protocol for Signing Contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.
6. T. Okamoto and K. Ohta. How to Simultaneously Exchange Secrets by General Assumptions. In *Proc. of the 2nd ACM Conference on Computer and Communication Security (CCS'94)*, pages 184–192, 1994.
7. L. C. Paulson. *Isabelle: A Generic Theorem Prover*. LNCS 828. Springer-Verlag, 1994.
8. L. C. Paulson. *Theory for public-key protocols*, 1996. `http://www4.informatik.tu-muenchen.de/~isabelle/library/HOL/Auth/Public.html`.
9. L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6:85–128, 1998.
10. S. Schneider. Formal Analysis of a Non-Repudiation Protocol. In *Proc. of the 11th IEEE Computer Security Foundations Workshop*. IEEE Press, 1998.
11. G. Zhou and D. Gollmann. Towards Verification of Non-Repudiation Protocols. In *Proc. of the 1998 International Refinement Workshop and Formal Methods Pacific*, pages 370–380. Springer-Verlag, 1998.
12. J. Zhou and D. Gollmann. A Fair Non-Repudiation Protocol. In *Proc. of the 15th IEEE Symposium on Security and Privacy*, pages 55–61. IEEE Press, 1996.

A Proof of Non-repudiation (Transcript of Discussion)

Larry Paulson

University of Cambridge Computer Laboratory

As last year, I'm actually talking about work of Giampaolo Bella and my other Italian friends, and I thought I would start by giving an update on last year's talk because there I was presenting my colleagues' work on a SET protocol and I thought a very brief update of what is happening might be useful.

I know everyone says that SET is dead, and maybe it is, but some of the things in SET I'm sure will never be dead. These are things, that as far as I know, haven't been verified in protocols before — ostensibly so-called digital envelopes. Well in SET you have a lot of this — Alice sends to Bob lots of stuff encrypted with some key together with the key itself and maybe some other junk, all encrypted with Bob's public key. The reason they do this is, of course, that public key encryption is very slow and so they don't want to send all this stuff using Bob's public key so they just have a short message that they transmit by the session key and all the rest they've encrypted using the other key.

I'm told this technique is used a lot, although I've never actually seen a protocol that used it before, and it complicates things rather a lot because suddenly now in SET each and every time I do these things I need a different key. In fact in a particularly unpleasant case I remember, one of the things sent in this message encrypted by this key is another key for Bob to use with his reply so that this key is encrypting the other key. Think of this as a stamped addressed envelope or something and then when Bob replies he uses that key and in fact sends other stuff that needs to be secret using this key. Now you see you have some rather tricky chains of dependency where this key here is only secure if the corresponding private key for that one there is secure, and then of course this one depends upon that and this thing here depends upon that and so if any one of these goes, the lot goes.

It took me a while to work out how to cope with this but it turns out that there is a pretty uniform way of doing it which meant we were able to deal with it. It took a while to come to grips with the one in the first phase of SET (which is cardholder registration) but when we'd done that, it only took something like half a day to apply the same technique to the next phase of SET which is merchant registration. This is nice because whatever people come up with next, if they're doing this sort of thing they are used to it now, so it's not too bad.

Right, now if I can go on to the advertised talk — non-repudiation protocols. I'm sorry it's in our usual tradition of not paying too much attention to the theme of the workshop but I suppose non-repudiation is just as important if you're walking around buying things using your WAP-phone as if you were at home. So I guess we all know what non-repudiation is, but just in case you don't, non-repudiation means that you can't run away from a transaction.

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 126–133, 2002.

I suppose all protocols have a degree of non-repudiation in them and I was trying to think what makes us different. The point is that although Bob knows that Alice was there, he might not be able to prove to a third party that Alice was there because maybe it depends either on the third party having been listening in on the whole conversation, which you wouldn't want to do because there's a lot of conversations happening at once, or because Bob knows Alice was there because of some secret he has that he doesn't want to give to anyone, not even to a judge[1]. So you need to design a protocol properly that will allow some third party who's off-line and who doesn't share all your secrets to also be able to know that Alice really was there.

Now the protocol which Zhou and Gollmann produced had a further property, which in fact we haven't looked at yet but this was fairness. By fairness we mean that maybe Alice will get proof that Bob took part in the protocol or maybe Bob will get proof that Alice took part in the protocol, but somehow it isn't fair unless they both happen at the same time. If it's possible to have a protocol running in which Alice gets evidence and Bob doesn't get evidence, then it's somehow unfair. So the protocol that we looked at, which Dieter had a hand in, is one in which the evidence for both parties is released simultaneously so you don't get one person having an advantage over another.

Each person in the running protocol must have a tangible thing (well not tangible in the sense that you can hold it, but as a bit pattern on your computer[2]) that you can present to a judge and say yes Alice definitely was present and she can't deny it. These two acronyms correspond to the two kinds of non-repudiation that this protocol looks at. NRO: Alice started this whole thing, she's the one who said I want to buy perfume. Bob will want evidence that she really did start this order for perfume, or whatever. Alice in her turn would like to have NRR: non-repudiation of receipt — that is she placed the order, her order was acknowledged and she wants to have a tangible thing as evidence that the order was acknowledged, that Bob really received the order. Finally, fairness is that these two pieces of evidence are available either to both parties or to neither party. The proofs that Giampaolo did don't yet cover all the eventualities, but in general, either because of a misadventure such as communications breaking down or somebody deciding to walk away from a transaction, things might go wrong and we want to try and show that can't happen.

The thing that we haven't done yet but I don't think it's too much harder, is to think of Alice and Bob being really malicious and not merely walking away from a transaction but using everything in their power to try and fake evidence or something like that. I don't think it would be much harder to do that but Giampaolo hasn't done that yet. He's also trying to verify SET remember.

Tuomas Aura: Is it a five year project?

Reply: Actually I don't think it will take very long, if you abstract out some of the junk there's not a lot left. Going back to SET, if you took that protocol that I looked at, if you got rid of all the digital envelopes, it would be

[1] or because knows enough to forge the evidence, but also knows that he didn't!
[2] LNCS 1361, pp 105–113.

such a trivial protocol it wouldn't even be worth looking at, it would be doing almost nothing. It's all those mechanisms that make it horrible, and of course the mechanisms are there for efficiency.

Now here's a protocol which I barely understand, but just about, and if I get stuck I think there might be someone here who can tell me what's going on. The idea here is that Alice wants to send Bob a message and the message is in this thing called C, it's called C for commitment, and I guess that commitment is very important but here actually Alice is the person making the commitment, and this commitment is, I gather, nothing more than the particular message that she wants to send to Bob, encrypted with the key K. Now the point is, Bob will receive this but he can't actually read the message until he receives the key and he gets the key later if he behaves himself. This thing here, non-repudiation of origin, is a thing that Alice signs to say, yes I did indeed send this commitment to Bob and once Bob has his hands on that he has a signed document from Alice saying, yes I did send this to Bob.

Now to continue the protocol, Bob has to reply and in particular he has this L which is a transaction number. So Bob replies to Alice, he echoes this label L and says yes, I received your request and now there's a trusted third party and this is by public key signature. Alice says OK, I'm happy and she now delivers the key needed to unlock the message, to a trusted third party, and I don't remember why this is necessary, but she signs something saying this is your key for this transaction label L so please make this available to Bob. And now there is a slightly tricky thing here, this so called ftp get. The point is, of course, that a bad person could cut the wire of your network and then Bob wouldn't be able to get his goods. So the idea is that this is some kind of courier service — a guy arrives on a motorbike and delivers personally to Bob the stuff in this message and then the same person zooms in on another motorbike and gives this stuff to Alice and not even Bill Gates himself can stop these motorbikes from reaching their destination.

So this is meant to be completely secure and unbreakable and I believe you can't tamper with them or intercept them or anything?

Dieter Gollmann: No I don't think that was the intention. The intention was rather to say that this information is on the server and the connections between A and B and the server will not be down forever. It is interesting to discuss what it means for the evidence to be available to both parties at the same time — does it mean they've got it, does it mean they can't get it or what does it mean?

Reply: Anyway, the key K is released to Bob who is now able finally to decrypt his message and at the same time the trusted third party signs a document himself saying, yes the transaction was completed, everyone did their part and I released this key for this session label, and so on. Alice wants NRR, non-repudiation of receipt, and Bob seeks evidence for NRO, non-repudiation of origin, so Bob cannot walk away from this transaction because Alice knows that he took part since Alice has got his acknowledgement, equally Bob knows that it was indeed Alice who did it.

It all makes sense on paper, but they always do don't they? So Giampaolo decided to have a look at this. slide describing other ways of doing it and you always say, these other ways of doing it are very good but of course our own way is even better. I remember at the 1999 workshop, the poor man got a lot of stick about what do you mean by "knows" when talking about computers[3] since computers don't know anything, but he never meant that. You see all the previous protocol proofs that we've done, and I think most people have done, have only cared about the knowledge of a spy or a bad person, so he's presumed to be an intelligent devious person who will manipulate things and try and cause bad things to happen, while everybody else is a robot. And so somehow the knowledge of the other people never mattered because they were robots anyway and they always played by the rules whatever they knew or didn't know. Of course what's happening in here is that everybody is a bad person, potentially, and so everyone's knowledge becomes important.

So Giampaolo had modified the set-up a bit so that you can talk about knowledge of individual agents which he's doing here, and he wants to be able to prove that if, for example, a piece of evidence such as a certain signed message is even remotely available to Bob, then a certain event happened — by event I mean typically sending a message. So if the evidence is remotely available to Bob, then a certain message was sent. In this case this particular message was a message sent by Bob, now maybe that's not so thrilling but you can then show that if this certain event "sent by Bob" occurred, then before that occurred a previous event "sent by Alice" also occurred and this gives you the chain of reasoning that justifies the non-repudiation.

Now you can imagine a court case — Alice ordered her perfume or her kalashnikov and it didn't come. So she goes to court and she goes to the judge and she presents this evidence. And I suppose the judge calls a respected expert in who says yes, this is a sound protocol that she has put in this evidence, it's correct and Bob really received the message and should have sent her that kalashnikov. So of course you don't expect the judge actually to do the protocol proof all over again, but just to take expert evidence. And the thing which arose, which was really rather more pleasant than I had hoped for, is that these proofs don't really involve big changes over the kinds of techniques we've seen before, going all the way back to 1996 or whenever I started doing protocol proofs in really the same style. You need slightly different lemmas and things, but there are no big changes here.

Just to show you the usual horrible symbolism here, this is the bit of the protocol specification in which the trusted third party receives the thing from Alice, he then puts together some pieces of evidence and he stores them on this ftp server. So this is what it looks like to specify the third event of the protocol. It's not bedtime reading but it's not a SET specification either, so it's manageable.

Giampaolo proved everything he wanted to prove, although for a slightly oversimplified model. As I said, he hasn't looked at fairness yet, which of course

[3] LNCS 1796, pp 91–94.

is an important property of a protocol, and he also hasn't looked at the possibility of Alice or Bob trying to falsify evidence. I imagine though it's pretty easy to show that they can't falsify evidence because it has be digitally signed, and I expect this proof to be trivial. We've looked at the more straightforward case where Alice and Bob might just try to walk away; and we have, in particular, done the case that if the collected knowledge of Bob includes this thing here called con_K, which the trusted third party makes, then he could only have received it from the trusted third party, there's no other way of his getting it, it can't be faked. And so the second level which connects to the first one is that if Bob has received it from the trusted third party, then Alice indeed stored the key with the trusted third party and you just keep reasoning backwards until you finally arrive at the point that Alice must indeed have taken part in the protocol, which is the whole point of it. And this is what the third tier looks like in symbols. This says that if Bob has NRO, non-repudiation of origin, which is that thing signed by Alice, then Alice sent the first message, so this is the ultimate goal for Bob which is that Alice really did order that kalashnikov or perfume.

Mark Lomas: You're also assuming that the trusted third party is trustworthy.

Reply: I think that if you don't trust the third party it falls to pieces, is that right Dieter?

Dieter Gollmann: If the server actually puts arbitrary stuff on the net, yes.

Reply: This actually reminds me of SET again, where you have certification authorities and in our model we assume that some of the certification authorities are crooked and, of course, you get the situation that you can only hope that it works if you are registered with a non-crooked certification authority.

Mark Lomas: I worry about the opposite case which is if, as you suggested, you turn up in a court with a proof and somebody wants to repudiate it they say, this third party was dishonest, when they did in fact participate.

Reply: I'm not in a position to comment on the practical feasibility of this in a court, I'm really looking at protocol but yes, you've got a point. If you actually want to give this stuff a legal status you've got issues but I think the courts have dealt with this. There are all kinds of crooked people out there, I mean there's even crooked lawyers and they appear in a courtroom all the time.

Mark Lomas: Some of them still owe me money.

Reply: Anyway, the two big limitations are that we don't allow the agents to try and fake things and we haven't yet proved fairness. I think Giampaolo thinks he can prove fairness[4]. For fairness you have to show that Bob has access to the evidence and so does Alice and I don't believe it can be difficult, it's just a matter of getting down to it, and I don't think it took him very long to do this.

Jan Jürjens: Do you know more exactly what is meant by reasonable?

Reply: He used a laptop so it wasn't, I was about to say a Cray but a Cray is an ancient machine that ...

[4] Doxastic logic at work.

Jan Jürjens: I meant the human resources.

Reply: I can do this stuff quite quickly, I suspect it took him a bit longer, but I really don't know if it took him weeks or days.

Jan Jürjens: So my question addresses more generally this approach of using mechanical theorem proving. You get a rather high degree of reliability of your results, at least once you've formalised your protocol. Could you estimate how many bugs are found in the last part of this verification process, namely the actual mechanical verification, as opposed to finding bugs in the earlier part where you set out to formalise the specification in the first place.

Reply: Well it is often said that the main point of formal methods is that it forces you to look closely at specification. Sorry to keep going back to SET, but I noticed what looks like a blunder in the cardholder registration part which is just obvious from looking at it, but which I wouldn't have bothered to do if I hadn't had to. Certainly that's an omen. It's very rare that I find bugs in protocols, the ones that I've looked at mostly have been analysed a lot already. It's funny, I found a bug in a suggestion in the BAN paper which turned out to be broken and I was annoyed to find out that someone else had found a bug in the same protocol. But that particular bug hadn't been found before.

Jan Jürjens: Yes, but did you find these bugs before you did the actual mechanical proving?

Reply: No, in this case I couldn't prove the protocol. What happens when you're doing a proof is you do an induction and the induction presents you with all the possible things that can happen in the protocol, and this corresponds to the situation that everything was fine until Alice sent message 3 and then something went wrong and you have to ask what could have gone wrong when Alice sent message 3. And you simplify the expression and it says, well if this existed and that was there, then that had been sent and then this could happen, and then you say, well is it possible for those things to have happened or can I prove they can't. And then you look at them and you say, well that could happen actually, and that could happen — you see if you could prove it couldn't happen, then you could just finish that case but if you suddenly see that all those things really could happen, then you put an attack together. If you're looking for attacks then model checking is a much faster way to find attacks; but when the model checker can't find an attack, you might want to try theorem proving.

Jan Jürjens: I was wondering about trying to make industry appreciate the use of formal methods in security more then maybe is currently the case. Formal methods are being used to some degree in industry but maybe not so much as in hardware verification. Probably one would also like to make them used more in the field of security, but then one would have to argue why it is not simply sufficient to formalise the specification.

Reply: Oh I see what you're saying, why bother to prove them. Well, yes sure, but I think once you've gone to the trouble of formalising it, it isn't that hard to do the proofs. The thing that's disappointing is that I've not been very good at teaching this technique. There are a couple of other people who can do these proofs but I'm a lot faster at it than anyone else and it would be nice if

other people could learn to do it. It is true that Giampaolo managed merchant registration by spending half a day on it when he decided to do it the way I had done cardholder registration, so that's pretty good. Then you suddenly realise that the effort taken to make this specification is actually much greater than the effort required to prove the specification because for specification it took months to go through thousands of pages of stuff about SET, and there's still a down until there's something that's small enough to type into your computer. So once you've done all that, you might as well go ahead and do the proofs which isn't that much extra work and is a lot more satisfying.

Roger Needham: It's worth commenting that when Burrows, Abadi and I did our first public key protocol, we thought we were going to exercise our logic. Well we received the protocol in a fax, wrote it on the whiteboard, and in the process of writing it on the whiteboard we saw what was wrong with it, burst into uproarious laughter and went to the pub. But the formalism was very useful because without it we would never have been able to persuade the perpetrators of the bug that it existed.

Tuomas Aura: Another question, I guess this may be more for Dieter than for you, about this non-repudiation and what does it mean. It looks to me like this protocol has the same kind of problem — I'm not going into what is a problem or a flaw in a protocol — as with the registered mail. If I send registered mail to you, the mailman will ask you to sign the receipt before he gives the mail to you. And then I can then get this receipt back and now I have proof that you have received something from me. In the same way, in this protocol, Alice will get a receipt saying that Bob has received a message with label L but the receipt says nothing about the content. With registered mail this is a problem because if I'm the receiver of the mail I should argue for that.

Reply: It's Alice's own fault if she didn't choose the unique sequence number and a unique key — she's only got herself to blame.

Tuomas Aura: Yes, but it's the non-repudiation, it's this question of proving things to this third party. It's just like you receiving an item of registered mail and then you sign the receipt and then you read it and throw it away and then later I say but I sent you this important document but you say no, you sent me some newspapers.

Reply: Well you don't want the postman to read your letter addressed to you, but as I said it's Alice's job to choose a label that uniquely identifies what she sent and if she can't choose a unique identifier then it's tough on her. Is there anything to add to that?

Dieter Gollmann: What's to add to it is that in Zhou's protocol, the intention was that the non-repudiation evidence consists of two parts. In this registered mail scenario, Alice gets evidence that Bob bought her letter number so and so, but no evidence about the content. She also presents evidence that Bob has access to the key to open the letter that is what she got from the ftp server, she has no evidence that Bob has actually opened the letter.

Tuomas Aura: Yes, also it doesn't help that you choose the label to represent the contents, because the receiver has to sign that before comparing the

contents and the label. So when the receiver is signing, the receiver does not know that this label tells him anything about the contents, it's just a number.

Bruce Christianson: The difficulty is some people might say well the label should include a hash of the contents because then you could be sure what was in the container, but you can't check that the object in the container has the correct hash without opening the container.

Tuomas Aura: You could play wonderful games here to say whether this commitment really needs to be a commitment to a message or is only a commitment to a message exchange.

Bruce Christianson: Yes, but I'm being presented with a receipt and told to sign here to acknowledge the receipt of one kalashnikov and I'm not allowed to see what's inside the box until I've signed the receipt, that's the difficulty.

Tuomas Aura: But what you could do, you could put an encrypted form of the commitment C into the receipt. Then you can at least prove that the receipt proves that you have received this certain message, and if I claim you have received another message, then you can ask me to decrypt that encrypted commitment C and prove it.

Using Authority Certificates to Create Management Structures*

Babak Sadighi Firozabadi[1], Marek Sergot[2], and Olav Bandmann[1]

[1] Swedish Institute of Computer Science (SICS)
{babak,olav}@sics.se
[2] Imperial College of Science, Technology and Medicine
mjs@doc.ic.ac.uk

Abstract. We address the issue of updating privileges in a dynamic environment by introducing *authority certificates* in a Privilege Management Infrastructure. These certificates can be used to create access-level permissions but also to *delegate* authority to other agents, thereby providing a mechanism for creating management structures and for changing these structures over time. We present a semantic framework for privileges and certificates and an associated calculus, encoded as a logic program, for reasoning about them. The framework distinguishes between the time a certificate is issued or revoked and the time for which the associated privilege is created. This enables certificates to have prospective and retrospective effects, and allows us to reason about privileges and their consequences in the past, present, and future. The calculus provides a verification procedure for determining, given a set of declaration and revocation certificates, whether a certain privilege holds.

1 Introduction

Many applications require a decentralised management of access permissions to their resources. We have identified the following kinds of applications in which management of access permissions should be decentralised.

- Applications operating in a highly dynamic environment, such as adaptive networks, where access permission updates have to be done frequently, locally, and partly automatically.
- Applications in which the administration of access permissions becomes so heavy that it affects the core business activities.
- Cases where security administrators are not fully trustworthy, and are potential security threats, whether deliberately or unintentionally.

To address the issue of updating access permissions, each organisation may define its own management structure. A management structure is normally a hierarchical structure defining how authorities and responsibilities are, or can

* This research is funded by Microsoft Research, Cambridge, UK.

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 134–145, 2002.

be, distributed within an organisation. In [5], the authors identify four typical roles for a management structure, namely User, Security Administrator, Manager, and Owner. An authority can be delegated within a domain using a predefined management structure. The idea is that the owner of an object has full authority concerning access and disposition of his object, and he can also delegate these authorities to managers. A manager defines a set of users and a set of objects as the administrative scope of a security administrator. The security administrator has the authority to give permissions to the users of his predefined domain to access the objects in this domain. Notice that a security administrator may or may not be part of the scope of his administration, which means that he may or may not be able to give himself access permissions.

In the current paper, we generalise the idea of management structure, because in real world scenarios there is a need for creating different types of management structures, and because the management structures may themselves be subject to frequent changes.

In [3] we distinguish between having a permission and being institutionally empowered, within a given organisation or management structure, to create a permission. In this paper, we employ the term 'authority' in place of 'institutional power' (because the term 'power', which has a technical meaning in this context, can also have unintended connotations). We use this notion of 'authority' as a prerequisite for creating and changing management structures as well as for creating and deleting permissions. We use the term 'privilege' as a general term to cover both 'authority' and 'permission'.

Separating the concept of authority from the concept of permission allows us to represent scenarios in which an agent has the authority to create a privilege (a permission or an authority) without having that privilege himself, or without having the authority to create that privilege for himself.

2 Delegation

In the information security literature, *delegation* normally describes the act of distributing access permissions to agents in a distributed system. Here, we allow for delegation of *privileges*, that is, for delegation of authorities as well as permissions. We distinguish between two possible kinds of delegation:

1. **Delegation as creation of new privileges:** The delegatee receives his own privilege which is independent of the delegator's privilege in the sense that if the delegator's privilege is revoked, then it does not necessarily mean that the delegatee's privilege is revoked. In this case the delegation is the act of issuing a new privilege. An agent may be an authority to create a privilege for another agent without having that particular privilege himself, or even without being an authority to create that privilege for himself. *Transfer* of a privilege can be seen as a creation of a new privilege and revocation of an old one.
2. **Delegation by proxy:** The delegatee does not receive his own privilege, but he can exercise the privilege of the delegator, in the sense that he *speaks*

for or *acts on behalf of* the delegator. In this case, if the delegator's privilege is revoked then the delegatee cannot exercise that privilege any more.

Some applications may require support for both kinds of delegation, and a framework capturing both would provide a flexible treatment for management of permissions. However, in this paper we will focus only on delegation of the first type.

3 Attribute Certificates and Privilege Management Infrastructure

Attribute Certificates (AC), sometimes called Privilege Attribute Certificates (PAC), have been proposed in various forums for securely providing privilege information using public key technology. The main proposal for use of AC is for distribution of authorisations [1]. Beside this, ACs can also be used for other purposes such as group and role assignment as suggested in [4], and for qualification certificates as suggested in [7]. Similar to the need for public key infrastructure (PKI) for use of public key certificates (PKC), there is a need for an infrastructure for the use of attribute certificates. In [1] this infrastructure is called the Privilege Management Infrastructure (PMI).

There are several reasons for decoupling an attribute certificate from a public key/identity certificate. For example:

- An agent's attributes (privileges) change more often than the public key associated to his identity.
- The authority issuing attribute certificates is usually not the same as the authority issuing public key certificates.

In PKI models, the public key certificate authorities (CAs) are usually large institutions at national or even international level, which are trusted or legally empowered to issue identity certificates. The structure formed by the relations between the CAs is fairly static and globally recognised by users of the PKI system.

Here, we argue that in contrast to the PKI model, in the PMI model the management structures for attribute authorities (AAs) can be highly dynamic and mainly determined locally, i.e. at organisational level. Being an authority to create a privilege is itself a privilege that is subject to change.

We propose the use of attribute certificates for delegating privileges and creating management structures. A certificate is a signed and time-stamped statement, which can be seen as an illocutionary act with a declarative force performed by its issuer[1].

[1] In speech act theory (see e.g. [6]), one distinguishes between different types of illocutionary acts. Here, we are mainly concerned with one type of illocutionary act, viz. declarative acts, or illocutionary acts with declarative force. The performer of a declaration (an illocutionary act with a declarative force) brings about the proposition

The issuing of a certificate can be seen as a declaration made by its issuer to bring about the propositional content given in the certificate. Notice that this type of certificate will not be effective unless its issuer has the authority for its content. A certificate issued by an agent without the necessary authority can be seen as an unsuccessful attempt by its issuer to declare its content.

The main components of an attribute certificate are:

- Issuer (the distinguished name, or the public key of the issuer)
- Subject (the distinguished name, or a pointer to the subject's public key certificate, i.e. its serial number)
- Attribute (the set of attributes that are associated to the subject)
- Validity Interval (the time-interval within which the given attributes are said to be valid)
- Signature (the digital signature algorithm used to sign the certificate)
- Certificate Serial Number (a unique ID number for the certificate, assigned to the certificate by its issuer)

An attribute certificate of this type says that the issuer is declaring that the subject has the set of attributes listed in the attribute field. The content of certificates can be a proposition stating various things, e.g. that the subject belongs to the group of administrators, that the subject is assigned the role of senior manager, that the subject is 20 years old, that the balance of the subject's bank account is £100, and so on. The validity interval field indicates the period of time for which the attributes hold for the subject.

We call an attribute certificate in which the issuer assigns some authority (or institutional power in the terminology of [3]) to the subject of the certificate as an *authority certificate*. An authority certificate can be used by an agent to delegate some authority to others, as for example when the owner of an object delegates, to some managers, the authority to create permissions to his object.

An authority certificate may be used to create an authority to initiate several chains of authority delegations. By expressing constraints on future delegations one defines the scope of future management structures in an organization [2]. However, any delegation chain must originate from a *source of authority* for what is delegated. Who is recognized as a source of an authority, and in what conditions, is a policy issue and is application–domain specific. Different applications may have different policies for recognising sources of authorities. In many applications, but not all, the owner(s) of a resource are recognised as sources of authority for permissions and authorities concerning that resource, for example.

Figure 1 shows the control model given in [1] in which the claimant receives a privilege, directly or indirectly, from the source of authority. In order for the

that is the content of his declaration, if, and only if, he has the required authority for doing so. It is also possible to view some certificates as *assertions*, that is, in speech act terms, as illocutionary acts that assert the truth of a proposition without necessarily creating it. However, in this present framework nothing is gained from making the distinction and so we choose to treat all certificates as having declarative illocutionary force.

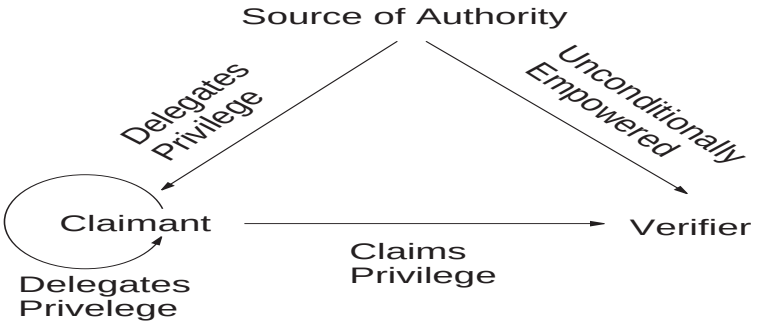


Fig. 1. The Control Model

claimant to exercise his privilege (his access permission), the verifier needs sufficient credentials, in the form of certificates, to verify the claimant's privilege. Note that there may be a number of intermediary authorities between the source of authority and the final claimant of the privilege. This means that the set of certificates provided to the verifier may contain a number of authority certificates showing a proper delegation chain originating from the source of authority and leading to the claimant.

There are at least two possible models for the control model:

1. *Centrally updated model*: Any delegation step is reported directly to the verifier, by sending the authority certificates to the verifier, who updates the existing management structures in its database.
2. *Distributed model*: At the delegation step, the delegatee (the claimant) receives his new privilege and all the intermediate authority certificates originating from the source of authority to the delegator. The claimant provides this set of certificates to the verifier at the time of his privilege request.

Each of these models has a number of advantages and disadvantages making them suitable for different kinds of applications. We will not discuss the issues associated to each model in this paper. However, in both models the verifier needs a mechanism for deciding, given a set of certificates, whether the claimant's privilege holds. In the next section, we describe a calculus that can be used by the verifier for reasoning about privileges and delegated authorities.

4 The Framework

In this section we present a framework for a privilege management system using attribute certificates. The issuing of certificates is the only type of action considered in this framework. Issued certificates are submitted to a privilege verifier as shown in figure 1.

The privileges managed by the verifier are of the following two types.

- *Access-level permission* (e.g., permission to read or write a file, or permission to execute a program).
- *Management-level authority* (i.e., authority to declare an access-level permission, or authority to declare a management-level authority).

Here, we consider only two types of certificates, declaration and a simple form of revocation.

- Declaration certificates are represented as:

$$declares(issuer, p[I], time\text{-}stamp, id).$$

We interpret a declaration certificate as an action description for a declaration performed by its *issuer* at time *time-stamp* to bring about that privilege p holds during time interval $[I]$. The *id* is the unique id of the certificate, either generated by its issuer or generated by the privilege management system which the verifier is a part of. Validation of signatures is of course an essential component of verifying a certificate, but signatures are not part of the reasoning process for verifying that a privilege holds, and for this reason signatures do not appear in the representation of certificates.

- Revocation certificates are represented as:

$$revokes(issuer, id, time\text{-}stamp).$$

Note that a revocation certificate does not have an *id* itself, but it contains the *id* of the certificate that it is revoking. In the present framework, we do not allow revocation of a revocation certificate. Of course, one can imagine scenarios in which there is a need for recovery from earlier revocations. However, in the current framework we do not consider this type of scenario.

4.1 Semantics of the Calculus of Privileges

Informally, the idea is that a privilege p holds at a time-point t when there is a certificate C declaring that p holds for some interval I containing t; the certificate C moreover must be 'effective' at t, in the sense that it was issued by s at a time when s had the authority to declare p to hold for interval I. The authority of s, in turn, requires a certificate that was effective at the time C was issued — and so on, in a chain of effective certificates back to some source whose authority can be accepted without certification (as determined by the organisational structure).

The following definitions make these ideas precise. The complication is that we are here dealing with two levels of time — the time at which a certificate is issued, when it can be effective or not, and the time at which a given privilege holds or not. It it important to notice that we do *not* require that a certificate declaring privilege p for time interval I must be issued before I. In our scheme, a certificate can create a privilege *retrospectively*. We comment further on this and other features after presenting the definitions.

Definition 1. Let *AGN, ACT*, and *OBJ* be the sets of agents, actions, and objects, repectively. We define the set of privileges Φ as:

- $perm(s, a, o)[I] \in \Phi$, if $s \in AGN,\ a \in ACT$, *and* $o \in OBJ$;
- $pow(s, \phi)[I] \in \Phi$, if $s \in AGN$, *and* $\phi \in \Phi$.

We define the set of declaration certificates Σ^+ and the set of revocation certificates Σ^- as:

- $declares(s, \phi, t, id) \in \Sigma^+$, if $s \in AGN,\ \phi \in \Phi,\ t \in \mathbf{R},$ and $id \in \mathbf{N}$, where $\mathbf{R}$ denotes the real numbers, and $\mathbf{N}$ denotes the natural numbers;
- $revokes(s, id, t) \in \Sigma^-$, if $s \in AGN,\ id \in \mathbf{N},$ and $t \in \mathbf{R}$.

Privileges of the form $perm(s, a, o)[I]$ denote access-level permissions, while privileges of the form $pow(s, \phi)[I]$ denote management-level authorities.
(In the definitions above $[I] = [t_{start}, t_{end}]$, where $t_{start} \in \mathbf{R}$, $t_{end} \in \mathbf{R}$ and $t_{start} \leq t_{end}$.)

Definition 2. We define a certificate database to be a tuple $\mathcal{D} = (\mathbf{SoA}, \mathbf{D}^+, \mathbf{D}^-)$, where $\mathbf{SoA} \subset \Phi$ is a finite set of *Source of Authority* privileges, $\mathbf{D}^+ \subset \Sigma^+$ is a finite set of declaration certificates and $\mathbf{D}^- \subset \Sigma^-$ is a finite set of revocation certificates. We adopt the following constraints on a certificate database.

1. If $declares(s_1, \phi_1, t_1, id) \in \mathbf{D}^+$, and $declares(s_2, \phi_2, t_2, id) \in \mathbf{D}^+$, then $s_1 = s_2,\ \phi_1 = \phi_2$, and $t_1 = t_2$,

   This says that $\mathbf{D}^+$ cannot contain two different certificates with the same id.

2. If $declares(s_1, \phi, t_1, id) \in \mathbf{D}^+$ and $revokes(s_2, id, t_2) \in \mathbf{D}^-$, then $s_1 = s_2$ and $t_1 \leq t_2$.

   This says that a certificate can be revoked only by its issuer and not before it is declared. In fact, the first restriction can be relaxed but this introduces the need for extra components which are omitted here for simplicity.

3. If $revokes(s_1, id, t_1) \in \mathbf{D}^-$ and $revokes(s_2, id, t_2) \in \mathbf{D}^-$, then $s_1 = s_2$ and $t_1 = t_2$.

   This says that there cannot be two revocations of the same declaration certificate in the same database. We adopt this restriction to simplify the database in order to streamline the theory.

Definition 3. Let $\vdash$ be the *validates relation* between a privilege and a declaration certificate, where

$$pow(s, \phi)[I] \vdash declares(s, \phi, t, id),\ \text{if } t \in [I];$$

and, if $\Gamma \subseteq \Phi$, then

$$\Gamma \vdash d,\ \text{if } \exists\ q \in \Gamma \text{ such that } q \vdash d.$$

Definition 4. We define the set of *effective* declaration certificates $\mathbf{E}_{\mathcal{D}}(t) \subseteq \mathbf{D}^{+}$ of a database $\mathcal{D}$ at a certain time t, as:

$$\mathbf{E}_{\mathcal{D}}(t) = \{declares(s, p[I], t_1, id) \in \mathbf{D}^{+} \mid t \in [I] \;\&\; (revokes(s, id, t_2) \in \mathbf{D}^{-} \rightarrow t_2 > t)\}.$$

Definition 5. Let $d_1, d_2 \in \mathbf{D}^{+}$, where $d_1 = declares(s_1, \phi_1, t_1, id_1)$ and $d_2 = declares(s_2, \phi_2, t_2, id_2)$. We define the *supports* relation $S_{\mathcal{D}}$ as follows:

$$d_1 \; S_{\mathcal{D}} \; d_2 \text{ if } d_1 \in \mathbf{E}_{\mathcal{D}}(t_2) \text{ and } \phi_1 \vdash d_2.$$

Definition 6. The set of certificate chains $C_{\mathcal{D}}$ in a certificate database $\mathcal{D}$ is the transitive closure of $S_{\mathcal{D}}$.

Note that, $C_{\mathcal{D}}$ at a time-point t may contain chains that are no longer of use; chains that can be extended with further certificates; and chains that are dormant (see figures in the following section).

Definition 7. We define the set of true privilege statements at a time-point t, in our calculus of privileges, by defining function $h_{\mathcal{D}} : \mathbf{R} \rightarrow 2^{\Phi}$ as:

$$\begin{aligned} h_{\mathcal{D}}(t) = \{p \mid p[I] \in \Phi \;\wedge\; & \\ & (p[I] \in \mathbf{SoA} \vee \\ & (d_1, declares(s, p[I], t_2, id)) \in \mathbf{C}_{\mathcal{D}} \wedge declares(s, p[I], t_2, id) \in \mathbf{E}_{\mathcal{D}}(t) \;\wedge \\ & \mathbf{SoA} \vdash d_1)\}. \end{aligned}$$

We also say that a privilege p *holds* at time-point t when $p \in h_{\mathcal{D}}(t)$.

4.2 Examples

In this section we present some diagrams to illustrate the formal definitions just given.

Each horizontal line of the diagram represents a (declaration) certificate. The vertical arrows depict the times at which certificates were issued. The shaded rectangles show the time intervals of the privileges declared by the certificates. For simplicity, the examples show only one privilege for each certificate, though this is not a restriction of the framework. Short vertical bars depict revocations; in Figure 2, the certificates issued at d_3, d_2, and d_4 have been revoked. The certificates issued at d_3 and d_2 were revoked before the associated privilege intervals expired, as indicated by the lighter shading. The certificate issued at d_4 declared a privilege for an interval which begins before the certificate was issued. The framework allows certificates to make retrospective declarations. Although not shown in this figure (but see Figure 3), it is possible that a certificate issued at time-point t could declare a privilege that holds for an interval entirely in the past of t.

The arrangement of the vertical arrows is intended to illustrate the *supports* relation between certificates. So the certificate issued at d_1 (which was issued

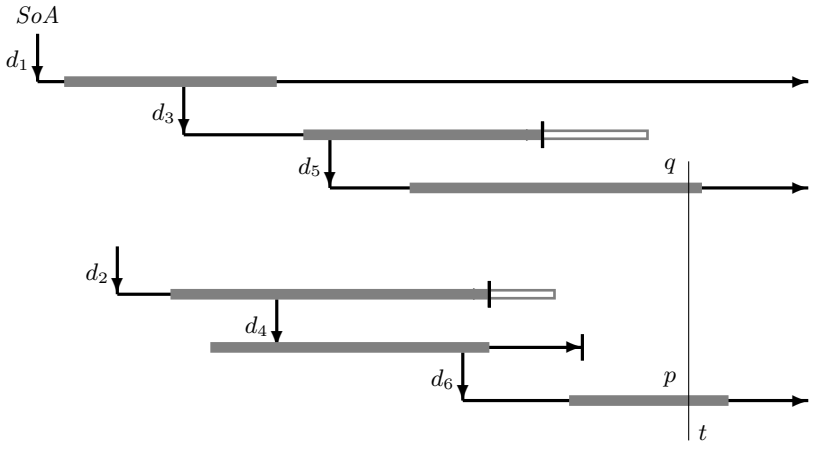


Fig. 2. Two certificate chains

by some source of authority *SoA*) supports the certificate issued at d_3, which in turn supports the certificate issued at d_5. The chain (d_1, d_3, d_5) is *rooted* since the certificate issued at d_1 is rooted (it was issued by a source of authority, we are supposing). The chain (d_2, d_4, d_6), on the other hand, is not *rooted*: d_2 is not issued by a source of authority, nor supported (we are supposing) by a rooted certificate. We call such chains *dormant chains*. Therefore, the privilege q declared by the certificate issued at d_5 holds at the time-point t shown in the diagram, but the privilege p declared by the certificate issued at d_6 does not hold at t (assuming there are no other chains besides those shown in the diagram).

Because certificates can have retrospective effects, a dormant chain can become rooted as a result of a later declaration. This is illustrated in Figure 3, where the previously dormant chain (d_2, d_4, d_6) becomes rooted as a result of the issuing of the declaration certificate at d_7.

Retrospective effects of this kind can be used to implement a type of *approval mechanism*. In the example, the issuer of certificate d_2 creates one or more chains of privilege-creating certificates. These remain dormant until eventually made effective ('approved') by the issuing of a suitable certificate at d_7.

Some observations:

- revoking a certificate declaring $p[I]$ before the interval I has started means that this certificate can never be used to create a chain (the privilege p can never be exercised on the basis of this certificate);
- revoking a certificate declaring $p[I]$ after the interval I has ended has no effect — any chain created using this certificate is not destroyed by the revocation;
- more generally, revocation of a certificate that has already been used to create a chain will not affect the chain — 'what's done is done', according to the specific notion of revocation supported in the present framework.

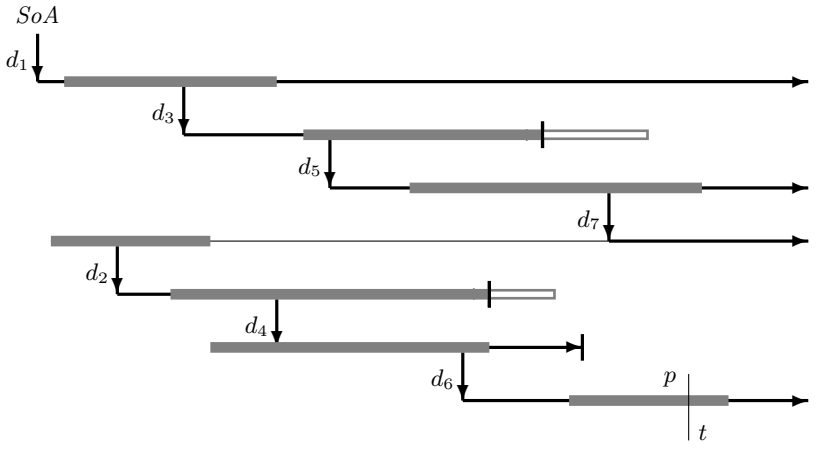


Fig. 3. A rooted certificate chain

As observed in the introductory section, it is also possible to conceive of other forms of revocation which can undo past effects. These forms of revocation are not discussed further in this paper, but will be presented in an extended version of the framework in future work.

4.3 The Calculus of Privileges

In this section we present a logic program which implements the semantics given above. The predicate $holds(P, T)$ is used to query the system to determine whether a privilege P holds at time-point T given a set of certificates database. (The program can be executed as a Prolog program as it stands, once the symbol ':' is declared as an infix functor).

[PC 1.] $holds(P,T) \leftarrow C = declares(S, P{:}[T_s, T_e], T_0, ID),$
$effective(C,T),$
$T_s \leq T \leq T_e.$

[PC 2.] $effective(C,T) \leftarrow C = declares(S, P{:}[T_s, T_e], T_0, ID),$
$rooted(C),$
$T_0 \leq T,$
$not\,[revokes(S, ID, T_1),\ T_1 \leq T].$

[PC 3.] $rooted(C) \leftarrow chain(C1, C),$
$C_1 = declares(S, P, T_0, ID),$
$sourceOfAuthority(S, P).$

[PC 4.] $chain(C, C).$

[PC 5.] $chain(C1, C2) \leftarrow supports(C1, C2).$

[PC 6.] $chain(C1, C2) \leftarrow supports(C1, C3),$
$chain(C3, C2).$

[PC 7.] $validates(pow(S, P){:}[T_s, T_e], C) \leftarrow C = declares(S, P, T, ID),$
$T_s \leq T \leq T_e.$

[PC 8.] $supports(C1, C2) \leftarrow C1 = declares(S_1, Q{:}[T_s, T_e], T_1, ID_1),$
$C2 = declares(S_2, P, T_2, ID_2),$
$validates(Q{:}[T_s, T_e], C2),$
$not\,[revokes(S_1, ID_1, T_3),\ T_3 \leq T_2].$

In this program it is assumed that there is an up-to-date *source of authority* database, and that a source of authority privilege is created using a declaration certificate issued by the source of authority of that privilege.

The program can be generalised very easily to define a predicate $holds(P, T, T_D)$ representing that, according to the certificates issued up to and including time T_D, privilege P holds at time T. This generalized form allows one to query not only the current state of the set of certificates database, but all past states as well. The required modification is very straightforward. Details are omitted here.

5 Implementation Issues

The scheme presented in the preceding sections supports many different models. For example, in a centralized system (ref. Figure 1), the *holds* relation can be materialized, that is, computed and stored for immediate look-up as required by the verifier, and updated incrementally whenever a new declaration or revocation certificate is received. There are well-established techniques for executing logic programs in this fashion. The database of all declaration and revocation certificates can also be queried to determine which privileges held at which times in the past, which may be useful for, e.g., auditing purposes.

In a distributed model, the 'claimant' presents a portfolio of certificates, which provide a set of certificates database on which the verifier can execute the reasoning calculus directly. Here there are several further options for the treatment of revocations. In one model the verification engine generates requests to a trusted revocation server as required. In another possible model, the verification engine checks locally against a list of revocation certificates broadcast from time to time by the revocation server. There are many other possible combinations. The point is that the same reasoning mechanism presented in the preceding section can be applied in each case.

Although we have presented the reasoning engine as a logic program, which can be executed in Prolog or in some other logic programming system, it is also easy to re-code the algorithm in another database formalism or programming language if that is preferred. We leave detailed discussion of such implementation techniques to another paper.

6 Conclusion and Further Extensions

We have addressed the issue of privilege management by using a type of attribute certificate that we call an authority certificate. We have made a distinction between two types of delegations — delegation as creation of new privileges, and delegation by proxy — though only the first of these is discussed in this paper.

We have presented a semantic framework and a calculus for reasoning about privileges based on a distinction between access level permissions and management level authorities. The calculus can be used by a verifier to check whether a certain privilege holds given a set of declarations and revocations. The framework supports flexible reasoning with time, such that certificates can be issued to create privileges in the past, present and future.

In the framework we present in this paper we have kept revocation certificates as simple as possible. Only the issuer of a declaration certificate can revoke it, and once revoked, a certificate cannot be reinstated. We are currently extending the framework by allowing more complex revocation certificates providing a richer set of revocation mechanisms. Finally, we intend to extend the framework with roles. These do not affect the core calculus but introduce a number of further choices which we are currently investigating.

Acknowledgement. We would like to thank Jason Crampton for suggesting a number of improvements to an earlier draft of this paper, which made the final draft tidier and easier to read.

References

1. Final Proposed Draft Amendment on Certificate Extensions(v6). generated from Collaborative ITU and ISO/IEC meeting on the Directory, April 1999. Orlando, Florida, USA.
2. Olav Bandmann, Mads Dam, and Babak Sadighi Firozabadi. Constrained Delegation. 2001. In preparation.
3. Babak Sadighi Firozabadi and Marek Sergot. Power and Permission in Security Systems. In B. Christianson, B. Crispo, and M. Roe, editors, *Security Protocols*, number 1796 in Lecture Notes of Computer Science, pages 48–53, Cambridge, UK, April 1999. Springer Verlag.
4. R. J. Hayton, J.M. Bacon, and K. Moody. Access Control in an Open Distributed Enviroment. In *Proceeding of IEEE Symposium on Security and Privacy*, pages 3–14, Oakland, CA, 1998.
5. J. Moffett and M. Sloman. Delegation of Authority. In I. Krishnan and W. Zimmer, editors, *Integrated Network Management II*, pages 595–606. North Holland, April 1991.
6. John R. Searle. *Speech Acts*. Cambridge University Press, Cambridge, 1969.
7. Petra Wohlmacher and Peter Pharow. Applications in health care using public-key certificates and attribute certificates. In *Proceedings of the 16th Annual Computer Security Applications Conference 2000 (ACSAC 2000)*, pages 128–137, New Orleans, Dec. IEEE Press.

Using Attribute Certificates for Creating Management Structures (Transcript of Discussion)

Babak Sadighi Firozabadi

Swedish Institute for Computer Science

I will start with presenting the motivation behind this work, then I will give some background from where we have borrowed some of the ideas. I will try to connect the ideas and give a complete picture, at the end. I will discuss the notion of delegation and present what we mean by delegation and then give a framework for representing and reasoning about delegations using attribute certificates. This research is funded by Microsoft Research here at Cambridge.

OK, so what is the motivation? We have seen several types of application where there is a need for decentralised management of privileges. It is important to mention that we are not talking about distributing privileges, but decentralising management activities within an organisation or between several organisations. We have for example looked at similar type of scenarios that was presented in the talk by Silja and Tuomas earlier[1]. What they (the military people that we talked to) are interested in is that they don't want to have a static access control list, because the environment they are working in is changing frequently. They want a model to capture the dynamics of access privileges. The issue is how to make these privileges dynamic and at the same time keep certain control over their updates. In other words the issue is to give flexibility and at the same time keep the control.

Let me give you an example from business-to-business type of applications. We were looking at an Extranet system between three companies A, B1, and B2. A is a manufacturer of components to mobile phones and both B1 and B2 are manufacturing mobile phones. The purpose of the system is to give B1 and B2 possibility to order components and do other things through the system. When A tested the system with only 100 users they realised that the administration of access privileges is a big issue. The problem is that both B1 and B2 are large and dynamic organisations. An employee can be involved in several projects and project members can change on daily or weekly basis. This requires an administration on A's side to update access privileges for employees of B1 and B2. What A is interested in is to let B1 and B2 update their access privileges but again in some controlled way. At the same time B1 and B2 wanted to keep certain information about their organisational structure and their employees secret. This example just illustrates the kind of applications and problems that we are studying.

Now I want to discuss some background work and to talk about ideas that we have borrowed from several areas. We distinguish between what we call insti-

[1] Silja Mäki, these proceedings.

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 146–150, 2002.

tutional power or an authority and a permission. There is a difference between having a permission and having the power to create that permission or creating a power for someone else. One does not necessarily imply the other. We can think of a manager who is empowered to sign a document giving permission to someone but perhaps not having that permission himself, or not even being empowered to create that permission for himself.

Another type of scenario is when a manager has got the power to sign purchase orders for his company, and he is permitted to sign orders for up to $3,000, he is permitted for that but not permitted for amounts higher than that. However, if he signs an order higher that $3,000, it is a valid order and a legally binding document. One could say, OK he's empowered to do it but he's not permitted to exercise his power in certain conditions.

In speech act theory there is a taxonomy of illocutionary acts. The type of illocutionary act that we are focusing on, that I'm going to describe here, is an illocutionary act with a declarative force, which means that the speaker creates a fact, brings about a fact, by declaring it, by saying it. It has effect only if he's empowered to do that. This is something other than informing someone by saying that a certain fact is true: the actual act of declaration is changing the state of affairs[2].

Now what is the purpose of issuing attribute certificates? Sometimes it's just to inform someone that a certain fact holds. Other times it is that the person who is issuing, who is signing, is actually changing the state of affairs, and bringing about certain facts. So maybe this kind of taxonomy could be useful for a discussion on certificates because one *could* see certificates just as signed statements and nothing more, but again issuance of a certificate could be seen as an action which has a certain effect — is the effect only to change a belief of the receiver or is it to change the state of the institution?

Usually, as I understand it from many approaches, delegation means letting someone act on one's behalf, whereas we could think of delegation as creating an independent privilege for someone. So if Marek, who is my supervisor, empowers me to sign a purchase order and at a later point in time his power is revoked or deleted because he is not my supervisor anymore, then this does not necessarily mean that I'm not empowered any longer. However, if he gives me the power to vote on his behalf and if his power to vote is revoked, then I cannot do anything about it — I don't have any privileges anymore.

Going back to what I was discussing before, we want to decentralise the management of privileges — how can we do that? One way of doing it is to create a management structure, and by that we mean the way in which management authorities and responsibilities are delegated in an organisation, perhaps in a hierarchical structure. But not only that, we also need a mechanism for updating these structures. We use delegations expressed in attribute certificates as a way of creating new facts (privileges). Now from a database containing time-stamped certificates we are able to deduce the actual management structure in an organisation.

[2] *cf* Genesis 1, v3.

In our framework the only actions that can be performed are issuance of certificates. The issuer declares that a subject has some attributes, some privileges, for certain period of time. The actual act of issuance is time-stamped and each certificate gets its own id.

(A revocation certificate would be that the issuer of the revocation certificate revokes an existing declaration certificate in the system. We don't deal with revoking revocation certificates just to keep the framework as simple as possible.)

A declaration certificate could be seen in two different ways. One is that the action is performed at the time the certificate was issued which requires that at that time point the issuer should be empowered to do that. The other one would be that the action is performed at any time in the time interval, so if we look at the certificate and if its validity time interval is between zero and ten, then at each time point between zero and ten one could say that the action is performed at that time point. This require that the issuer is empowered for that action at any time point in the validity interval of the certificate. So these two give us two different revocation semantics but now I just discuss the first one.

Now we need a calculus for reasoning about certificates and their issuance time and validity intervals.

We instantiate all management activities as issuing attribute certificates and have a database that keeps the history of their issuance.

A privilege is brought about by an agent for a certain time period T, by issuing an attribute certificate that contains the privilege and has validity interval T. This is the case only if the issuer is empowered to bring about that fact at the issuance time of the certificate. In the same way, we define the termination of a privilege by either the certificate which initiated it having expired or there is another certificate which explicitly revokes the first one. And again, this revocation is an act which should be supported in terms of empowerment.

Here, we let users declare what they want, which may have effect or not depending on if they are empowered or not, but at the same time if I declare something now, it can be the case that at a later time point I get somebody giving me the required power retrospectively. Since we have a declarative way of reasoning, we can go back and ask, OK are there any certificates issued in past that this new certificate is supporting and what are their effects? So at each time point, based on the database of certificates we can reconstruct the management structure that held at that time.

So we're talking about decentralisation of managing privileges. We would like to change the focus from having a permission towards having the power to create a permission: who can create a permission and in what conditions? In this framework we are working on mechanisms for creating these structures and we define delegation as creating facts rather than letting someone act on one's behalf. We use attribute certificates as the way of creating facts, that means we consider them as statements with declarative forces. What we have done is defining a calculus and a framework for reasoning about a database of attribute certificates to construct management structures.

Hiroshi Yoshiura: You distinguished two types of delegation. But I cannot understand how you describe the two types of delegation in the event calculus.

Reply: That's a good question. Actually, I am not discussing the second type of 'acts on behalf' in terms of event calculus. I'm not sure if it is useful to do it in that way or not, I have to look into it. But at least the first type can be seen as a separate action done at a certain time, as a database of events that have happened, and in that case event calculus helps us.

Hiroshi Yoshiura: What is the essential difference between the two types of delegation? It seemed that the lower type of delegation to only give privilege can include the higher type of delegation because the delegatee can sign everything.

Reply: Yes, I think it's more a case of independence. Maybe one can define one of these in terms of the other but the essential difference I see, is that the first type is independent. When I receive a privilege from someone, then from this time point onwards my privilege is mine and it has nothing to do with him. He can be a spy and kicked out of the organisation but it doesn't mean that I don't have my permissions or powers anymore. Whereas in the second case, I don't have a permission by myself, I'm using, I'm exercising, his rights.

Stewart Lee: When you are delegating you have to be careful to consider what it is that you're delegating. You can delegate an authority, you can delegate a permission, you can delegate responsibility, but what you must never do is get into a situation where nobody has responsibility anymore. And I think you'll find, if you think about it in that way, that's the fundamental difference between the two forms of delegation that you had on the slide because the second example was you acting as a proxy for Marek.

Reply: Yes, and Marek has the responsibility.

Stewart Lee: If Marek dies then you can no longer act as his proxy because Marek is responsible.

Reply: Exactly.

Stewart Lee: And you can delegate responsibility but the person to whom you are delegating is going to be crazy if you delegate responsibility without authority. So you have to be very careful with what you're delegating.

Bruce Christianson: I think that there is a second difference between Babak's two types of delegation. In the second case, the delegation is acting as a proxy for somebody. In the first case, it's creating a new privilege which may be to do something that the delegator doesn't have the power to do themself.

Stewart Lee: I could argue that any delegator who is responsible for something has to have the power to do it, or to see it, or to massage it, otherwise he's nuts.

Reply: No, when we were discussing with the military people, they were saying that in many situations the general has got the power to decide who's going to use this equipment but since he hasn't got the right education for that, the system should not allow him to use it himself.

Bruce Christianson: For example you wouldn't allow him to fly the plane because he's not licensed to fly that particular plane.

Matt Blaze: In language terms this looks very simple. As a programming language person, I would see this as a question of the time of binding. In the first case it's very simple, the delegation is valid if it was valid at the time of delegation. And the second type is valid if it's valid at the time it's used.

Bruce Christianson: There's actually a more fundamental distinction than that.

Matt Blaze: But that's the functional description.

Bruce Christianson: No that isn't quite right. There's one type of delegation which is handed on and then continues. There's a second type of delegation which I have to make continually and if I lose my right to exercise that power, then you lose the power at that moment.

Matt Blaze: So it's evaluated at the time it's used rather than at the time it's delegated.

Bruce Christianson: Yes, but look what is being evaluated. In the second case it's ...

Matt Blaze: ... the authority of the delegator to exercise the right.

Bruce Christianson: Yes, yes, exactly. Whereas in the first case we are evaluating the power of the delegator to *confer* the right upon the delegatee.

Reply: The fundamental difference here is that the second type — by proxy — can be implemented purely with certificates and then if you revoke it you revoke the certificates, but the first type requires a trusted computing base to enforce the policy.

Matt Blaze: It requires the delegation to be on-line in some sense.

Reply: Yes.

Tuomas Aura: But it's time-static.

Stewart Lee: [head in hands] I feel like we've found the head of a pin and we're all trying to dance on it. [laughter].

Bruce Christianson: There's a separate argument about mechanism. Even if you decentralise management authority, you could in theory still have one central ACL and event log that everybody updates. How or whether you choose to distribute that database is an orthogonal question.

Reply: Exactly.

Trust Management and Whether to Delegate

Simon N. Foley

Department of Computer Science,
University College, Cork, Ireland.
s.foley@cs.ucc.ie

Abstract. Trust Management systems use security credentials to determine whether action requests are authorized. This paper examines how they might be used in determining whether requests for delegation are safe.

1 Introduction

Delegation certificates/credentials specify delegation of authorization between public keys. When an application receives a request to execute a particular action, then authentication notwithstanding, the application may use the credentials to determine whether the request is authorized. Trust management systems such as KeyNote [3] and SPKI [4] provide assistance to applications in making these decisions. For example, an application can request authorization judgments using the KeyNote API without having to be aware of the details of how to interpret credentials or the details of the underlying security policy. As an engineering paradigm, the desirability of such separation of concerns is well understood, as it can lead to applications that are easier to develop, understand and maintain.

In addition to judgments on authorization, applications may also desire assistance in determining whether it is safe to delegate authorization for some action to a public key, that is, to write and sign a new credential. As with authorization judgments, it is preferable that the details of these delegation judgments are separated from the application. In this paper we consider how a trust management system might provide such assistance.

The problem to be discussed is depicted in Figure 1. The owner of public key

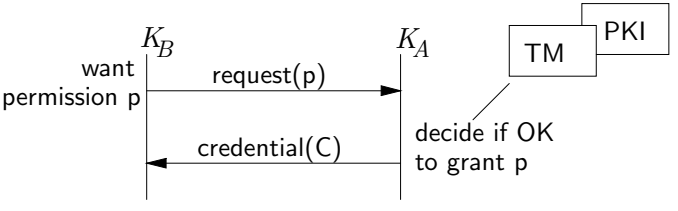


Fig. 1. Request for Delegation of Authorization

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 151–157, 2002.

K_B requests authorization permission p from an application that owns public key K_A. K_A may use credentials presented by K_B to determine whether it is safe to sign a credential C delegating permission p to K_A (principles represented by their public key).

Section 2 discusses the trivial case where the credentials presented prove that K_A has already been delegated authorization p; writing a new credential is simply certificate reduction. Section 3 considers the case where K_A uses additional 'local' rules to determine if delegation is safe. When handling a request, K_A may be willing to offer an alternative, but similar, credential to K_A. Section 4 considers how techniques from similarity-based retrieval for case-based reasoning systems might be used to support flexibility in requests.

2 Defacto Delegation and Reduction

A simple model is used to represent delegation of authorization between public keys. A signed credential, represented as $\{\!|\, K_B, p \,|\!\}_{K_A}$, indicates that K_A delegates to K_B, the authorization permission p. Permissions are structured in terms of lattice $(PERM, \leq, \sqcap)$, whereby given $p \leq q$ means that permission q provides no less authorization than p. A simple example is the powerset lattice of $\{\mathsf{read}, \mathsf{write}\}$, with ordering defined by subset, and greatest lower bound defined by intersection.

Given a credential $\{\!|\, K_B, q \,|\!\}_{K_A}$ and $p \leq q$ then there is an implicit delegation of p to K_B by K_A, written as $(\!|\, K_B, p \,|\!)_{K_A}$. Two reduction rules follow.

$$\frac{\{\!|\, K_B, p \,|\!\}_{K_A}}{(\!|\, K_B, p \,|\!)_{K_A}} \qquad \frac{(\!|\, K_B, p \,|\!)_{K_A}; p' \leq p}{(\!|\, K_B, p' \,|\!)_{K_A}}$$

If delegation is regarded as transitive, and if K_A delegates p to K_B, and K_B delegates p to K_C, then it follows that K_A implicitly delegates p to K_C.

$$\frac{(\!|\, K_C, p \,|\!)_{K_B}; (\!|\, K_B, p' \,|\!)_{K_A}}{(\!|\, K_C, p \sqcap p' \,|\!)_{K_A}}$$

This corresponds to SPKI certificate reduction [4] (greatest lower bound is equivalent to SPKI tuple intersection). It is not unlike a partial evaluation over a collection of KeyNote credentials, resulting in another credential. At this point, we do not consider permissions that cannot be further delegated.

These rules are used by a trust management system to determine whether a request for an action (permission) is authorized. If the local policy specifies that K_A is authorized for p, and delegation $(\!|\, K_B, p \,|\!)_{K_A}$ can be deduced, then it follows that K_B is authorized for p.

The trust management system can be used to provide guidance when writing new credentials. From Figure 1, if the owner of K_A can derive $(\!|\, K_B, p \,|\!)_{K_A}$ from the credentials presented by K_B or via a PKI, then it can safely write and sign a new credential $\{\!|\, K_B, p \,|\!\}_{K_A}$ for K_B. Alternatively, K_A may write a

weaker credential $\{\!|\, K_B, q \,|\!\}_{K_A}$ for the highest $q \leq p$ for which $(\!|\, K_B, q \,|\!)_{K_A}$ can be deduced.

We call this *defacto* delegation as it does not confer any additional authorization on the requester. Advantages of this kind of delegation have been discussed in [1]. It can be used to reduce the size of delegation certificate chains to be stored and/or presented; this may be useful for limited devices such as PDAs, and so forth. It can also provide for a degree of anonymity: long delegation chains may reveal sensitive information on how authority was acquired.

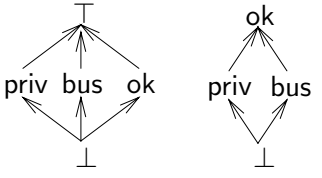
3 Dejure Delegation and Reduction

One difficulty with defacto delegation is that it does not help a principle to decide whether it is safe to confer new permissions on a requester. For example, a customer opening a new bank account must present an appropriate identity credential before a credential for transacting on an account can be issued. We call this *dejure* delegation: by providing appropriate credentials the requester can acquire additional authorization; authorization that is not defacto.

The principle and/or application system making a delegation decision requires guidelines on when it is safe to do a dejure delegation. For the purposes of this paper we make the simplifying assumption that these guidelines can be encoded as additional reduction 'rules' specific to the application.

Example 1 A communications company with public key K_C provides private (priv) and business (bus) services. Private customers must provide a certificate signed by a recognized introduction agency K_I. Business customers must provide a credit-worthiness certificate from rating agency K_R.

Alice (K_A) and Bob (K_B) have certificates $\{\!|\, K_A, \mathsf{ok} \,|\!\}_{K_I}$ and $\{\!|\, K_B, \mathsf{ok} \,|\!\}_{K_R}$, respectively. For the purposes of authorization testing, permissions priv, bus and ok are considered disjoint (Figure 2(a)) during reduction.



(a) Authorization (b)Delegation

Fig. 2. Permission Orderings

Alice requests private customer authorization priv and presents $\{\!|\, K_A, \mathsf{ok} \,|\!\}_{K_I}$. For the purposes of dejure delegation, K_C is willing to trust K_I as an introducer. In effect, delegation of ok (by K_I) implies priv. For flexibility, delegation of ok (by K_R) implies bus and priv. This is characterized by providing additional rules specific to the application and modifying the permission ordering (Figure 2(b)).

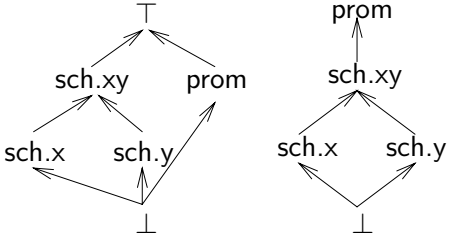
$$\frac{}{(\!|\, K_I, \mathsf{priv} \,|\!)_{K_C}} \qquad \frac{}{(\!|\, K_R, \mathsf{priv} \,|\!)_{K_C}} \qquad \frac{}{(\!|\, K_R, \mathsf{bus} \,|\!)_{K_C}}$$

Just as reduction can be used by an application to determine defacto delegation, it can also be used for dejure delegation. For example, by reduction, it is safe for K_C to write and sign $\{\!|\, K_B, \mathsf{bus} \,|\!\}_{K_C}$ since we have $(\!|\, K_R, \mathsf{bus} \,|\!)_{K_C}$, $\{\!|\, K_B, \mathsf{ok} \,|\!\}_{K_R}$ and $\mathsf{bus} \sqcap \mathsf{ok} = \mathsf{bus}$. △

The advantages of characterizing dejure delegation as reduction is that the trust management system remains responsible for trust calculations, not the application. It can answer two kinds of query. The first is the 'normal' authorization query based on the presented credentials and defacto permission ordering. The second type of query is for defacto or dejure delegation (Figure 1). In this case reduction also uses the additional credential rules and dejure permissions. We assume that the defacto permission orderings are a subset of the dejure orderings.

Example 2 WebCom [8] provides architectural support to servers scheduling mobile computation and services to client systems. KeyNote [3] provides trust management: server credentials prove authorization (to clients) to schedule certain services; client credentials prove (to servers) the right to receive certain services [5]. Servers may 'promote' [7] clients to become servers in their own right. A client must present credentials proving its authorization to be promoted; if satisfied, a server then delegates service scheduling authority to the promoted client. An arbitrary client does not have defacto authority to schedule; it must acquire it from an authorized server.

Servers and clients trust a common authority K_W for authority to schedule and to be promoted. For example, authority for scheduling services x and y is granted to K_S as $\{\!|\, K_S, \mathsf{sch.xy} \,|\!\}_{K_W}$. Authority to be promoted is granted to K_C as $\{\!|\, K_C, \mathsf{prom} \,|\!\}_{K_W}$. Figure 3(a) defines the permission ordering used when carrying out a defacto reduction.



(a) Authorization (b) Delegation

Fig. 3. Permission Orderings

A server is willing to delegate scheduling authority to a client only if the client presents a credential with the promotable permission. In this special case the

server is willing to recognize promotion as a delegation of scheduling authority. This is reflected by additional delegation rules that are used for determining dejure delegation, along with the existing credentials and the revised ordering in Figure 3(b).

$$\frac{}{(\!|\, K_W, \mathsf{sch.x} \,|\!)_{K_S}} \qquad \frac{}{(\!|\, K_W, \mathsf{sch.y} \,|\!)_{K_S}}$$

Thus, the dejure reduction to $(\!|\, K_C, \mathsf{sch.x} \,|\!)_{K_S}$ means that server K_S may promote K_C to schedule x services and write the corresponding certificate. △

4 Similarity and Delegation

A request for particular authorization may fail because it was not expressed in precisely the right way. For example, a request for a service might be rejected, despite the requester having dejure credentials for a similar service. Rather than rejecting such requests, it may be preferable to offer a credential with a similar authorization. For example, Alice likes the service that Bob's mobile phone credential provides, takes a copy, and requests a similar credential.

Supporting imprecision for retrieval systems has been considered in the literature on similarity-based retrieval for case-based reasoning systems [10]. For the purposes of this paper we will use a very simple notion of similarity. Define a similarity measure over permissions: $p \sim q$ returns a value indicating the degree of similarity between permissions p and q, ranging from 0 (not similar) to 1 (identical). When K_B requests permission p from K_A, then K_A may issue $\{\!|\, K_B, q \,|\!\}_{K_A}$ where q has the highest $p \sim q$ value such that $(\!|\, K_B, q \,|\!)_{K_A}$.

Example 3 Consider Example 1. An extended service permission (t, x) specifies that the quality of service for $t \in \{\mathsf{priv}, \mathsf{bus}\}$ is value $x \in [1, \mathsf{max}]$. The permission ordering from Figure 2(b) is extended such that $(t, x) \leq (u, y)$ iff $t = u$ and $x \leq y$. Suppose that the following dejure delegation rules are specified.

$$\frac{}{(\!|\, K_R, (\mathsf{bus}, x) \,|\!)_{K_C}} \quad [x \in \{5, 3\}]$$

$$\frac{}{(\!|\, K_R, (\mathsf{priv}, x) \,|\!)_{K_C}, (\!|\, K_I, (\mathsf{priv}, x) \,|\!)_{K_C}} \quad [x \in \{3, 1\}]$$

Reduction satisfies Bob's request for a credential with permission $(\mathsf{bus}, 3)$. Bob must phrase his request exactly: a request for precisely $(\mathsf{bus}, 1)$ will be rejected, even though a similar and possibly acceptable credential could be written, that is, $\{\!|\, K_B, (\mathsf{priv}, 1) \,|\!\}_{K_C}$.

Table 1 defines a possible similarity measure. Similarity is primarily based on the difference between the quality values. However, a weighting is used to give preference to the same service when the difference is small. Suppose that the maximum quality value is 5. Bob's request for permission $(\mathsf{bus}, 1)$, retrieves $(\mathsf{priv}, 1)$ with the highest similarity measure (0.8) when compared with $(\mathsf{bus}, 1)$.

Table 1. Permission Similarity Measure

$\sim$	ok	(bus, y)	(priv, y)
ok	1	0	0
(bus, x)	0	$\frac{\mathsf{max} - \lvert x-y \rvert}{\mathsf{max}}$	$0.8 \times \frac{\mathsf{max} - \lvert x-y \rvert}{\mathsf{max}}$
(priv, x)	0	$0.8 \times \frac{\mathsf{max} - \lvert x-y \rvert}{\mathsf{max}}$	$\frac{\mathsf{max} - \lvert x-y \rvert}{\mathsf{max}}$

A request for $(\mathsf{bus}, 2)$ retrieves $(\mathsf{bus}, 3)$ (measure 0.8) as the better match since both $(\mathsf{priv}, 1)$ and $(\mathsf{priv}, 3)$ gives measure 0.64 when compared with $(\mathsf{bus}, 2)$. $\triangle$

In the previous example the similarity measure is applied only to the reduction result. How the measure might be used during a reduction requires further study: direct application may have a cascading effect along delegation chains making the measure ineffective.

5 Conclusion

This paper considers how a trust management system might be used by an application to determine whether it is safe to do defacto or dejure delegation. When making a decision, similarity-based retrieval techniques provide for a degree of imprecision that may be useful when it is not clear beforehand precisely what form the delegation should take.

Delegated trust is treated as transitive, as exemplified by the reduction rule above. While it is straightforward to extend it to include a 'no-delegate' credential flag, it remains quite a rigid notion of trust. It would be interesting to consider how other notions and/or measures of trust might be applied to authorization credentials. For example, a PGP certificate [11] provides a degree of trust field, indicating the extent to which the signer trusts a key to vouch for the identity of others. Applying this notion to an authorization credential may require reduction to find multiple independent delegation chains from K_A to K_B before writing a new credential $\{\!|\, K_B, p \,|\!\}_{K_A}$. A variety of other measures for trust have been proposed [2,9,6]. One avenue worth pursuing would be to use the insurance certificate logic in [6] to form the basis of dejure rules that specify what kind, and how much, insurance is necessary before delegation should be done.

Acknowledgment. This work was supported in part by a grant from Enterprise Ireland National Software Directorate. The example on promotion and delegation in WebCom (Example 2) grew from useful discussions with colleagues John Morrison and Thomas Quillinan.

References

1. T. Aura and C Ellison. Privacy and accountability in certificate systems. Technical Report HUT-TCS-A61, Helsinki University of Technology, Laboratory for Theoretical Computer Science, 2000.
2. T. Beth, M. Borcherding, and H. Klein. Valuation of trust in open networks. In *European Symposium on Research in Computer Security*, LNCS 875. Springer Verlag, 1994.
3. M Blaze et al. The keynote trust-management system version 2. September 1999. Internet Request For Comments 2704.
4. C Ellison et al. Spki certificate theory. September 1999. Internet Request for Comments: 2693.
5. S.N. Foley, T.B. Quillinan, J.P. Morrison, D.A. Power, and J.J. Kennedy. Exploiting KeyNote in WebCom: Architecture neutral glue for trust management. In *Proceedings of The Fifth Nordic Workshop on Secure IT Systems*, Reykjavik, Iceland, Oct 2001.
6. J.K. Millen and R.N. Wright. Reasoning about trust and insurance in a public key infrastructure. In *Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000.
7. John P. Morrison and David A. Power. Master promotion and client redirection in the webcom system. In *PDPTA, Las Vegas USA*, 2000.
8. J.P. Morrison, D.A. Power, and J.J. Kennedy. A Condensed Graphs Engine to Drive Metacomputing. Proceedings of the international conference on parallel and distributed processing techniques and applications (PDPTA '99), Las Vagas, Nevada, June 28–July1, 1999.
9. M.K. Reiter and S.G. Stubblebine. Authentication metric analysis and design. *ACM Transactions on Information and System Security*, 2(2):138–158, 1999.
10. I Watson and F Marir. Case based reasoning review. *The Knowledge Engineering Review*, 9(4), 1994.
11. P. Zimmermann. *The Official PGP Users Guide*. MIT Press, 1995.

Trust Management and Whether to Delegate (Transcript of Discussion)

Simon N. Foley

Department of Computer Science,
University College, Cork, Ireland

A colleague in UCC, John Morrison, has been working on the development of an architecture for mobile code, WebCom[1], and in the last year or so we've been looking at using trust management systems to control the secure scheduling or dissemination of this mobile code. The main components to the architecture are masters and clients. A master schedules out operations to clients (the scheduling can be done in a push or in a pull fashion) and the clients use KeyNote to determine whether or not a master is authorised to schedule mobile code to a particular client. The code could be a COM object, for example, and the client wants to make sure that it's coming from a trustworthy source with trustworthy parameters. On the other side the master is also using KeyNote to determine whether clients are authorised to execute. So it's a very straightforward application of KeyNote.

One problem that we're beginning to look at is that in this architecture masters have the ability to promote clients. This means that a client which usually just does work that's given to it, or that it asks for, can also be promoted to be a master, which means that if it becomes overloaded, then it may in turn pass some of that work on or schedule that work on to other clients. A promoted client must also be given suitable credentials that prove that it has the authority to hand out mobile code to other clients for them to run.

Now what we want to do is somehow to use the trust management system to do checks to see whether or not it's safe to write a delegation certificate or credential. Normally we think of trust management systems as being used to ask, given some credentials and a requested action, is this person authorised to do that action. We are going to use our trust management system to assist in determining whether it's safe to delegate authorisation to a promoted master. So that's the context for this work.

So given this I started thinking a bit about this question of how should delegation get done. In all the papers I read, people always assumed the existence of these delegation certificates — there's not an awful lot written about how these certificates, credentials, come into being in the first place. So the outline for the talk is that we have some principal who wants a particular permission, they ask some other principal for this permission or for a credential which grants that permission, perhaps presents credentials to prove that they're entitled to get this permission and then they are given the credential.

[1] WebCom is a meta-computer that supports mobile code and parallel/distributed computing. It was developed by John Morrison and his students at University College, Cork.

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 158–165, 2002.

There are three kinds of delegation that I could think of. The first kind is what I call a *de facto* delegation, where the principal who's looking for the permission already has it implicitly. The next type is a *de jure* delegation, whereby the principal doesn't already have the permission, nor does it have it implicit in some authorisation chain, but it's entitled to get it from the other principal. And the last case, which I think is perhaps the most interesting of them, is *similar* delegation where the client says, I'd like a particular permission, the principal here says, well you're not authorised, or you haven't presented the right credentials for this permission, but I'll give you something that's similar or that approximates what you're looking for.

Authorisation by certificate reduction is a very simple idea. In the case of the WebCom system, the master wishes to schedule a particular operation to a client, sends the request, perhaps a collection of credentials are presented to prove that the master has authorisation, in this case it's a chain that chains back to K_A, and K_A uses the trust management system to ask, is it safe for me to execute this X, has the master got the right credential? The nice thing here is that the application system doesn't have to worry about how to make these trust calculations, it uses the trust management system.

So if we think of authorisation in terms of just plain delegation, then we have our certificate reduction. In this case the master says, I would like to have direct authorisation for this particular permission. Here the master doesn't have it direct from A, it has it via an intermediary B, so it sends the chain and K_A needs to ask, is it safe for me to write this particular credential, I use certificate reduction, I can do it, and I write the credential. So this is *de facto* delegation, you can either present the credentials or you can just make the authorisation request direct. So it doesn't confer any additional authorisation on the requester.

The *de jure* delegation is slightly different, here the master is saying, I wish to schedule this particular operation X, here's a credential. I've been authorised by K_W, the client has a local policy which basically states that they trust anything coming from K_W and anything that K_W delegates they are willing to accept, and so you can build up the trust chain as in the previous example and do the scheduling and the execution.

If we were going to do delegation by reduction, in the WebCom example the client says to the master, I want to become a master, I want to be promoted and so now the master needs to decide whether I am entitled to be promoted and if so, the master must write some credentials that allow me to do further scheduling. So in this case, the promoted client sends a credential saying, I have promote permission and so the master then needs to see if is it OK to grant. In this case I'm using a slightly different permission ordering. Normally if the client who had a promote authorisation just presented it with a schedule it wouldn't have authorisation to schedule, it needs to go through the promoter to say, here's my promote permission, my promote permission allows me to get a "schedule X Y" permission in the credential. So by providing appropriate credentials the requester can *de jure* acquire additional authorisations, authorisations that would not be *de facto*.

Bruce Christianson: If I have a permission to schedule X and Y and I just want to schedule X is that *de facto* or *de jure*?

Reply: If you have permission to schedule X and Y then that's *de facto*. There are two different orderings, one is the authorisation ordering, if you have "schedule X Y" you're allowed to do schedule X or schedule Y. The second one is basically giving you rules that you use to help to decide when to write certificates or to do delegations, so if somebody comes along and presents a "promote" you can then in turn write a "schedule X Y" which allows them do scheduling.

Bruce Christianson: But from the "schedule X Y" I don't have to go to a separate operation to get a "schedule X"?

Reply: No, if you think of the ordering here, these could be just sets of permissions. Since we're looking at mobility, if we looked at, say, a telecommunications company which provides two kinds of services, private and business services and somebody wants to sign up for a new service, well K_A in the telecommunications company trusts a credit agency K_R, public key K_R, to provide credit-worthy certificates for business customers and it trusts some introduction agency to introduce private business customers, so our local policy for the telecommunications company says that anything coming from K_R can delegate business, anything coming from the introduction agency can delegate private, and the introduction agency can also delegate business. So in this case a customer would present a certificate to get a business service, so a customer here says, I want business service, here's my credential coming from the rating agency saying I'm OK, this is my key. Then I use the delegation, you just need the trust management system to determine whether or not I should write this particular credential.

And basically this is enforcing a form of the principle of least privilege, because the customer can't just start using service, they have to first of all present the certificate and then get a credential that in turn allows them to use the particular service.

Now if people are going around asking for certificates or credentials, they might not express them in exactly the correct way. Maybe I like the particular services that (say) Virgil has on his mobile phone and I say, you know, I'd like services just like Virgil's please. So Virgil has a credential on his mobile phone which has various attributes set and I go to the telecommunications company and I say, well, here's some attributes, attributes of a service that I like, here are my own credentials that prove things like perhaps, you know, I'm a legitimate business, or whatever else, and I ask, please give me a credential that's similar to Virgil's credential. Now it's likely that maybe my characteristics don't quite match Virgil's, so that the telecommunications company says, sorry we can't give you this particular permission or a credential identical to this, but we could give you something similar.

So I started thinking that in the area of knowledge engineering people have similar problems to this. Suppose you're looking to buy an apartment in Cambridge, you provide various characteristics, you say I'm looking for an apartment that's less than £100,000, two bedrooms, and so forth, and you type in these de-

tails and the assistant comes back and gives you some recommendations, maybe not exactly that, it might offer you an apartment that's £110,000 and has half a bedroom, so it tries to approximate and give you something that's similar. So why can't we use similar techniques when we're dealing with credentials; people want some credential, maybe not the exact same, but would be happy with something that's similar.

The first scheme would be where — given a particular request for a permission — we return a credential for the least restrictive permission Y that can be *de jure* delegated to the requester. In other words, maybe you can't have the entire permission that's there but we'll give you a restricted subset of it. The other alternative is, given a request for permission X, return credential for permission Y with the highest value of X similar to Y that can be *de jure* delegated to the requester. Now this relationship here is a similarity measure and it attempts to quantify how similar two things are. In the area of knowledge engineering I understand the knowledge engineer sits down, looks at the attributes and devises these measures. In this example it's basically a function which takes two permissions, or attributes, and gives a value between 0 and 1, where 0 means they're not similar at all and 1 means that they're identical.

So we look again at our telephone example and let's extend the permission. Before we listed two kinds of permissions, one was private the other one was business, now we're going to add in an extra value which corresponds to the quality of service. The rules of my telecommunications company are that business accounts can have quality 5 and 3, and private accounts are offered a quality 3 and 1. So my local policy then is the following credentials, I'm saying that anything coming from the ratings agencies, rating credit worthiness of businesses, can establish permissions of business x where x is 5 and 3 while for private customers the x is 3 and 1 and then I need to construct my similarity measure, and without going through the details of this, the essence of what's happening here is that I'm going to use a weighting to give preference to the same service when the difference is small.

Suppose we have a customer who'd like a low quality business service and so puts in a request for a business service quality 1. I've got my rating from my credit rating agency so K_A knows that it can write the appropriate credential, but it can't write a credential for business 1 because such permissions don't exist, so it needs to search for the best credential using the similarity measure. In this case the similarity between business and private account of quality 1 is .8, while the similarity between business 1 and business 3 is .6, so .8 is the best so we give back a private credential for quality 1. If the customer would like a slightly better quality business service, say, for example, 2, then the similarity measures are slightly different, so the customer ends up with a business service, weighted up with quality 3.

So the intention here is to use our trust management system to help us make judgements on whether or not we should delegate. We're used to the idea of using them to tell us whether or not a request is authorised; I think it can also be used to help us to make judgements about delegation. There are three simple forms

of delegation: the *de facto* delegation is permissions you already have, you just have to present the entire chain, the *de jure* delegation is permissions that you're entitled to but you don't necessarily hold at the moment, and *similar* delegation is the case where we can't delegate this particular requested permission to you but we give you something similar.

Now, should the generation of the similarity based delegation credentials be automatic, or should the requestor first be offered any alternatives? Again, I look at my friend's mobile phone, I like the characteristics from talking to my friend, and I say I'd like a credential just like this person's. Maybe I'd supply a measure and I'd say give me the nearest credential to this, within say .8 or .9, if its less than that I'm not interested. Should the telephone company offer me some credentials I can choose from, or should it just give me the best one it has based on its own similarity measure. Perhaps I'm not interested in the low level details, all I'm working on is the experience of my friend and I'm happy enough to get something that has similar characteristics, and of course, assuming I'm willing to trust the similarity measure that the telephone company was using. Automatic delegation might be used for embedded or mobile devices that can only support limited user interaction.

Another thing I've been thinking is whether or not the similarity used can provide some sort of guide when writing credentials, could the similarity measure be somehow encoded within the credential itself, so that it's not something that's external to it.

And then finally, in knowledge engineering and for similarity measures I think they use things like nearest neighbour algorithms for graphs to do something that will be quite similar to similarity reductions, so I'm hoping that they will come up with reasonably efficient algorithms to find the best credential.

Bruce Christianson: I think the issue about trusting the similarity measure is a really important thing that you've bought into focus. One of the reasons for wanting *de jure* delegation is to protect myself, I don't want to have privileges when I don't want to use them, *i.e.*, I want to be able to prove that I didn't have the power to open the safe except during the periods when I do actually need to be able to open the safe. So I'm quite happy with the least privilege argument that says, well, just give me the least privilege I need to open the safe, take out this one exam paper and shut the safe again, and I trust you to give me enough to do that and nothing that I don't need to do that. But with similarity it's a much more dangerous game if there's no user confirmation, you now have the possibility of a dispute about whether somebody was given a power that they didn't ask for or whether they had in fact requested it.

Reply: So should the provider also be defining the similarity measure, or should the customer get the similarity measure from the Consumers Association or some body that we're all willing to trust?

Bruce Christianson: Third party provision of similarity, yes, a similarity service is in a sense a meta-service. But the point is, you are delegating to that third party the power to sign off on a particular aspect of delegation on your

behalf. That's the crucial step that most accounts don't bring into the foreground at all, and your approach does.

Reply: Should you read the small print this long, or do I just accept the value of .9?

Bruce Christianson: Well certainly you don't have to read the small print yourself, but if you don't you have to have given somebody else the right, duty, privilege, authority to read it on your behalf for you and sign as your agent. Otherwise there isn't a contract.

Pekka Nikander: Do you use some kind of structure for your permissions or whatever you're delegating? In the examples you had some kind of lattice.

Reply: Yes, the lattice is just the simplest way I could represent the delegation graphs rather than sets of permissions on arcs.

Pekka Nikander: But I'm just trying to understand what are good motivations for limiting how little you can get. You explicitly say I don't give the small permission but I'm willing to give this big one, and now I'm trying to understand why you would like to exclude those small sets if you are willing to give a bigger one. Why would your similarity policy never say like in your example, you have excluded the case of having a business account with very low quality of service? I'm not quite sure whether there are any situations where you would like to set your permission policy so that you don't allow that kind of permission.

Reply: Maybe as a telephone company I decide that businesses can't have low quality and if they want low quality they have to a private account.

Tuomas Aura: A lot of businesses work like that, if you want to improve one part of your service. Let's say you want to buy flight tickets, and you want the use of the business lounge, then you must pay for the full business class service for everything.

Pekka Nikander: I must pay for it, but if I specially request that I want to fly in the business class but I don't want to go to the lounges but I'm willing to pay the same price, why don't they give me just a credential saying, OK.

Tuomas Aura: If you don't use this privilege, how does it hurt you to have it?

Pekka Nikander: Because I'm protecting myself, maybe I might meet somebody at the business lounge that I want to say that I wasn't there, I couldn't get in there.

Bruce Christianson: Yes, you want to be able to prove that you could not have been at the scene of the crime. It's always very dangerous when people give you permissions that you don't need to have because it means when that permission is misused by someone, you are one of the people who could have misused it.

Tuomas Aura: One problem with this distributed system certificate is you cannot prove that I didn't have this certificate.

Bruce Christianson: Well it depends on the protocol for issuing a certificate; if a certificate is only valid if I sign it when I receive it, then that assertion fails.

Pekka Nikander: Right, as a specific example, you can have this upgrade permission. Then whenever you want to be upgraded you get a very short time-valued certificate which is valid only for a couple of seconds, and then the party which is issuing those certificates has a log saying who has acquired that kind of certificate, so you are relying on a secure log to show whether you had it or not.

William Harbison: Bruce, there is a slight modification that you could introduce which is you only sign it when you exercise it.

Bruce Christianson: Yes, that's another alternative and similarly as Pekka says, you can have an interaction with a log at various points. There are all sorts of mechanisms you can use. The principle is that you don't get privileges you haven't asked for and the question, which is good one, is are there reasons, other than marketing reasons, for not having a least upper bound principle in what sets of credentials you can acquire.

Pekka Nikander: And that wasn't the reason why I asked about the structure, it's because there might be other kinds of structures that are not lattices and so you don't have this.

Bruce Christianson: Yes, exactly right.

Tuomas Aura: Well it must be difficult actually, if your request is something like give the least permission required for this action, ...

Bruce Christianson: ... there might not be a least set, that's exactly right. There might be sets that are minimal but there might be more than one such set and then the question is, well, which one is most similar to what I want.

Tuomas Aura: And given the granularity of this access control then they might not even be really what you want, they might have some extra.

Bruce Christianson: Oh sure, but the point is, even in that case I might have a firm preference about which set I would prefer, based on my threat model. But now how is that reflected in the credentials that I actually get written?

Pekka Nikander: Does the situation change if we consider personal certificates, personal delegation? I think that might bring something quite useful.

Tuomas Aura: Maybe you could say that you don't want this right.

Bruce Christianson: Well this kind of happens when you have someone who's declared to not be in a position of trust, and that means that if the employer ever gives them the key to the safe the employer's liable for any loss. There are several cases like this where banks have failed to follow procedures and as a result of that some cleaner has been presented with an open safe. The courts have usually decided that in those cases the cleaner, when they get back from their spending spree and turn themselves in, they have to hand back what they still have but there's no liability on them to repay any of the other money because they were not in a position of trust and so therefore the bank should not have given them access.

William Harbison: So that's what they mean by being taken to the cleaners.

Bruce Christianson: This is why your employer will always try to say that you're in a position of trust, and you think, oh that's nice, they trust me but of course what they're really saying is, you're liable.

Tuomas Aura: One problem can be that if the access rights form a lattice and this lattice is assumed to be generally known, then you might at one time ask for this right, another time you ask for that right, and it's pretty difficult to calculate that you don't want the right that you automatically get when you have this phone.

Bruce Christianson: How much for the newspaper if I only want to read the sports page? It's a very similar problem to this bundling problem that you get with telecommunication services where you say, I want channel X, and they say, oh well you must upgrade to package Y, and you say, hang on ...

Pekka Nikander: This is just a crazy idea but what if you don't use the certificates for permissions but for certifying something else like inter-personal trust?

Bruce Christianson: Well, because Simon has moved the issue of trust management back to the level before where we usually consider it, this is a good suggestion, because you could do this. You're not simply looking at a presented chain and saying do I allow access or not, the permission you're looking for is permission to write a credential.

You Can't Take It with You (Transcript of Discussion)

Mark Lomas

A common problem is that you want to be able to have access to your data, but you aren't sure where you're going to be at the time you want to access it. There are several different solutions to that. One is to carry all of your data around with you. In fact I do, it's that laptop, chained down over there. But I don't want to carry it absolutely everywhere, because it's a bit heavy, it's a nuisance, so in practice you want to use equipment at the location that you go to. On the other hand, if you do that then you're revealing information about yourself: the data itself, and — if the system isn't designed properly — you may also reveal authentication information that allows somebody else to masquerade as you later.

So a very long time ago, and I don't remember the exact date, a company called Security Dynamics (subsequently renamed RSA Security) came up with a thing called SecurID. These [holds up tokens] are more recent versions, this one is about credit card size, and this one is a key fob, the technology is essentially the same technology that's been in use for literally decades. It's basically a little micro-controller, there's a clock, and it knows a secret in common with an authentication server. And about once every sixty seconds this six digit number on the front display changes. All it's doing is computing some hash function of the time and the secret that it knows. The server at the other end can then validate it, and the authentication protocol is trivial but actually rather weak. You send a PIN — the purpose of the PIN is in case somebody has picked up your card, it makes it a little harder for them to use it — followed by the value that appears on the screen, which is the hash function applied to time and secret. The reason I say this isn't particularly well designed is, if I'm using your keyboard and you have the opportunity to remember my PIN, then you can also play about with the transactions that I've authenticated, because this doesn't set up any keying information, all it does is authenticate the start of the session.

There's a variant of this which looks very similar to the previous card. In fact if you pull one apart you find it's identical to the previous card but with a little keyboard on it, so you can type a PIN into the card rather than into the terminal that's in front of you. This is slightly more secure, it computes a function of the time, the PIN, and the secret, and then you send that value to the authentication server. So again, it can validate that you know the PIN and that you've got the card and therefore it assumes that you are you.

There are several reasons why you might want to use this technology. One is that you don't necessarily trust the keyboard that you're using not to keep authentication information. This version gives you a bit of a gain in security if you don't trust the keyboard. On the other hand, if the machine has been compromised it still knows all the transactions you subsequently do and it may

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 166–169, 2002.

actually interfere with those transactions. So this has some uses but it's far from perfect.

I'd like to look at a particular property of this that people haven't generally discussed. It shares a symmetric key with the server, that's the way the protocol works, except that in real life I don't talk to only one server. I want to be able to authenticate myself in several different domains, and if I use this the way that RSA security suggest I use it, I have to get a different security card for each different organisation I want to communicate with. So it's a bit of a nuisance. If I were to share that secret between servers — so that I only have one card — then any one server knowing my secret can reproduce exactly what the card would have done done, and therefore they can masquerade as me. So it's definitely not a good idea to use the same authentication token for several different services.

But as I said, if servers don't share secrets then clients need to carry several cards. You might think that this is a trivial matter, but I've spoken to equity traders who get issued with these cards to authenticate themselves for doing high value trading transactions, and the counterpart bank will give them a SecurID card and say, this is to authenticate you to our service. And the problem is, they end up with a drawer full of SecurID cards and no convenient way of telling between them, they're a real nuisance.

It would be nice to replace the symmetric key that's in the card with an asymmetric key, so that I could safely give my public key to servers that I want to authenticate to but which shouldn't be able to masquerade as me. Unfortunately the way that the thing works makes this actually very difficult. I'm assuming that there is a PIN pad on the card here. If the function value you compute is too long when you're using symmetric keys, then you can make a trade off. You can say, for instance, my users don't like typing those six digits, I'm going to type five of them, and I'm going to tell the server to check the five and ignore the other one. That will still authenticate, but there is a higher probability, there's tenfold higher chance, that somebody can masquerade as me by guessing the current value on the card. But I can work out whether that trade off is appropriate. You might, for instance, look at the value of the transaction you are authenticating: very high value, I need as much information as possible to authenticate; low value, I'll tolerate a small authenticator. For symmetric keys I can just discard digits, or there are various other ways I can shorten the amount of information the user has to type. But if I use a digital signature instead, if I'm using a public/private key system and I compute the signature of the time and PIN with the private key, then that value is almost certainly too long to type. If I'm using an RSA key with 1024 bits, my signature is going to be in the order of 1024 bits and most users don't want to type that amount of information just to authenticate themselves.

Bruce Christianson: So why not just use a 32 bit modulus? [laughter]. Seriously, your point is that to verify a signature the server needs the whole signature.

Wenbo Mao: Can we use radio or infrared technology to exchange data between the secret in the card and the terminal?

Reply: If you have an extra communication channel with the card then it's much easier because you can just send the information. One of the other advantages I forgot to state is that use of a SecurID card doesn't presuppose that there is a particular type of technology in the terminal. As long as I've got access to a keyboard, I can use a SecurID card. In fact I don't even need a keyboard, they're sometimes used when you telephone a company and say, I'm Mark Lomas, and they'll say, well if you're Mark Lomas you can tell us what's currently displayed on your card. You then authenticate where there is no computer involved in the communication part at all, absolutely no technology apart from the telephone, nothing but you that actually understands how to talk to the card. So it would be convenient in those circumstances to be able to send the signature.

Think about it this way, you want to be able to read out a digital signature down a telephone line. Bruce said that you need the whole of the signature, but you can actually discard a small number of digits. I might for instance type most of this signature, but my fingers are getting tired by the end, I'll drop off the last two digits. The server at the other end can then search and work out whether there was a plausible signature. But you can't discard huge amounts of the information, because the computational effort of the server to work out what you actually meant becomes too high exponentially fast, and as I said, public key signatures tend to be much too long to type.

Tuomas Aura: Why do you want to use public keys? Why not just have several secret keys on the card, since you have the key pad on the card it's easy to choose which one of them you want to use.

Bruce Christianson: One difficulty is that the person who gave you the card doesn't want to give you their secret for you to put into somebody else's card. The whole point of them giving you their hardware is that they can subsequently ask for it back and check that you haven't tampered with it.

Matt Blaze: But using a public key wouldn't solve that problem either.

Tuomas Aura: If they trust that this card has not been tampered with and that it is a real product by this company, then you can generate the secret on the card. And so when you go to register to the server, you press a certain code on the keypad and the card generates the number.

Reply: There's still a problem with generating a *shared* secret in that way. If I share a secret, then I've still got the possibility of getting into an argument with a server as to whether this audit trail entry really represented me giving it to him, or did he create it himself, because he had the system secret. So I really would prefer to have an asymmetric system if I've got the chance.

Here's a suggestion to contemplate. I should emphasise this suggestion suffers from one of the weaknesses of SecurID, which is it only does the authentication, it doesn't take any account of what it is you're going to authenticate, the subsequent transaction that the authentication was supposed to support. What if we were to pre-compute a set of signatures for a number of plausible times? The clock ticks, in effect, only once a minute, so that might not be a huge number of signatures if you plan it right. So I pre-compute the signature of the time and

the PIN under the private key for a large number of possible times. If I handed all of that information to the server beforehand, then it can authenticate me, but again this suffers from essentially similar problems to symmetric keys: in a service for clients, you never know whether a particular signature was actually sent by the client or it was merely part of the pre-computed set.

So let's try a slight variant on this. I've already said that if I'm using asymmetric systems then I can only discard a small amount of data. So suppose I pre-compute a set of signatures, and I split each signature into three parts. You can regard the discarding we did earlier as splitting it into two parts and then throwing one part away. I'm going to give a list of the values of the first component, S1, from the signature to the server. So that's part of the state of the server, which it can use it to help speed up the authentication as I'll explain in a little while. At set-up time you just discard S2 and S3. When I actually come to authenticate myself I can reconstruct the signature, because I've got the private key, and I send S2, the second component, to the server. The reason I'm not sending all of it is that I've still got the constraint that the user doesn't like typing very much. So I work out how much a user is willing to tolerate typing and that constrains the length of S2. And then to verify, the server does the same sort of search as before to find S3, but because it already knows part of the data it's doing the search on, there's the possibility that you can again trade off how much state you're having here against the amount of computation that you're willing for this server to have to do to authenticate.

So, questions that I've got. One is, is this practical? If you're using things like RSA keys, then I think it's not practical, even with this technique, because the length of data that you actually have to send is still too long for people to tolerate. It's probably on the edge of practical if you use something like an elliptic curve system, where the signature length is going to be closer to 200 bits than 2000 bits. So it may be practical in certain circumstances, although I'm not saying it's an ideal system.

Secondly, it suffers from a significant problem, which is what I'd actually like people to think about: it does not authenticate the data that's sent subsequently, after you're authenticated. It would be really nice to be able to say, not Mark Lomas started this session, but Mark Lomas started this session and agreed to this set of transactions. Is there some way in which all of that data passed over can in some way be processed by my authentication token?

The third question. Technology has advanced quite a lot since this SecurID came out. Do things like mobile phones make all of these ideas obsolete, or are they still useful? I should point out that RSA Security have actually embedded SecurID technology in telephones from both Ericsson and Nokia which are currently available, so they believe that some people want it.

Markus Kuhn: The SIM card on a mobile phone is an 8 bit micro-controller with (today) 8Kb of EPROM, it can do a signature in five seconds or so.

Roger Needham: Yes, one of our people has implemented an HTTP server in a SIM card.

Protocols Using Keys from Faulty Data

David Wheeler

Computer Laboratory, University of Cambridge, England

Abstract. Some methods are derived for making faultless keys from faulty data such as iris codes or other analog measurements.

1 Introduction

We consider methods for making faultless cryptographic keys given some data such as an iris code where there are changes in the iris coding for each sample. The model is a data base B with one copy of A's iris code, and A who uses a different sample of her iris code. The number of error bits is perhaps 1-30%. Thus source and destination have different samples of the same iris code, and a new sample is taken for each key generation.

We usually assume the probabilities are the same for each bit and there is no correlation among different bits. Weaker assumptions will cause the implementation to be more difficult if advantage is taken of them. We also assume that the bits are synchronised so there are no missing or extra bits.

2 Basic Method

C is A's error prone code; D is B's archive copy; E,F are error correcting bits.

A sends E based on C.
B receives E and forms F from D and uses E XOR F with D to form C.

We now have a mutual key C which may be used to encrypt messages.

A can send with E a message M and/or a session key K, encrypted under C, making a one shot protocol. Safe communication both ways is now possible. We should choose E to minimise information leakage.

If there are about e_s errors between different samples of the same source but e_d errors between samples of different sources then we would choose E so that the errors from the same source could be corrected but not from different sources. The threshold would lie between e_s and e_d and be chosen to balance the chance of false keys and false rejections according to requirements.

Also, if there are known redundancies in C they may be and should be exploited to reduce leakage.

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 170–179, 2002.

2.1 Selection Variant

Instead of using error correction we can use key bit selection. The simple case involves a number of messages.

A and B send the parities of groups of iris code bits to each other. They simply eliminate the groups with different parities, and the process is repeated with the remaining bits. The groups are chosen so that, as far as possible members of a group are distributed among different groups at other levels, i.e. as orthogonal as possible.

More and more groups get eliminated but the probability of errors decreases rapidly. For different (wrong) codes the number of groups is about halved at each stage.

Later stages where few parity bits are involved can be combined, at little cost, thus reducing the number of messages. A verification hash can be sent each time so that the process can be stopped when no errors remain. See tables 4 and 7 in the Appendices.

2.2 Direct Key Generation

A sends 2048 bits to B, say 128 groups of 16 elements of her code, each group XOR-ed with one bit of a random key of 128 bits generated by A. A hash of the key is also generated and sent, together with a few error correcting bits (ref 1).

B gets the data, and for each group has 16 samples of the key bit generated by using his code bits. He uses the majority to obtain the key bit. The few errors remaining are corrected by the error correcting code or else by searching using the hash function.

Making reasonable random assumptions, Eve can eventually learn each 16 bit group and its complement each with 50% chance. Thus the key bit remains secret although the difference between successive keys may become known. To avoid related key attacks it is advisable to use the key bits to generate a working key by a suitable one way hash.

Table for frequencies of number of errors for 16/8 element sum, each element having a chance of error 0.1 to 0.25. The column headed inf gives the information, as a percentage of a bit, per original element.

no	err%	inf/cts	0	1	2	3	4	5	6	7	8	9	10
			....number of errors per 1000 samples.....										
16	10	54	46	185	329	274	142	51	13	2			
16	15	40	60	74	209	277	228	131	55	17	4		
16	20	28	72	28	112	211	246	200	120	55	19	5	1
16	25	19	10	53	133	207	225	180	110	52	19	5	1
8	10	54	46	430	382	148	33	4					
8	15	40	60	272	384	237	83	18	2				
8	20	28	72	167	335	293	146	45	9	1			
8	25	19	81	100	266	311	207	86	23	3			

For 128 blocks of 16 and an error chance of 1/4, 2.5 sums give no information, while 7.3 have an error chance of 1/9 etc. Thus the error correction is simple.

Note that although the groups combine 16 elements the original data had only about 1/5 bit per element.

2.3 Transformation Methods

We code indirectly using a transform. The Hadamard transform seems best as it is implemented by XORing with a set of vectors and counting ones in the result, and is reversible so no information is lost.

This amounts to projecting C onto a set of orthogonal vectors. The mean values are subtracted to give a set of signed offsets. The signs of the offsets will be taken as the key bits.

The Hadamard transform gives N separate offsets and the vectors of the transform are mutually orthogonal. This will give N offsets which may be considered random of variance N. We expect the use of pseudo random vectors is OK, but the Hadamard vectors are as easy to generate.

Enough offsets have to be used so that the errors can be corrected. We can use every offset and a suitable decoding technique. In this case the number of error correction bits is large, and we make an unnecessarily large key.

The information of an offset near zero is very small as it takes very few errors to have a 50% chance of changing the sign of the offset. When the offset is large a few errors are unlikely to change the offset sign.

Thus it is better to reject small offsets, and only use the larger and less error prone ones. In this case the selection has to be transmitted.

A very simple way is to send say an m bit code indicating the best of the following 2^m offsets. A few error correcting bits will still need to be sent, unless the probability of an error has been made small enough. The information leakage is also reduced by this method.

This is further improved using the correlation of large offsets in C and error prone D. A salt should be used each time so that a different Hadamard transform is used each time.

2.4 Leakage of the Key

The basic method leaks the key so that it can be used safely very few times. As we are unlikely to need 2048 key bits, we can select and use many fewer bits thus reducing the leakage. The leakage may be large as each parity bit has a corresponding bit equation and when enough equations are present they can be solved. Each use allows Eve to generate additional parity equations. A minor help is the fact that as some digits are wrong the equations are not consistent.

The transformation method leaks which offsets are used but not their signs, so it is difficult for Eve. She knows which vectors are referenced but not if they are nearer or farther, so it is difficult for her to use the information.

If error correction is used the parity equations correct the signs, so that we obtain sets of equations of the sign bits. However if a salt is used the further sets of equations are distinct and the accumulation of data little help.

3 A Method Using a Two Message Protocol

A can select the N largest offsets, then A can send their locations to B. B can then select his N/2 largest offsets in her list and send their locations to A by N bits. Heuristically, about one half the offsets are improved, so we try to select that half. This makes a key of length N/2.

These bits are very reliable and need little by the way of error correction.

3.1 Precautions

A should probably include a salt or time so that the generated keys are always different. If possible some non linearity should be included so that the solution of non linear equations is needed to find the key. The transformation methods seem to do this quite well.

The hash of the key used for verification should have enough bits and be invertible only by search.

A second hash may be used to avoid related key attacks, by making unrelated keys.

3.2 Error Correction Methods

Although we may use any standard method of error correction such as turbo coding, BCH, etc. there is one major difference in the usage. We do not send the data but only the error correction bits. Thus non systematic methods where these two types of bits are scrambled together, are non optimal.

Also the error correction bits do not need correcting (or are independently corrected for transmission errors) so that fewer are needed. Information theory indicates if up to one in nine are in error, an error prone code of N bits would need at least N error correction bits if the latter were equally error prone, but at least N/2 if they were error free.

Also we can modify the selection process so that the error correction bits are made zero (or mutually known) and do not have to be explicitly transmitted.

Thus if we need K key bits, KC correction bits and we use a selection method with a linear error correction method such as BCH: Then we generate KC bits and for each of these we choose the original choice and the best choice of opposite sign. Then by solving KC linear equations we can force the error correction bits to zero. Although any such choice will do, we can try to optimise by ensuring the resulting offsets are as large as possible.

A neater way is to make the error correction bits and use them to 'correct' C so they become zero. The changes in selection are made by choosing the best offset of opposite sign in the same group. Some extra minor complications ensue

if the correction needs more changes, or the group does not have an offset of opposite sign.

The best selection process is unclear. Instead of say, choosing the best of 16 and getting 128 key bits, we can select the best 128, but we must then send about 5.4 bits per selection rather than 4. However, it is a better selection and the chance of errors reduced.

We can make use of the verification hash to correct some errors by simple 'change and test' search. It is useful to combine both methods so that both search and error correction are used only occasionly when the simple error correction fails.

3.3 Applications

These are likely to be rather specialised. We could for example use a voice password (ref 2), although the synchronisation requires care. The error prone code can readily be used for authentication directly. However, our methods allow a key generation, and will leak less.

Although we can use these methods to reduce the size of a data base, the search speed may be reduced due to the extra calculations.

3.4 Weaknesses of the Methods

These are subject to the usual attacks. Most simple forms allow key leakage, but perhaps less than the use of a PIN.

A certain amount of work has to be done at both ends. The source usually does more work than the destination.

4 Examples

4.1 One Message Protocol

We generate a 64 bit key assuming the statistics of table 3. The error rate is 3 per 1000, so the chance of an error in 64 is about 3/16

- Take a sample and transform using a Hadamard transform selected by a salt, PIN, time etc.
- Select the 64 best offsets from the groups of size 32.
- Split the 64 into groups of 21,21,22 and use Golay codes capable of correcting up to 3 errors in each set.
- Send the salt, selection bits 64x5 and the correction bits 11x3 plus a verification hash of at least 64 bits.

The chance of a failure is about $3 * (3/1000)^4 * 21 * 20 * 19 * 18/4! = 1/500000$. In this case the protocol is repeated, possibly in the reverse direction, or a hash search is used.

4.2 Two Message Protocol

- A sends a salt, locations of the 128 largest offsets to B, say 64 + 128x5.5 = 768 bits.
- B finds its own 64 largest offsets in A's selection. It sends its own selection, using 128 bits and perhaps error correction, and hash verification.

B may also send an encoded message.

From the table 5 the average location of the first error is 127 and 84 for the first two categories. We have a single case for the marginal category 2, where the location is 44.

4.3 Multi-message Protocol

A and B send each other the parities of groups of iris code bits. The groups with different parities are eliminated otherwise one bit of the group is eliminated. The choice of group size affects leakage, length of message etc. The table 4 give results for the sequence g=3,4,6,9,13

The group size sequence is non optimum. We can choose at stages other than the first to make the group size depend on the current error rate derived from the size reduction.

We can also take into account the number of key bits required, to reduce the number of parity bits exchanged. Hash verification can end the exchange and perhaps remove an error by search.

5 Conclusions

In some circumstances we can use analogue or error prone data to derive common keys. As work has to be done at both ends, these methods do not reduce the search time for a nearest match.

Acknowledgements. My thanks to Bruce Christianson and Mike Roe who clarified and contributed to this paper.

References

1. Ari Juels and Martin Wattenberg. A Fuzzy Commitment Scheme. pp 28–36 in 6th ACM Conference on Computer and Communications Security. 1999.
2. Fabian Monrose, Michael K. Reiter, Qi Li and Suzanne Wetzel. Cryptographic Key Generation from Voice. in Proc 2001 IEEE Symposium on Security and Privacy. May 2001.

Appendices

Calculation Results

Some 2150 iris codes were used. Each was compared with the preceding one using the standard counts. The error count was made using masks. The results were split into ten categories E0 to E9 corresponding to the first figure of the error count per 1000.

The calculations were repeated using 9 different Hadamard codes. The column headed no. gives the number of results averaged. For non Hadamard dependent data the effective number of samples is no./10

The 2mask gives % of ones for both masks while mask gives % for this code. err/1000 gives the proportion of errors and category Ex gives the first digit x of this proportion. Thus the results are sorted into at most ten categories.

Table 1. A's transform is sorted and scanned in order largest (best) offsets first. We compare the corresponding bits of the previous transform code and the scan location of the rth error is given below.

Table 1

no.	cat	err	2mask	mask	r=1	2	4	8	16	32
6160	E0	18	54	63	281	360	452	563	702	877
130	E1	129	45	58	29	51	82	129	201	314
10	E2	204	46	57	14	27	49	76	122	209
30	E3	364	30	45	3.5	6.7	12	25	49	95
8030	E4	471	39	58	2.3	4.4	8.8	18	35	69
7070	E5	526	40	58	1.8	3.6	7.3	15	30	60
50	E6	611	34	52	1.6	3.3	6.4	13	25	50

Table 2. Table 2 has A and B each select 16, 32 etc best offsets from their sorted offset lists. Those in both lists are then classified as success or failure.

Table 2

		16		32		64		128		256	
no.	cat	s	f	s	f	s	f	s	f	s	f
6160	E0	6.0	0.0	13	0.0	29	0.0	65	0.0	144	0.0
130	E1	1.9	0.0	5.0	0.0	12	0.0	32	0.0	80	0.2
10	E2	1.1	0.0	3.0	0.0	8.2	0.0	22	0.0	59	1.0
30	E3	0.1	0.0	0.6	0.0	2.5	0.2	8.6	0.9	29	5.9
8030	E4	0.1	0.0	0.3	0.2	1.3	0.8	4.8	3.3	19	14
7070	E5	0.0	0.1	0.2	0.3	0.8	1.3	3.3	4.8	14	18
50	E6	0.2	0.2	0.3	0.6	0.5	2.0	1.8	8.2	8.2	26

Table 3. A takes the best offset from groups and sends their locations to B. The proportional error count is given as before.

Table 3

		group size						
no.	cat	1	2	4	8	16	32	64
6160	E0	184	99	44	18	7.3	3.2	1.4
130	E1	306	236	171	119	84	58	42
10	E2	349	296	244	202	171	143	115
30	E3	443	420	397	367	343	325	312
8030	E4	487	482	477	473	468	464	459
7070	E5	511	516	521	526	531	535	539
50	E6	544	561	578	593	614	620	641

Table 4. Table 4 gives results applicable to the multi-message protocol, using parities of groups of digits. The remaining digits are permuted (randomly ?) between each parity exchange. e gives the error count and sz gives the new size. When a group is retained one digit is rejected in order to increase Eve's workload.

Table 4

parity group size			3		4		6		9		13	
no.	cat	mask	msk12	e	sz	e	sz	e	sz	e	sz	e	sz
6160	E0	1301	1124	20	717	1.9	533	0.1	444	0	395	0	365
130	E1	1207	926	120	465	25	287	3.4	224	0.2	198	0	183
10	E2	1177	957	196	455	48	237	9.1	157	0.9	133	0	123
30	E3	922	619	220	218	62	82	20	36	8.4	17	5.2	9.6
8030	E4	1193	815	385	276	122	103	46	44	19	20	9.6	10
7070	E5	1201	819	431	271	135	102	51	43	21	19	11	10
50	E6	1070	715	436	235	135	86	49	36	21	17	11	11

Notes

- The source of inaccurate data can come from a variety of places such as physical measurements. Using the deviation from the average would give more useful results in most cases.
- It is an advantage that the generated key may change each time.
- In some applications quite a small key may yield more security than say a PIN.

Table 5. Table 5 gives the location of the first and second errors where A has sent the location of his largest offsets and B sorts this list his largest first. The average location of the first and second errors is given for various list lengths. They are from 0 to N-1 with N indicating no error

		Table 5											
		group size											
no.	cat	16		32		64		128		256		512	
6160	E0	16	16	32	32	64	64	127	128	246	251	439	464
130	E1	15	16	29	31	53	59	84	99	111	141	107	158
10	E2	12	16	22	27	35	46	44	57	52	74	47	60
30	E3	4.0	6.7	6.3	9.7	6.9	11	8.3	14	8.4	15	5.7	11
8030	E4	1.4	3.6	1.4	3.7	1.4	3.8	1.4	3.8	1.4	3.8	1.3	3.6
7070	E5	0.6	2.4	0.6	2.3	0.6	2.2	0.6	2.2	0.6	2.2	0.6	2.3
50	E6	0.3	1.6	0.3	1.4	0.3	1.4	0.3	1.5	0.4	1.7	0.4	1.8

Table 6. Table 6 tests for filtering possibilities. A and B merely select the largest offset in each group and we count the errors. For M groups we expect M/2 for different irises and M/4+closeness for same iris. The filtering seems poor.

		Table 6		
no.	cat	group size 128	64	32
		error counts		
6160	E0	35	19	10
130	E1	50	26	14
10	E2	57	29	14
30	E3	60	31	15
8030	E4	63	32	16
7070	E5	65	32	16
50	E6	67	33	16

Table 7. In Table 7, pairs with different parities are deleted, otherwise second element rejected. sz gives number remaining and e the errors count.

		Table 7											
no.	cat	msk12	e	sz	e	sz	e	sz	e	sz	e	sz	e	sz
6160	E0	1124	20	544	0.9	271	0	135	0	67	0	33	0	16
130	E1	926	120	371	14	174	1.2	86	0.2	42	0	21	0	10
10	E2	957	196	361	39	151	5.0	70	0	35	0	17	0	8.0
30	E3	619	220	171	41	62	8.3	26	2.0	11	0	5.3	0	2.7
8030	E4	815	385	222	100	62	26	19	6.7	5.7	1.5	1.6	0.2	0.4
7070	E5	819	431	223	122	62	36	19	12	5.5	4.1	1.5	1.3	0.4
50	E6	715	436	203	141	63	51	23	21	11	10	4.4	4.4	1.8

- In the multi message protocol we have randomised the bits between each parity exchange in order make each group be spread among other at the next parity exchange. Either A or B can falsify parities to confuse Eve.
- A confirming hash should be used to verify the key. It should be good enough not to leak the key.
- The hash receiver can also use the hash for error correction by search. This can be used to extend the error correction power or even do all the correction.

Generation of the Hadamard Matrix

A Hadamard matrix of order 4 is

```
0 0 0 0
0 1 0 1
1 1 0 0
1 0 0 1
```

The discrepancies between any 2 rows or columns are exactly 4/2. Generally for a matrix which is a power of 2, the n,m th element can be the XOR of the bits of n&m.

To obtain other Hadamard matrices we can permute the rows and columns, and also complement any sets of rows and columns. However, permuting rows merely permutes the offsets, and complementing rows merely changes the signs of the offsets.

Thus all these are equivalent to permuting the input iris code and complementing some digits of it, while using a fixed Hadamard matrix.

All the rows can be generated from 11 basic rows by XOR-ing them together according to the ones in the row number. If a gray code sequencing is used each row can be generated from the previous row by XOR-ing the appropriate basic row.

Thus in practice we can store 11 basic rows or generate them rapidly. They are
0101010...,
0011001100...,
00001111000011110000... etc.
Then the production of each Hadamard vector is obtained by one vector XOR operation.

Protocols Using Keys from Faulty Data (Transcript of Discussion)

David Wheeler

University of Cambridge Computer Laboratory

1 Introduction

What I'm interested in is, how can you construct a key given some analogue information like fingerprints or an iris scan — a set of measurements which have the property, they're probably analogue, and that if you repeat them, you don't quite get the same answer.

So the first question is, is it possible to derive a key from error-prone data of this form? As far as I know, the answer is no. However, you normally change the problem if you can't solve it, and what you have to do in this case is to not rely solely on the fingerprint data, but on some open (*i.e.* publicly known) auxiliary information as well. If you do that, then you can then construct a key from this faulty data. Now this is not new, I believe it's been done in quantum cryptology for years, but nevertheless it's possibly worth working our way through the procedures.

2 Basics

We first of all need a model. The model that I shall take is the iris code of the eye, which can be captured by a camera and then reduced to 2048 bits plus a mask of the same length. (This is existing iris code technology. I happen to have access to John Daugman's data so I have 2000 or so iris scans of which round about 600 are from the same eye.) The model is that Bob is a data centre and has one copy of an iris key, and Alice is somewhere in the field and wishes to transmit or generate a key between, I suppose, a secure terminal and the database. There's a number of other models you can use, but this is the kind of reference model that I will use.

We assume a lot of things: we assume a source, a destination, there's communication between the two, and I assume there's no correlation between adjacent bits, which is untrue, I assume problems of synchronisation don't exist, which they don't in this case, but in many cases they do, and that's just the beginning. We assume that the source and destination have different samples of the same iris code, perhaps a new sample is taken for each key generation. The number of bits which differ between the two samples is perhaps between 1% and 30% of the total, and the source and destination want to agree a key.

Now, how should we arrange to solve this problem? The answer really is rather simple. If we take Alice and Bob, and let C be Alice's error-prone code,

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 180–187, 2002.

and let D be Bob's, then E and F are forward error-correcting bits corresponding to C and D respectively. We usually send forward-correction bits (which are often the remainder of the message when it is divided by a cyclical polynomial) at the tail end of a message, however in this case we don't send the message, but *only* the error correction bits.

The protocol then proceeds straightforwardly:
Alice generates E from C, and sends E.
Bob receives E, computes F based on D, and then XORs E to F.
By using this as a forward error-correcting code, Bob can work out the changes which must be made to D so as to obtain C. So both sides have C, which is the session key which has been generated between them. This assumes some minor things, like using a linear error correcting code and so on, but basically we can say (a) that we've finished, and (b) that Alice can now send the message M with the session key.

So this is actually a one-shot protocol to send a message or a key directly, using the iris code at both ends *almost* as if the iris code were a private key — but not quite. After that you can use the key in any way that you like: you can use it for authentication, you can use it for encipherment.

There's a number of design problems, such as how you design your error correcting code. Actually I'm not all that worried about the technology, but (roughly speaking) you could have a large number of errors to correct, and that's awkward. You can use various turbocodes and so on but it's still awkward. On the other hand statistics say that unless you can correct a large number of codes you're not going to be very efficient. In fact if you make reasonable assumptions then the chance of two errors turning up is about a half, so if you have an error correcting scheme which can correct only a single error then you really haven't done very well. And to make this really efficient, you have to correct some hundreds of errors. But perhaps efficiency doesn't matter.

Well, you've got a choice. In any system you've got (a) the copies of the same eye, which are different, and (b) copies of different eyes, which are vastly different. You have to choose somewhere in between to set a limit, which determines the design of your error correction. There are various choices you can make, you can minimise the number of false codes recognised or you can minimise the number of rejected clients, and you can choose the appropriate value for your purposes[1].

3 Error Deletion

I have spoken about error correction, but I shall now give a rather simple sketch of an alternative scheme which yields the same results. Instead of using error correction, each party can take their iris code, group all the bits (into pairs for example), and send the parity of each group of bits across the channel to their correspondent. Then they throw away any group for which the parity they

[1] If there are about e_s errors between different samples of the same source but e_d errors between samples of different sources, then we must choose E so that e_s errors could be corrected but not e_d.

received is different from the parity which they sent. Now the remaining groups (pairs) have the property that they have an even number of errors, and you've rejected most of the errors anyway. Again this depends upon probabilities and you can do the maths and get some figures. I don't intend to do that now, but if you repeat the process again and again, in a multi-message protocol, you will eventually get some digits which are the same at both ends and which can be used as a key.

In all the methods which I describe, in general you will send also a hash verification of the code, just to make quite sure that things haven't got mislaid or something. And in some cases it might be worthwhile deriving the session key by an extra hash, so as to prevent the various relative key attacks which exist.

4 Direct Random Key Utilisation

Now, there's an alternative to this which I think somebody called fuzzy logic (I seem to have derived it slightly differently) which deals with very many of these problems. If we take direct key generation, and suppose that we want a 128-bit key, then A sends 2048 bits to B, and they classify these into 128 groups of 16 bits each. A sends the bits in each each group XORed by the appropriate digit of the new random key, and at the far end B will use his version of the iris code and XOR this with what he got, and use the majority for each group as the corresponding key bit. This is quite easy to do, it takes very little computation, and in addition you've got the counts out of 16 which indicate the reliability of the bit which you computed, and you can exploit that in your error correction.

But in any case, if you just do this simply, you've got a few errors remaining. You can correct these by doing a search. This is not usually the way of correcting errors, but if you have the value of a hash function of the key, and you try a few keys, there's just one which will give the correct result when hashed. So you can search among likely candidates, by putting them into a probability order using the reliability counts and then searching. But what is more, you can use this technique to extend the range of the error correction that you do at the receiver. You wouldn't like to search for everything, but searching for perhaps half a dozen elements is not particularly bad.

We should perhaps consider how bad this is for Alice and Bob, or how good it is for Eve. You've sent the entire code with one key bit per 16 bits so you've revealed most of the information in the iris code, so you might imagine that Eve had a good time of it. However, the leakage is definitely limited. Although Eve knows the exact iris code — except for the sign of each of the 16 bit groups — she gets no more information each time the code is used, provided the key is made at random. She's still equally uncertain about which groups are true and which are inverted. You will naturally point out to me that there is a problem here, that there is a relation between successive uses of the same key, because Eve will know the differences between all the keys, though she doesn't know the first one. But this — and avoiding related key attacks — can of course be

tackled by hashing the 128 bits to make the working key, and so it's really not particularly serious.

5 Transformation Methods

Let me move on further. These is a separate set of *transformation* techniques which are useful, and fortunately there's a rather large number of them. You take the original iris code and you transform it and then work on the transform. I think there's some work about this in the literature. What is the best transform? I chose the Hadamard transform because it's so easy to implement: you just XOR Alice's error-prone code C with a set of vectors and count the ones in each result. This is rather faster than multiplication — although in this particular case it is logically equivalent to multiplication — and the transformation is reversible so you've lost no information. It amounts to projecting C onto a set of vectors. Having done the transformation, you subtract the mean number of ones — half the number of bits — from each count to give a set of signed offsets, and you can take the sign of each offset as the corresponding key bit.

Now, if you think about it, some offsets aren't any use. Offsets which are near to zero contain almost no information. If they're large then the sign bit is rather well defined, because it takes a lot of errors to change the sign. So it's worthwhile doing a bit of rejection, or selection, and you can do this as we did before. You can say, let us therefore form a transform and select the best values, the ones with the highest offsets. There's a variety of ways you can do this, you can search, for example, the best in 16 and do this 128 times, or you could say, I will choose the best 128 bit offsets[2]. So we can do this in various ways, I'm not certain which is the best way, it probably depends upon the practical circumstances.

We have again to consider how we're leaking the key to Eve. The information that Eve gets tells her which offsets are used, but not the size of the offsets. So it is very difficult for Eve, basically there's very little information leaked. It will almost certainly do for once. If however you repeatedly do this, you should use a different Hadamard transform and so ensure that the relations don't boil down[3]. I don't think you can prove that this is quite as secure as the direct method, but its security is at least reasonable at first sight.

We can of course transcribe these into various protocols. We can make a two message protocol, where A does the transform and sends the locations of the N largest offsets to B, and B selects the $N/2$ largest of these, according to his list,

[2] We can send an m bit code indicating the best of the following 2^m offsets. This reduces the message length. A few error-correcting bits will still need to be sent, unless the probability of an error has been made sufficiently small. The information leakage is also reduced by this method.

[3] If error-correction is used, the parity equations correct the signs, so Eve obtains a set of equations for the sign bits. However, if a salt is used to select the precise transformation used, then each set of equations is distinct and the accumulation of data is little help.

and sends these locations back to A. These are likely to be reasonably reliable, you will have to do very little error correction and so on the whole it's really good.

There's an argument which I will verbally describe: if you've got a large offset for A and you randomly perturb it, half of the time this will increase the offset, the rest of the time it will decrease it, so B should get rather good results by selecting the half which may have improved. That's not a strong argument, I'm sure the mathematics could be elaborated. So we can make this fairly simple, straightforward protocol using two messages with a transformation.

I've waffled about precautions, I'm sure you can devise as many precautions as I can, I see no point in going through them in detail. You should probably make a hash verification, which should have enough bits and be invertible only by search. You should always use a salt, so that the generated keys are different each time. And there are various other practical precautions[4] which would be the same as for other systems.

6 Error Correction Methods

I'll now dive back to talking about error correction methods. We can use any standard "pure" error correction method, such as turbocoding, BCH etc, but we only send the error correction bits, we don't send the data, and so methods which scramble the two together are not really very useful. Secondly we don't have to send very much data, only the error correction bits, and this means the amount of information which is sent is somewhat smaller. However there's one or two other advantages which we have, because the error correction bits don't actually need correcting. (There may be transmission errors, which you deal with independently, but when I use the term "error" I'm stretching the language, I really mean differences between the two iris codes.) So we have a considerable gain, in that the number of bits needed is far less than you might imagine from a standard error correction method. In the case of errors at the rate of 1 in 9 it's about half.

But you can go further. One rather nice thing is, if you've got a BCH code or something like that, you can arrange that the error correction bits are zero by altering your selection. To do that you can solve a set of linear equations, and change the selection so that the error correction bits become zero at both ends. When I say change the selection, there's a degree of choice about this. Although any choice will do, ensuring that the resulting offsets are as large as possible will make life much simpler for the receiver.

However, there is a further thing that you can do, which I rather like. You can take the original iris code, you can work out the error correction bits, you can then change the iris code so the error correction bits become zero. And you do this by doing a standard error correction using the bits that you've got. If

[4] For example some non-linearity should be included, so that the solution of non-linear equations is needed to find the key. The transformation methods seem to do this quite well.

you do that then of course the error correction bits become zero and you don't have to send them[5].

There's the obvious questions which I think I mentioned earlier. What is the best way of doing the selection? Do you choose the best 128 bits, or the best of 16 bits, 128 times? Actually transmitting the information is worse in the case of the best 128 bits[6], but on the other hand, they're a better selection and the chance of errors is reduced. So there's room for someone to do some design work there.

I won't go very far into applications, I suspect you can come up with those just as well as I can. So as I say, there are various applications, you can imagine them, I can imagine them so I won't waste any more time on those[7].

There is a one-message protocol which I worked out in some detail just to show how. From the tables in the paper you can estimate that the error rate is about 3 in 1000, which is (3/16)/64 bits if you want a 64 bit key. If you use a Hadamard transform simplified by using Golay codes then you can send about that many bits and the chance of a failure is of the order of two in a million, or something like that.

We can do the same thing with a two-message protocol. It's much easier to make a two-message protocol, but it's probably also useful in practice because the second message can delete some of the unwanted or very weak information bits which you otherwise have to do by error-correction. The two-message protocols work quite nicely.

I have almost come to the end. There's some minor notes on Hadamard codes, most of which you already know. But they do have some interesting properties only some of which I've mentioned. The code, the actual matrix, is very simple, it's all noughts and ones, there's always exactly n/2 noughts and n/2 ones in every row, except one. It does have very simple ways of being generated, it comes down to using one XOR of a vector for each row that you generate, and you can work it out from a basis. All of which means that it doesn't take much time to do the generation, you may as well use that and search for random bits. You can see the pattern which arises from Hadamard, you can permute the rows, you can permute the columns, you can complement the rows, you can complement the columns, all of which give you different Hadamard matrices. However, these are all equivalent to taking the input stream, permuting it and randomly changing noughts to ones. So it's quite easy to get a selection of effective Hadamard matrices.

So what can we say in conclusion? In some circumstances we can use this technique to extend the power of a PIN, or to be useful as a key. It does not do

[5] The changes in selection are made by choosing instead the best offset of opposite sign in the same group.

[6] You must send about 5.4 bits per selection rather than 4.

[7] For example you could use a voice password, although the synchronization requires care. An error-prone code can readily be used for authentication directly, however our methods allow a key generation, and leak less. Although these methods can be used to reduce the size of a database, the search time may actually increase owing to the extra calculations.

what you might want, which is to enable you to search a database quickly for a nearest match, because unfortunately work has to be done at both ends. For every reference to the database you've got some work to do, so you may as well use the simplistic XOR in most cases for nearest match.

Well that's roughly what I have to say, has anyone any questions?

Jan Jürjens: Is it the case that I can generate keys from two different iris scans and use the one key to encrypt something and the other key to decrypt the encrypted message and get back what I started with?

Reply: No, it always depends on two different iris scans. I am assuming for simplicity that you deposited one iris scan in a database. You then generate a new one, and by sending the error correction bits, etc, you can generate a key which can be used at both ends because both ends know it. And you can use that key for anything you care to do, for example, authentication, sending a message, and so on.

From two different iris scans, one in the database, you derive a common key. But if you are wandering around the world, and you have a different iris scan at every camera you use, then the codes will all be different, so the keys will all be different. The key is not necessarily unique.

Jan Jürjens: If I take the iris scan at one point, how do both ends know this is *my* iris scan? What if someone just scans my iris without me knowing and generates a key and pretends to be me... Does this mean I have to keep my iris scan secret?

Reply: Yes. With PINs you can have a camera which reveals your secret to an attacker. In the case of iris data, if somebody uses a camera on your eye and generates the appropriate iris scan, they can indeed generate a mutual key and behave as you[8]. Anybody can do this, and you can do it from some distance away. Of course, carrying a notebook containing your key is equally vulnerable.

You should take precautions against this by combining these techniques with other protocol elements. There's no particular reason why you can't throw a few extra errors into your iris code and then it's different each time and the leakage is reduced, but I'm not saying it's made zero.

Michael Roe: As a general design principle, you shouldn't use biometrics as crypto keys because biometrics aren't secrets. Biometrics may start off being secrets, but they eventually become not secret and, unlike crypto keys where you can just generate a new one, biometrics are fundamentally fixed for the lifetime of the person whose biometric they are.

If only trusted systems ever measure your biometric then they can hash it and salt it and produce a separate secret each time which doesn't compromise the original biometric, but unfortunately you can't make sure that untrusted systems don't measure your physical characteristics.

But the schemes that David is presenting here also work in a setting where you're not relying on secrecy in a biometric at all. Suppose that *everybody* knows your iris scan, but some trusted piece of hardware scans you, takes from its elec-

[8] Actually it's worse than this, even a passive Eve can then intercept all subsequently agreed keys.

tronic database the error correction bits, reconstructs them and signs something — using credentials that that trusted box altered itself — which says, a person who matches this typical pattern was next to this measuring device at this particular time.

Matt Blaze: Right. Then you'd have to trust that the person was in fact what was in front of the measuring device.

Michael Roe: Yes, it's a trusted device. And not just the CPU, it has to go right the way up.

Bruce Christianson: A lot of the literature on the "biometrics are better than passwords" campaign wagon doesn't set out the threat model clearly. In a lot of cases there's an implicit assumption that the hardware at the client end is trusted by both parties, and all that the client hardware is actually saying is, this biometric is physically present in my field of view at this time.

Matt Blaze: And it's not just a trusted measuring device, it's a trusted measuring *process*. I presume at some point it will be possible to build a little iris that I can hold up to a machine? Now the threat is, I work for AT&T, have an iris scan, then I leave AT&T and go work for NTI. Now the AT&T people have my iris data, they can go build something that lets them get into NTI offices.

David Wheeler: That's assuming, for example, they both used the same eye. I should say here that John Daugman went into this in some detail, and the usual iris scan process does measure the actual eye motion, and it takes a variety of scans so a dead picture wouldn't work. But you could make little robot, yes.

Bruce Christianson: We can imagine a macabre scenario involving a glass jar and a pump. But as Mike points out, David's protocol can be used to address a slightly different problem: it allows a third party to agree a key with you and neither of your employers has insight into the key that's been agreed between you and the third party. The third party doesn't know which employer you currently work for or which eye they use. Trusted hardware is still needed, but at least now it is an orthogonal problem.

However iris scanning is just one example of a wide range of applications for this. Another field of application is using astronomical beacons to agree keys, For example, you and I might agree that we will use as a shared key the result of a recording made by pointing a radio telescope at the sky in a certain specific direction. Now the attacker is unlikely to be able to pre-record the entire stellar field, so subsequent leaking of where the telescope was pointed doesn't allow the attacker to break back into the past and obtain the key. David's approach gives us a protocol that we can use to get a shared key in spite of the fact that our recordings will have been slightly different because of television broadcasts and so forth. And the exchange of information that David and I make as part of this protocol doesn't reveal anything about the key that we eventually agree.

On the Negotiation of Access Control Policies

Virgil D. Gligor, Himanshu Khurana, Radostina K. Koleva,
Vijay G. Bharadwaj, and John S. Baras

Department of Electrical and Computer Engineering,
University of Maryland, College Park, Maryland 20742
{gligor, hkhurana, radost, vgb, baras}@eng.umd.edu

Abstract. Although the notion of negotiation has been used extensively in secure communication protocols to establish common keying states, protocol modes and services, this notion is only now emerging in the area of access control policies. In this paper, we review the motivation for access policy negotiation and we provide some examples of use in practice. In particular, we illustrate the meaning, the types, and the process of negotiation for establishing common access states and common interpretations of policy models for dynamic coalitions and large-scale access control in the Internet.

1 Introduction

Collaborative computing requires a variety of resource-access agreements ranging from those for peer-to-peer sharing of application objects and services to those for joint administration of access policies among autonomous domains of the Internet. An important characteristic of such agreements is the negotiation of common access to a set of resources reflecting the sharing preferences of the parties involved. Such negotiations typically seek agreement on a set of access properties, which represents the *common interpretation of a policy model*, and then on a *common state*, which reflects the shared accesses to resources of the parties involved and satisfies the negotiated access properties. We say that such an agreement is the result of the negotiation of access control policies.

A common interpretation of the policy model adopted by collaborating parties is essential for reaching resource access agreements, particularly when joint administration of shared resources is being sought. The common interpretation of the policy model consists of the specific properties of access authorization, management, and attribute-definition components of the policy model implemented by the collaborating parties. For example, the access management area may include the property that selective and transitive distribution of privileges requires selective and transitive revocation of privileges; or that the distribution and revocation of privileges be owner-based. Such properties must be supported by all collaborating parties for joint administration of shared resources and hence must be negotiated.

In this paper we illustrate the salient aspects of the negotiation of access control policies in the context of dynamic coalitions formed by autonomous domains

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 188–201, 2002.

of the Internet. We view dynamic coalitions as peer-to-peer networks that include resources which are jointly owned and administered by the member domains [9]. Such coalitions have two characteristics that help motivate the key aspects of policy negotiations, namely membership that (1) comprises domains managed under diverse access policies reflecting different sharing interests and (2) varies dynamically thereby ruling out off-line, one-time access policy agreements.

Previous work in the area of access control for dynamic coalitions illustrated some of the important aspects of access policy negotiations in specific settings. For example, Shands *et al.* [12] and Herzberg *et al.* [6] addressed the problem of unambiguously specifying a common access state, communicating this common state to all member domains, and committing this common access state. However, this work assumes that common access states were agreed upon by extra-technological (e.g., off-line) means and that all member domains have the same interpretation of the common policy model (i.e., a common interpretation of a role-based access control model). Other work addresses specific aspects of bilateral authorization-check negotiation, for instance those that enable clients and servers to agree on common authorization properties; i.e., matching client credentials with server access checks [13,11]. Although this work on authorization-check negotiations introduces the important notion of client-server trust negotiation, it addresses neither the notion of access-management property negotiation (e.g., negotiation of common access distribution and revocation properties) in multiparty settings, such as those of dynamic coalitions and *ad-hoc* peer-to-peer networks, nor that of common state negotiation.

In contrast, our work explores the possibility of automating the negotiation of a common access state whenever all the domains of a coalition share the same interpretation of a common policy model. We cast this negotiation problem as one of satisfying diverse coalition-member objectives and use well-known results in game theory to find solutions. We also consider coalitions where member domains do not share a common policy model or model interpretation, and hence must negotiate them, since a common policy-model interpretation is a prerequisite for establishing common access states.

The rest of this paper is organized as follows. In Section 2 we present the negotiation of common states with examples. In Section 3 we discuss negotiation of policy model interpretations with some examples. Section 4 concludes the paper.

2 Negotiation of a Common Access State

We define the problem of negotiating a common access state as follows. We assume that a number of autonomous domains wish to collaborate to achieve a *common objective* that is desired by all domains and achievable by none of them individually. To achieve the common objective, these domains form a coalition and share some resources (e.g., data and applications). That is, each domain has a set of resources that are of interest to the others and is willing to share some or all of its resources with some or all of the other members to achieve the

common objective. Resource sharing by a domain with other domains means that the domain (administrator) grants access privileges for the resource being shared to the other domains. The sharing relationships among these domains are defined by the access state of the coalition. An access state is defined by the permissions each member domain has to the shared resources of that coalition. Thus negotiating a common access state means obtaining the agreement of each domain to share some of its data and applications with the other domains of the coalition. Such an agreement implies that each domain values achieving the common objective more than retaining private access to the resources it must share with other domains. In addition, a common access state of a coalition may include some jointly owned resources created during coalition operations, and a common policy for accessing these resources must be negotiated. Access to jointly owned objects is one of the benefits individual domains derive from their membership in the coalition.

In any coalition, there may be a number of common access states that satisfy the common objective. However, member domains may place some constraints on which of these states are acceptable; e.g., these constraints may arise due to technological limitations, or they may be self-imposed. For example, a domain whose participation is deemed essential to achieving the common objective might use its extra bargaining power to stipulate that it only accepts states in which every other domain contributes at least as many resources as it is contributing to the coalition. Even among the states it considers acceptable, a domain may prefer some states over others. For example, a domain may prefer to share certain resources only with those domains that it considers to be close allies; an example of such a situation can be found in [3]. The negotiation process aims to find a preferred common state that satisfies the constraints of all the domains. We expect that in a typical application these constraints would be programmed by the domain administrators at the start of the negotiation, and from that point on the process of finding a common state would be automated.

We divide negotiations into three classes based on the types of constraints under which domains negotiate. These classes, which are progressively generalized, differ from each other depending upon whether all the objectives of each domains (1) coincide with those of other domains and (2) are known to the other domains of the coalition.

1. **No constraints.** In this case all the objectives of all domains coincide and all objectives are known to all domains. In a typical case, all domains have a single common objective.
2. **Global constraints.** In this case some of the objectives of some domains may not coincide with those of other domains, yet all the domains have complete knowledge of each other's objectives.
3. **Local constraints.** In this case some of the objectives of some domains may not coincide with those of other domains, and domains may not have complete knowledge of each other's objectives.

Although some objectives of some domains may remain unknown, we assume that each domain has complete knowledge of which resources of interest

are possessed by the others. (An even richer set of negotiation problems would emerge, if we relax this assumption. That is, domains would have to guess at what resources other domains have, and tactics such as bluffing, commonly seen in real-life negotiations, would undoubtedly be used. These negotiation problems are outside the scope of this paper.) Given this assumption and the three negotiation classes, the following simple protocol arises naturally. The negotiation starts with one domain proposing a common state. Each of the other domains can then either accept this proposal or propose an alternative. Agreement is reached when all the domains accept a single common state. Within this simple negotiation protocol, the following four questions arise:

a. *Termination.* If a common state exists that satisfies all the constraints and all domains' objectives, will it be reached in the negotiation? How can the member domains detect if a negotiation is going to fail (if no mutually acceptable state exists)?
b. *Strategy.* What is the best negotiating strategy for each domain? In other words, how can a domain pick the content and/or sequencing of its proposals to ensure a more favorable outcome for itself?
c. *Preference specification.* How should domain preferences be specified to allow automated negotiation?
d. *Stability.* Is the negotiated state stable (i.e., self-enforcing)? In other words, is the common state such that no domain has an incentive to cheat or renege on its part of the agreement?

Below we illustrate the three classes of negotiations in some detail, and answer the above questions for each class.

2.1 Negotiation with No Constraints

In this type of negotiation, all domains have a common objective and are willing to accept any common state that satisfies that objective. Some of the previous work in this area [6,12] falls into this class; i.e., any proposal from the other domains is accepted, and extra-technological methods can be used to ensure in advance that only acceptable states will be proposed.

In this case, the answers to our four questions are obvious.

a. *Termination.* The negotiation always concludes successfully.
b. *Strategy.* The negotiation strategy is irrelevant, as any proposal will be accepted (usually, extra-technological agreements are made in advance to restrict abuse of this assurance by any domain).
c. *Preference specification.* Domain administrators need not specify anything beyond the common model of access policy and the common policy model interpretation.
d. *Stability.* Since the common objective is all-important, domains will not try to subvert the agreed-upon state. Hence, any state that helps to attain the common objective is stable.

As an example of this type of negotiation, consider a situation where a university needs to share some research data with engineers of some outside companies to get their feedback, with the *common objective* being the successful transfer of technology. In this case these companies simply tell the university which of their engineers are working on the relevant project, and the university role-based access control system places those engineers into local roles that have access to the necessary research data. Mechanisms for such role assignment have been studied by Shands *et al.* [12] and Herzberg *et al.* [6].

2.2 Negotiation with Global Constraints

Now consider the case where the coalition domains face (or have agreed upon) a set of constraints the common state must satisfy. Each domain knows all these constraints, and will accept any common state which satisfies all the constraints while achieving the common objective. It is possible that such a state does not exist, in which case the the negotiation fails. Note that since the constraints are known to all, any domain can compute the best common state independently.

For example, consider three domains representing three airlines that wish to share six route types to expand their market coverage. Here a route type corresponds to a certain set of destinations, for instance, U.S. - Europe, U.S. - Asia, U.S. - U.K.. Sharing a route implies that the domain which owns the route gives some users of a foreign domain the access privileges required to execute the route applications (e.g., reservations, billing, communications).

Table 1. Negotiation with global constraints: Route sharing.

Domain	Route Type Controlled	Number of routes in type
*	1	5
	4	8
1	5	4
	6	7
*	1	5
	2	6
2	3	2
	4	9
*	1	7
	2	4
3	4	6
	5	3

The initial state before negotiation is shown in Table 1. Observe that some route types may already be common to multiple domains; e.g., route types 1, 2, 4, and 5. Further, each route type consists of a number of routes, depending on which domain it belongs to. The *common objective* is to provide all three

domains with access to all six route types. The following global constraints have been agreed upon:

- Each domain must share at least one route type.
- Domains that have unique route types must share them. As a consequence, domain D1 must share route type 6 and domain D2 must share route type 3.
- Sharing of route types must minimize the number of routes shared (as possible application of the *least privilege principle*); i.e., if two or more domains are capable of sharing the same route type, then the one that comprises the lowest number of routes will be used.

Upon inspection of Table 1, we observe that two common states would satisfy these constraints. Domains D1, D2 and D3 could share route types {1,6}, {3} and {2,4,5} respectively (we call this Solution A). Or they could share {6}, {1,3} and {2,4,5} respectively (we call this Solution B). Either of these is acceptable to all domains, and as soon as one of them is proposed, it will be accepted. Any other state would violate at least one of the constraints, and will be rejected if proposed.

With this example in mind, let us try to answer our four questions.

a. *Termination.* Success or failure of negotiation depends on the specifics of the problem and on the constraints. However, each domain can always construct the set of common states that would be acceptable to all the others, and can predict whether the negotiation will fail or not by checking whether this set of acceptable states is empty.
b. *Strategy.* The domain that makes the first proposal has an advantage - it can construct the set of acceptable states and propose the one most advantageous to it, knowing that the proposal will be accepted. For instance, if D2 were to make the first proposal in the above example, it would propose Solution A, so that it has to share fewer of its own objects; similarly, if D1 were to make the first proposal, it would propose Solution B. Any other proposal would be meaningless, since the domain would know in advance that all the others would not accept it.
c. *Preference specification.* Domain administrators need to specify the information required to find the least-privilege solution, namely the data on which route type provides access to how many routes. In other words, data pertaining to the global constraint must be specified for the negotiation to be automated.
d. *Stability.* As long as the common objective is the primary goal for all the domains, no domain will try to subvert the agreed common state.

In the above example, we assumed that all the domains are committed to achieving the common objective. More complex negotiations may arise when domains have competing or even conflicting objectives. For example, consider the setting illustrated in Table 2. Here we have two are two domains, each of which controls a certain number of routes. Routes can be of three types. Each

domain has a set of three objectives it could pursue (e.g., markets to which the airline could cater), and each of these objectives has a certain value to that domain (e.g., the profit that could be made from that market). However, each such objective requires additional resources, in the form of routes, as shown in Tables 2(a) and 2(b).

Table 2. Game theory in negotiation

(a) Domain 1

Objective	Value of Objective	Type 1 Routes (U.S. - Europe) needed	Type 2 Routes (Europe - Asia) needed	Type 3 Routes (U.S. - Asia) needed
A	100	8	10	5
B	80	7	10	6
C	95	7	11	4

(b) Domain 2

Objective	Value of Objective	Type 1 Routes (U.S. - Europe) needed	Type 2 Routes (Europe - Asia) needed	Type 3 Routes (U.S. - Asia) needed
D	120	9	10	7
E	90	6	8	5
F	45	5	7	4

(c) Available resources

Domain	Type 1	Type 2	Type 3
1	20	20	11
2	18	19	11

We assume that each domain starts with a limited number of routes of each type (see Table 2(c)). The domains are unable to share routes, and can only trade some of their routes with each other in order to increase their profits. This is a route-trading (i.e., a barter) problem, unlike our previous example, which was a route-sharing problem.

In this case, the *common objective* is to increase profits for both domains through cooperation. However, this common objective no longer means the same thing to the two domains. The two domains will derive different profits from a successful negotiation, and each will seek to maximize its own profit. Profit maximization is an important aspect of most real-life negotiations, and we expect it to be a primary source of differences among the objectives of different domains.

Game theory [2] provides a mathematical framework for studying situations where individual rational decision makers are working towards competing or conflicting objectives. We can use game theory to find a solution for the problem in Table 2. First, observe that the best thing Domain 1 can do if it does not

collaborate with Domain 2 is to devote its resources to objectives A and B, and abandon objective C. This will result in a total profit of 180 to Domain 1. Similarly, Domain 2 should work towards objectives D and F, and abandon E, thus achieving a total profit of 165.

To find a barter arrangement that leaves both domains better off, we use the concept of the *nucleolus* [10]. The nucleolus is defined as the solution that minimizes the maximum dissatisfaction experienced by any one of the domains. The dissatisfaction is measured by the difference between what the domain could have achieved on its own and what it achieves as a result of cooperation. It can be shown that the nucleolus exists and is a stable outcome for very large classes of useful problems. Here, stability means that no single participant has both the power and the incentive to change the status quo. Thus, if any one participant in a negotiation tries to violate such an agreement, they will always be worse off than before, either because there is no more beneficial state for them or because the other participants can collaborate to punish the offender (for example by revoking his access privileges to their objects). A nice feature of the nucleolus in practice is that it is unique, and can usually be computed efficiently using linear programming techniques. If the nucleolus does not exist for a particular problem, its absence can also be detected efficiently.

For this example, the nucleolus solution is for Domain D1 to give one Type 3 route to Domain D2 in exchange for one Type 2 route. This will allow Domain D1 to fulfill objectives A and C, achieving a value of 195, while Domain D2 can fulfill objectives D and F, achieving a value of 210. Thus each domain achieves a higher value in this cooperative agreement, and therefore neither has any reason to try to violate it.

To illustrate the stability of the nucleolus in this case, we consider an alternative common state which also achieves the same profits for both domains. In this arrangement, Domain D1 gives two Type 3 routes to Domain D2 in exchange for one Type 2 route. However, Domain D1 can now depart from the agreement by withholding one of the two promised routes, without any ill effects (Domain D2 still achieves its best possible outcome, and so has no reason to try and penalize Domain D1). Thus this alternative is not a stable outcome. This observation is intuitively satisfying, since from a security point of view we can see that this alternative solution violates the least privilege principle (defined in the previous example).

In light of this second example, let us reexamine our four questions for the general case of any negotiation that can be cast in terms of objective values and resources, as in Table 2. (Note that our previous example (in Table 1) can also be recast in this form.)

a. *Termination.* All domains can check if the negotiation will succeed by checking if a nucleolus exists, if it does the negotiation is guaranteed to succeed.
b. *Strategy.* The best negotiating strategy is to propose the nucleolus.
c. *Preference specification.* Negotiation can be automated if domains specify the different objectives, their values, and resource limitations.

d. *Stability.* The nature of the nucleolus ensures that no domain will try to subvert it, since it is now in their interest to help achieve the common objective.

2.3 Negotiation under Local Constraints

Suppose, as in the previous section, that a number of domains with a common objective are seeking to negotiate a common access state. Each domain has its own constraints and preferences, as well as its own estimate of the importance of the common objective; i.e., each domain has its own objectives that may differ from those of other domains. It will accept any common state that satisfies all its constraints, fulfills the common objective, and is preferable to the outcome of non-cooperation. In other words, a domain will not agree to accept a common access state that requires it to give up more than it can gain from a negotiation that achieves the common objective.

Now suppose that no domain has complete knowledge of the constraints and preferences of all the other domains. Thus each domain must try and negotiate the best state for itself without knowing precisely what resources the other domains are able (or willing) to contribute to the common access state. Negotiation strategies become very important, since a domain cannot determine beforehand what is the best common state (as could be done in the case of negotiations with global constraints). Also, a domain cannot predict whether a common state exists that satisfies all the constraints of all the domains, so a domain cannot predict whether the negotiation will succeed. Detecting success or failure becomes part of the negotiating strategy, and a decision as to whether to continue negotiating must be made at each step of the negotiation. For example, consider the problem in Table 2, and assume that the contents of Table 2(a) are known only to Domain 1 and the contents of Table 2(b) are known only to Domain 2. This would constitute a negotiation with local constraints.

Another example of negotiating under both local constraints is shown in Table 3. This is a route sharing problem, with three domains and twelve route types. The *common objective* is to increase profits for all domains by increasing the number of routes accessible to each domain. Each domain has some constraints which it can reveal to the other domains, and some which it cannot. Note that if all constraints could be revealed, this would become a negotiation with global constraints. The local constraints imposed by each of the three domains are as follows:

- Domain D1: willing to share half as many routes as other domains because of some (e.g., geographic) advantage. D1 can reveal this constraint.
- Domain D2: willing to share twice as many routes as D1; unwilling to share route 6 and cannot reveal the constraint regarding route 6.
- Domain D3: willing to share twice as many routes as D1; needs one of the routes {2, 5, 6}, is unwilling to share route 11 and cannot reveal this constraint regarding route 11.

One way to carry out the negotiation would be as follows: all the domains construct lists ranking the various feasible common states in their order of prefer-

Table 3. Negotiation under local constraints

Domain	Routes Controlled	Routes shared in final solution
D1	1,2,3	2
D2	4,5,6,7,8	4,5
D3	9,10,11,12	9,10

ence, based on their local constraints. Each domain starts by proposing its most preferred state, and proceeds down the list if this proposal is not accepted. If the number of feasible common states is finite (as in this example) this negotiation will terminate, with success if some agreement is reached or with failure if all domains reach the end of their lists without obtaining agreement. The following two examples show negotiations that terminate in agreement.

1. – D1 proposes route-sharing state {D1: 1, D2: (4, 5), D3: (9, 10)}
   – Result: all domains agree and common state is committed
2. – D3 proposes route-sharing state: {D1: 2, D2: (5, 6), D3: (9, 10)}
   – Domain Response: D2 rejects it (unwilling to share 6), and instead proposes: {D1: 2, D2: (4, 5), D3: (10, 11)}
   – Domain Response: D3 rejects it (unwilling to share 11) and instead proposes: {D1: 2, D2: (4, 5), D3: (9, 10)}
   – Result: all domains agree and common state is committed

However, this approach has a problem, as shown by Arrow [1]. The Arrow Impossibility Result states that if each domain describes their preferences by a preference list (a preference list is a weak order, where an alternative A appears above another alternative B only if the entity making the list considers A to be at least as good as B), and if an impartial arbitrator tries to use these lists to construct an overall preference list for all the domains, then there is no algorithm for the arbitrator to always construct a list that is

– Pareto efficient; i.e., there is no other list that would make at least one domain better off (by ranking its more preferred alternatives higher) without making some other domain worse off.
– Nondictatorial; i.e., there is no one domain whose preferences are always followed, regardless of the other domains' preferences.
– Independent of irrelevant alternatives; i.e., the decision to rank alternative A above or below alternative B is taken only on the basis of different domains' preferences between A and B, and is not affected by whether, for instance, some domain ranks a third alternative C above or below A or B.

Furthermore, it can be shown that if the arbitrator's decision is restricted to just picking a most preferred solution instead of constructing a full preference list, then there is no algorithm for the arbitrator which is nondictatorial and

non-manipulable (i.e., some domain can achieve a better result by lying about its preferences).

This result has strong implications when we seek to answer our fourth question. It implies that a negotiation scheme based on preference lists will either not conclude, or if it does, the common state reached may not be stable, because

- it may not be Pareto-efficient, in which case there might be another common state that is better for all the domains.
- it may be dictatorial, in which case it is unfair to all but the "dictator" domain. This may cause the other domains to try and subvert the outcome of the negotiation.
- it may not be independent of irrelevant alternatives, in which case any domain could influence the outcome of the negotiation by proposing some extra common states, even infeasible ones.

Thus, Arrow's result suggests that, to construct useful solutions to such negotiation problems, we must at least have a way of defining *cardinal preferences*. In other words, it is not enough for each domain to be able to rank the possible outcomes; it should be able to quantify the strength of its preference for each outcome. Thus the information required to assign a numeric value to any given common state must be specified by the system administrator. This was the case in Table 2, where preferences were quantified as the value associated with each objective.

This discussion and attempts to provide answers to our four questions for the case of negotiation with local constraints opens several avenues of further research:

a. *Termination.* This question is strongly linked to the negotiating strategies adopted by the domains. Finding negotiation schemes and strategies that attain stable outcomes in an efficient way while allowing domains to detect failing negotiations is an interesting area for future research.
b. *Strategy.* It is not clear what strategies will be the best in this case. We conjecture that the best strategy would be for each domain to start with an initial offer and then make concessions if the other domains show a willingness to compromise, or break off the negotiations if the other domains are adamant. Known results of game theory [2] also point in this direction, but this has not been proven yet.
c. *Preference specification.* Arrow's result indicates that each domain must at least specify the strength of the domain's preference for all possible common states, or provide a rule for computing the strength of the preference given a common state.
d. *Stability.* This question is also related to the negotiating strategy and remains an area of future research. For example, if we could find a strategy that allows the domains to converge to the nucleolus in a finite number of negotiating rounds, then such a solution would always be stable.

3 Negotiation of Policy Models

In this section we discuss cases when negotiation of common policy models and common policy-model interpretations becomes necessary. We first describe the three types of policy properties that comprise a policy model. Then we give examples of negotiation of these properties in trust negotiation [13] and in dynamic coalitions.

Negotiating policy models involves negotiation of policy properties, namely, *access-attribute* (AT), *access-management* (AM) and *access-authorization* (AA) properties [4,5]. Access attributes include subject and object attributes (e.g., user, group, role, location identifiers, secrecy and integrity levels, time-of-access intervals), and AT properties typically establish invariant relationships among attributes (e.g., lattices of secrecy and integrity levels, user-group membership invariants, inheritance of role membership and/or role permissions). AA properties determine whether a subject's current access attributes satisfy the conditions for accessing an object given the current object attributes. AM properties include conditions under which subjects are granted or revoked attributes for accessing objects, subjects and objects are created or destroyed, objects are encapsulated within protected subsystems. In general, AM properties characterize commands and constraints that bring the system to desired states, whereas AA properties characterize constraints that ensure that the system remains in desired states.

A negotiation of AA and AT properties can be identified in trust negotiations between a client and a server [11,13]. The client sends requests to the server and the server needs to verify the client's credentials in order to authorize the request. However, the server is unwilling to disclose up front the AA rules; i.e., the type of credentials that will allow the client to have his request approved. The purpose of this is to keep some privileged association private, for example, if the company to which the server belongs is in an alliance with another company C and wishes to keep this fact private, then disclosure of the fact that a certificate from company C will allow the requested service would result in exposing the alliance. Also, the client is reluctant to disclose all his credentials along with the initial request in order to keep some privileged associations private and to protect sensitive private information (such as social security number, bank account number). So the client and server engage in a negotiation process where clients must provide credentials whose AT properties can be verified by the server, and the server discloses further AA rules to enable the client's request to be satisfied (the client may also require the server to present some credentials whose AT properties can be verified by the client).

A need for negotiating AM properties can be seen in a case when two domains, both of which use role-based access control models, are trying to share resources but each domain maintains different "rules" for assigning users to roles. Then the domains need to engage in a negotiation for a common set of rules that each one is going to use to assign foreign users to its local roles. For example, let Domain1 and Domain2 represent two airlines. Domain1 requires a user to present an identity certificate and a digitally signed proof of job title, in order to assign him to the local role of Travel_Agent1. For the assignment to a similar local

role of Travel_Agent2, Domain2 requires a user to present an identity certificate and a conventional company-identification credential. So the domains need to negotiate either one of the existing sets of credentials or a new set of credentials for foreign user assignment to roles. For example, Domain2 can agree to accept digitally signed job titles or alternatively, or both domains may agree to accept digitally signed credentials issued by a trusted third party.

Another scenario for negotiation of AM properties is possible in regard to selective revocation of attribute certificates [7,8]. Selective revocation means that whenever an identity certificate is revoked, so are all the attribute certificates that were issued based on this identity certificate. Now consider the situation when two domains are trying to collaborate, but one of them supports selective revocation and the other one does not. The domains must then negotiate AM properties of attribute certificate revocation before distributing privileges to users. There are three options, each of which may cause a change in AA properties. The first option is that both domains agree to support selective revocation, then on every access request only the attribute certificate's validity need to be checked. The second option is that both domains agree not to support selective revocation. In such a case both identity and access certificates need be verified. The third option is to have each domain support its individual AM properties. In such a case it is necessary to distinguish requests from users from different domains and for each case use corresponding authorization policies.

4 Summary and Further Research

We defined the problem of negotiation of access control policies in the general setting of dynamic coalitions and *ad-hoc* peer-to-peer networks. We presented examples of negotiating common access states and common policy-interpretation properties, and identified some areas of future research. We intend to develop a prototype for establishing dynamic coalitions among autonomous domains and for conducting negotiation of common access states under different types of constraints. This prototype will include a flexible language for negotiation that can capture a large set of real-life examples, not just the examples mentioned in this work. We also intend to illustrate how game theory can be applied to negotiations in practice, for example in negotiations where cardinal preferences can be defined.

Acknowledgments. This work is supported by the Defense Advanced Research Projects Agency and managed by the U.S. Air Force Research Laboratory under contract F30602-00-2-0510. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency, U.S. Air Force Research Laboratory, or the United States Government.

References

[1] K. J. Arrow. *Social Choice and Individual Values.* Yale University Press, New Haven, CT, second edition, 1963.

[2] F. Forgo, J. Szep, and F. Szidarovsky. *Introduction to the Theory of Games: Concepts, Methods, Applications*, volume 32 of *Nonconvex Optimization and Its Applications.* Kluwer Academic Publishers, 1999.

[3] T. J. Gibson. An architecture for flexible multi-security domain networks. In *Proceedings of Network and Distributed System Security Symposium*, San Diego, CA, February 2001. The Internet Society.

[4] V. D. Gligor, S.I. Gavrila, and D. Ferraiolo. On the formal definition of separation-of-duty policies and their composition. In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, Oakland, California, May 1998.

[5] V.D. Gligor and S.I. Gavrila. Application-oriented security policies and their composition. In *Proceedings of Security Protocols 6th International Workshop*, Cambridge, UK, April 1998.

[6] A. Herzberg, Y. Mass, J. Michaeli, D. Naor, and Y. Ravid. Access control meets public key infrastructure, or: Assigning roles to strangers. In *Proceedings of IEEE Symposium on Security and Privacy*, Oakland, California, May 2000.

[7] H. Khurana and V.D. Gligor. Review and revocation of access privileges distributed with PKI certificates. In *Proceedings of the 8th International Workshop on Security Protocols*, volume 2133, pages 100–125, Cambridge, UK, April 2000.

[8] H. Khurana and V.D. Gligor. Enforcing of certificate dependencies in ad-hoc networks. In *Proceedings of the IEEE International Conference on Telecommunications*, Romania, April 2001. ISBN: 973-99995-1-4.

[9] H. Khurana, V.D. Gligor, and J. Linn. Reasoning about joint administration of access policies for coalition resources. Submitted for publication. Available at http://www.glue.umd.edu/~gligor.

[10] D. Schmeidler. The nucleolus of a characteristic function game. *SIAM Journal on Applied Mathematics*, 17:1163–1170, 1968.

[11] K. E. Seamons, M. Winslett, and T. Yu. Limiting the disclosure of access control policies during automated trust negotiation. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, California, February 2001.

[12] D. Shands, R. Yee, J. Jacobs, and E. J. Sebes. Secure virtual enclaves: Supporting coalition use of distributed application technologies. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, California, 3-4 February 2000.

[13] W. H. Winsborough, K. E. Seamons, and V. E. Jones. Automated trust negotiation. In *DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, Hilton Head, SC, January 2000.

Negotiation of Access Control Policies (Transcript of Discussion)

Virgil D. Gligor

University of Maryland

This is a small part of a larger project on 'Integrated Security Services for Dynamic Coalition Management', which is a DARPA project of the University of Maryland and this work has been performed by my colleagues and students: H. Khurana, R. Koleva, V.J. Bharadwaj and John Baras.

Firstly the question is, what is a dynamic coalition, and then what do we mean by negotiation of access control and why do we need to do it? Basically a dynamic coalition is formed by a group of parties that have a common goal and want to share a number of objects to achieve the common goal. So they conduct negotiation on multiple dimensions. This presentation focuses on the dimensions of such a negotiation.

Now we've seen the term negotiation used in secure communication protocols — perhaps some of the ones that we dislike such as Oakley, although Oakley is a better variant of its successor IKE — and generally what we negotiate here is a selection of a common state, in some sense. We try to select a common keying state between two parties that have never communicated with each other before and we select perhaps the Diffie-Hellman parameters, we select the shared key, we become aware of or have a common awareness of the identities involved and their nonces. We also select the encryption modes, the hashing, the authentication algorithms, possibly the compression techniques that the two parties share, and this all will become parameters of the shared common state.

We also select common protocol modes. For example, we may want to use an aggressive mode as opposed to a conservative mode of authentication. So an aggressive mode might be a mode with very few messages but with highly encoded semantics; really dangerous, but nevertheless useful in some cases. We select common security services and features such as identity privacy, perfect forward secrecy keys, and perhaps identities. So that's essentially what we negotiate, at least in this example of communication protocols.

In access control policies we seem to negotiate two distinct things and also the combination of the two. First, just like before, we negotiate the use of a common state. Here the common state is a state obtained by the assignment of privileges for coalition objects and coalition services to users from foreign domains. The idea is that we have a number of foreign domains that try to form the coalition, and they contribute common objects to the coalition. They run common applications on these common objects to achieve the common goal and the common state will be represented here by this assignment of privileges, by the sharing of privileges to these common objects. That's what we mean by common state.

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 202–212, 2002.

So far there is no real bargaining in the selection or in the definition of the selection of the common state, but as you will see later we can have negotiation with bargaining or without bargaining depending upon the constraints that we put on the negotiation process.

We can have constraints on the common state. We can have global constraints such as making sure that we share the least set of privileges. We can have local domain constraints such as a certain domain may refuse to share some of the objects that it has, and of course the goal is to find a common state that satisfies all the coalition members. By that we mean all the common applications that the coalition members execute.

In addition to negotiating the common state, we can negotiate on the interpretation of a particular security policy model. In a security policy model, at least in our conception, we have three different kinds of properties that you can negotiate on:

1) standard access authorisation properties that specify more or less what actions require what privileges for carrying out the action.

2) access management properties which specify the conditions under which access privileges are distributed, revoked, reviewed, and so on.

3) access attribute properties which specify the properties of attributes of access control regardless of the other properties

For example, you may have properties where the security levels defined form a lattice, or the rôles that you define form higher key privileges — also inherited — based on this higher key. You may have properties of rôles: regardless of how rôles are assigned, you have properties of rôles themselves. So these are the kinds of properties that we want to negotiate in terms of the policy model.

What we have in mind is essentially creating the coalition domain dynamically, fairly dynamically, although not as dynamically as the kind of ad-hoc network that you create when you have personal devices. We'd like to have some sort of a server that enables us to do the negotiations, and also perhaps another related server, that actually constructs certificates representing negotiated common privileges and distributes those certificates to users. Clearly we intend to rely on the authentication authorities for identity certificates for the various domains participating in the negotiations, because we envision those domains to be fairly independent.

Let me start out with a simple example of the negotiation of a common state with global constraints. Suppose that we have three companies, such as United Airlines, British Airways and Delta, and they form a coalition called Star Alliance. The goal of this coalition is to increase the number of route types available to each one of the domain members. So they may have route types such as US-Europe, US-Asia, Europe-Asia, US-UK etc. The negotiation output would be essentially the sharing of specific routes for these route types. In this particular example we have six types of route between three domains, where each domain controls a subset of those six route types. Eventually we will end up with having routes shared among the three domains which these domains did not have before.

So route sharing essentially means a user or administrator of a domain assigning privileges to users of these other foreign domains for use with route applications which for example execute reservation billing, communications and so on. So we want to be able to do the assignment of shared objects to various domains. Suppose that we have the following example. Domain 1 controls routes of types 1, 4, 5 and 6 where these route types are numbered or identified. Domain 2 controls a different subset of these routes, and Domain 3 controls yet another slightly different subset. Of course some of the route types are already shared, and what we'd like to do is to increase the overall value of these companies, or profits of these companies[1], if the companies assign monetary value to these routes.

The global constraint that we have here is that we'd like to do the sharing of these route types in such a way that we negotiate the fewest shared routes. For example Domain 1, which has route type 1, may have 5 routes of that type. Domain 1 would have route type 4, 8 routes of that type, and so on. So if we apply this global constraint that we might want to call the "least privilege principle application" we end up with Domain 1 sharing route types 1 and 6. So Domain 1 makes available routes of type 1 and 6 to the others, Domain 2 makes available route type 3, and Domain 3 makes available route types 2, 4 and 5.

We can actually have other global constraints. In particular, in terms of administrative access to this route type, we may have some form of separation of duty.

One thing that we tried to determine is whether there are some theoretical settings for these kinds of negotiations, for selection of these common objects. It turns out we could find game-theoretical settings that enable us to come up with the application oriented rules for the sharing itself. The sharing mechanically occurs just like in any other operating system — in other words, either the domain itself, some user in the domain or some administrator assigns these privileges to objects — but the rules based on which this sharing is done clearly depend on the application at hand. For example, in this case the coalition goal that increases the value of the coalition might be to minimise maximum dissatisfaction of each domain. In other words, each domain may have some individual goals, or a set of goals, and given the resources of the objects that the domain has it may not in fact be able to achieve all its objectives. By negotiating accesses to objects of other domains it might increase the value and decrease its dissatisfaction.

In this case we have just 3 types of routes, which makes life a lot simpler. The 3 types of routes which are shared between 2 domains and the objectives are very specific. The main objective is value, so again we have this constraint and here is an example of the actual state of each domain. Domain 1 may have Objective A, which has 3 route types and multiple routes of each route type totalling a value of 100. Objective B is to have say 7, 10 and 6 routes of the 3 types totalling a value of 80 and we may have Objective C that has a value of 95, again having different types of routes for different routes for these 3 different

[1] Bearing in mind Warren Buffet's calculation that the sum total of all airline profits since the birth of the industry is zero.

types. However, all these objectives cannot be satisfied simultaneously because this particular domain has only 20 routes of type 1 altogether, 20 routes of type 2 and 11 routes of type 3. So the total value of the objectives is simply the sum of the individual value of the objectives. The maximum achieved value, under the routes available, would be about 180. Maybe that satisfies Objectives A and B, but not C.

The idea is that this domain may want to negotiate with some other domain for use of these routes, either to obtain the satisfaction of the next objective, maybe Objective C, or to actually have a different mix of objectives at a maximised value. Right now it cannot really have either of those alternatives satisfied. Domain 2 again wants to achieve Objectives D and F, and just like before we have a setting of route types and routes for each one of the types. We have the routes available to Domain 2, which clearly indicate in this very simple setting that the maximum achieved value is based on Objective B and Objective F but not on Objective E. So now what we do is use some very straightforward results of game theory. These say that the goal being achieved is the total value achieved by Domains 1 and 2 by minimising the maximum coalition dissatisfaction. We obtained a solution for the nucleolus that is a vector solution that minimises the maximum dissatisfaction for all domains. Most of the time the solution exists, it's the only likely solution if negotiations of this type are undertaken, and the solution is efficiently achieved by linear programming. Also we can also determine whether the nucleolus exists.

In this particular example, applying this straightforward theory, we find that Domain 1 should in fact share 1 route of type 3 out of its allocation and Domain 2 should share 1 route of type 2 out of its allocation, and by doing so we achieved a maximum value for both domains that significantly exceeds the previous value. What we have done is minimised the maximum coalition dissatisfaction. For example the dissatisfaction of Domain 1 is now that it couldn't achieve Objective B. It can achieve Objective A and C but that has a higher value than A and B being satisfied. Similarly for Domain 2 dissatisfaction, we cannot achieve Objective F but we achieved Objectives D and E that has a higher value that the initial value before negotiation.

We notice here that there is another solution that has the same maximum value and the same value of the coalition dissatisfaction. So our first solution is not unique, but in this second solution there is more sharing, so it doesn't really satisfy our least-privilege concerns, namely to share the least number of routes that would maximise the value. So essentially we reject this second solution.

In this example what you notice is that the decision as to what to share, and how much of it, is based on the straightforward application of the theoretical results. However this is not the only type of negotiation that you have when you try to establish this common state. In particular we may have negotiations with local constraints. Now what would these local constraints be? Well, here is an example that illustrates the notion of local constraints. Local constraints are like the "privilege-principle": applicable to all the domains, but applied strictly to an individual domain. A local constraint might be that the domain is not willing

to share more than half of the routes it has. Furthermore some of these local constraints may not necessarily be revealed. So the existence of local constraints, some of which cannot be revealed *a priori*, leads to bargaining. In other words, each domain would come out with some sort of a preference list of the common state, a set of objects in the common state, and of course the privileges, and would post that list of desirable objects to each domain which has access to the negotiation server. Everyone does the same thing and then the negotiation server tries to come up with a solution that would satisfy everyone. Needless to say the solution may not exist, in which case we go back to the bargaining table and issue another preference list.

Going back to our example, we have specific route objects that are partitioned. Domain 1 controls routes 1, 2 and 3, which are not shared with anyone at this point. Domain 2 controls routes 4, 5, 6, 7 and 8, and Domain 3 controls routes from 9 to 12. The local constraints might be, as mentioned earlier, that Domain 1 is willing to share half as many routes as other domains because it may have some geographical advantage and these constraints may be revealed. Domain 1 makes this public to the others. Domain 2 is willing to satisfy some of the dictatorial requirements of Domain 1, but is unwilling to share route 6 that it has and it is unwilling to reveal that route 6 is a non-negotiable object. Domain 3 again agrees with the dictatorial decision of Domain 1 and it needs one of the routes 2, 5 and 6 and is unwilling to share route 11 and it cannot reveal this. These are the kinds of constraint that we call local constraints.

An example of what may happen here is that if Domain 1 proposes a particular setting of route sharing, namely that it is willing to share route 1, and it wants Domain 2 to share routes 4 and 5 and Domain 3, 9 and 10, everybody agrees that that should be a common state because it satisfies all the local constraints of each one of the routes. In particular, notice that Domain 1 shares only one route and everybody else shares three. On the other hand, suppose Domain 3 initiates a negotiation by proposing the following route sharing state: Domain 1 will share route 2, Domain 2, routes 5 and 6, and Domain 3, routes 9 and 10. This is unacceptable to Domain 2 because there is this magic route 6 which Domain 2 is unwilling to share and is unwilling even to say that it is unwilling to share!

Of course what will happen is, if you negotiate long enough, everybody would know everybody's secret local constraints. But anyway, Domain 2 rejects this proposal by Domain 3, it proposes its own set of routes, so notice that both Domain 2 and 3 agreed with Domain 1's dictatorial imposition, but now Domain 2 evidently asks Domain 3 to share route 11 which is a no-no and Domain 3 rejects this and ends up proposing an alternative which is this case fortuitously everybody agrees to because of all the local constraints are satisfied. Notice however that this process may not necessarily stop, it may not necessarily converge on a solution.

So the question is, when will this happen and how do you conduct negotiations so as to avoid this? Well one thing that we already know is that if the domains have ordered preferences, in other words the preferences within a

list are ordered by some criteria, and if the domain preferences differ, then the negotiation server that implements this arbitrator, or selector of the optimal solution, takes this preference list and tries to output another preference list which is pareto-efficient, non-dictatorial and independent of irrelevant alternatives. It turns out that no such arbitrator exists in theory or practice.

What is a pareto-efficient set of preferences? That's a solution in which if you make one of the parties better off, you have to make some other party worse-off. Non-dictatorial means that no one player, or no one domain, can impose his preference at the expense of all the others. We have seen that our previous solution was in fact dictatorial, because Domain 1 imposed the local constraint that it's willing to share only half as many routes as the others. Independent of irrelevant alternatives says that the solution of set objects or values to two domains A and B really depends only on Domain A and B. In other words there are no other sets of alternatives in the solution vector that affect the allocation of values to A and B.

Kenneth Arrow showed in the early 1960s that no such arbitrator exists under all possible patterns of domain preferences. So as a consequence, as indicated before, negotiations may not conclude without dictatorial preferences, as in case 1 where Domain 1 imposes its preferences. This essentially tells us that a dictatorship occasionally helps.

The other aspect of policy negotiation refers to a negotiation of policy models, or actually of an interpretation of policy models. So, for example, you may want to negotiate access management properties, namely properties under which domains are assigned membership to rôles. All domains, when they decide to share these objects, will follow the same rules in assigning local users of domains to membership of various rôles. So that means that we are actually negotiating the same thing; it's an example of negotiating the same access management properties. Again this could be done with or without constraints.

Furthermore you can negotiate access authorisation properties, and this is an area that has been looked at by various people. Imagine that you have a client and a server who know nothing about each other, and this client wants to access some of the objects in the server. The client issues an access request, the server responds, not to say yes or no, but with an access authorisation rule stating what type of credentials the client needs to execute the request successfully. The client complies and sends a set of credentials along with the access request and at this point the server replies and says, yes this looks OK to me, I'm willing to grant you access, or may say, oh by the way if you really want this option then you need to provide more credentials.

The idea is that there is partial disclosure of the access authorisation rules in the server, there is also partial disclosure of the client's total set of credentials. The reason why this happens is because the server is unwilling to disclose perhaps privileged relationships with other parties. In other words the server has a bunch of access rules, some of those are privileged, just like a trusted process, or a privileged process, has privileged accesses to various objects inside an operating system and unprivileged accesses. Now in this case the privileged relationship,

or privileged access, might be a privileged business relationship that the server is unwilling to reveal until it finds out who the client really is or who the client really isn't, and what kind of relationship the client has. Similarly the client is unwilling to disclose all the credentials up front because again, the client wants to keep certain associations private or to protect sensitive private information. For this reason they have to negotiate the particular set of rules necessary for access.

There are further examples of this sort of thing, such as access management rôles if a domain supports selective revocation of identity certificates. In particular, an identity certificate may have dependent identity certificates and if the identity certificate is revoked, it revokes the dependent set of identity certificates. How you enforce that rule is up to you. Some people prefer cryptographic linkages of these dependencies, as I mentioned last year[2]. Some other people prefer system solutions. In fact, we prefer system solutions. Again, this is something that we can negotiate, which way is better.

The reason why we prefer this system level solution to enforcing these dependencies to cryptographic solutions is the following: suppose that whenever an identity certificate is revoked then the dependent certificates are revoked at the system level. What that entails is the following authorisation: when you have any access request involving the identity certificate and the attribute certificate, you only need to check the validity with respect to the revocation standards of the attribute certificate. So that means that instead of verifying the validity of the identity certificate upon which the access control certificate is based you simply verify only that the access control certificate has not been revoked. This saves revocation standards verification. At this point we haven't quantified how much performance this buys, but we suspect it buys quite a bit because these kinds of actions are actually very common, unlike these revocation actions which are quite infrequent.

So we've decided that the distribution of complexity in enforcing this dependency check should be at the system as opposed to the access authorisation level. We enforce this dependency at the access authorisation level if we enforce a dependency cryptographic link, meaning linking cryptographically those certificates. In the access management properties that follow we can assume any one of these alternatives and we obtain some sort of a negotiation. In other words, if we all agree that we do this dependency enforcement at the system level then we all gain by not doing this extra verification of the validity of the identity certificates.

To summarise, what we've tried to do is to come up with some sort of a framework that explains what we mean by negotiation of access control policies. What we are doing now is actually to implement a negotiation server that will enable us to at least illustrate how some of these negotiations are performed in a real setting on a real network. Also we want to try to show the benefits, if any, of negotiating some of the access control policy properties, as opposed to just the common state which this policy membership should have.

[2] LNCS 2133, pp 100–124.

Roger Needham: I don't find it easy to grasp how much the various players have to have in common in order that this can work. The example you use is revealing because all airline companies have basically the same objective, although they really want to do it in slightly different places. But in the other context where you want to have negotiated access arrangements, then the objectives, the methods, the internal procedures of the players, are much more different than that.

Reply: Very good point. So the question is how to identify these common objectives. First of all we know one thing, that there are some common objectives. We don't need to form a coalition if there are no common objectives, so at least we are guaranteed that these exist. Identifying them may not be as obvious as in this example, you are absolutely right, in fact there may be cases in which we have a bunch of common objectives which are identifiable and then we have private objectives which are not identifiable. Now the question is, can we actually cast these common objectives in terms of specific applications, specific objects that the applications will have access to, and values that apply? For example, game theory. It remains to be seen, but it's quite clear that given the fact that the coalition would have some common objectives, there would be some common applications which are shared, and some objects which become shared as a result of the coalition set-up. That is not particularly under dispute because otherwise there is no sense in forming a coalition. I mean it follows that that ought to be the case.

Now let me go further in that direction. Suppose that in fact the commonality of objects happens to be only fairly ephemeral such as communication mailboxes for say multicast. We certainly need in coalitions the notion of multicast or the notion of broadcast, so it may be the common goal gets reduced to the ability to communicate securely using multicast and to allow fairly frequent member join and departure. That in itself might be the coalition rôle and we still have to have some shared objects and of course shared applications, and that would not require game theory to set up. Nevertheless it can get quite complex, particularly this integration of multicast with access control and the notion of wholesale distribution and revocation of privileges and keys when members join the multicast group and when they leave the multicast group.

So in some sense I agree with you that the objective may not be obvious but there always are some common objectives.

Roger Needham: I've recently encountered several instances of a slightly surprising character, of systems that don't work because of the lack of means of doing what you've been talking about. One of them was aid agencies in Sierra Leone who have an interest in their staff not getting killed but they have communication difficulties because their systems won't work. Another one is conducting peacekeeping operations in Kosovo, where the constraint is actually the vastly varying technological capabilities of the players not the absence of a common goal.

Reply: Exactly, so it is in East Timor for example. The US Pacific Command that came into the news because of the summer in Greenville. Few people know

that most of their operations are humanitarian operations. They cover a large percentage of the entire globe, and they have to form coalitions with all sorts of other partners and they have to form them dynamically. They are not made once to last forever, they form a coalition with Australia, they form a coalition with Korea, they form a coalition with Taiwan, maybe one with China, and not all those members trust each other at all times. But the example that's more recent that I thought you might mention was the coalition that we had in November, December and January, in reviewing the Oakland submissions, namely the coalition formed by the Microsoft Conference Management Tool. This clearly acts as some sort of a negotiation server, perhaps not as sophisticated as this one, but you certainly were given a maximum value that each one of us, the reviewers, could achieve and we were constructing a set of preference lists of these papers and we assigned values to all these papers and the conference management tool was supposed to come back and tell us exactly what papers we got and with who else we shared those papers.

Roger Needham: So you got all of Microsoft papers then? [laughter]. Another example of the same thing occurs in the construction industry where you have an *ad hoc* coalition, an architect, a general contractor, a young structural engineer, a quantity surveyor, and so forth. I've talked to the construction industry computing association and they're very conscious of the sort of difficulties you've been talking about.

Reply: Yes, it turns out that you also have this for car manufacturers, so you have quite a few of these examples, including banking as well. Of the problems that appear nowadays in practice, one is the problem that Roger mentioned, the use of widely disparate technologies. But using the same technology doesn't necessarily solve the problem. In particular, one of the major difficulties that people find in administrative private practice is actually setting up new accounts for users. It turns out that in banks, to set up a new user account allowing that particular user to use the application in his or her job responsibilities, takes between 10 days and 3 weeks. Most of the departments in the US government doing the same thing take at least 3 weeks. So one question is, why does this happen? At least from the point of view of privilege distribution, it's fairly clear that the products on the market are extremely expensive and extremely specialised to use. There is no tool to do wholesale distribution of privileges or wholesale revocation of privileges, and furthermore, as Roger pointed out, there is no tool to do a "before-the-fact" audit to find out what are all the privileges that I have in the system and so enforce separation of duties. So these administrative problems cause tremendous difficulties in setting up this form of coalition. But we see more and more of these coalitions appearing in practice and we notice the difficulties of setting up coalitions. Of course I have addressed only a small problem; there are several other problems to be addressed, in particular the problem of integrating access control with authentication infrastructures and integrating this with multicast over the Internet, because we really want to use a common technology with well understood and hopefully well liked technologies.

William Harbison: Going back to what you and Roger were saying earlier, about common objectives, I can conceive of a situation where there is no common objective, or rather more precisely the common objective does not involve one of the parties. For example, in the airline case you gave, one could possibly think of airline A wishing B and C to join it in a consortium because they want to stop B and C joining another consortium. The common objective may just be that B and C join a consortium but A may not have any other motive. B and C joining A doesn't necessarily mean that they are seeking to add value to A in that sense.

Reply: If their reason is to stop them from doing something else, then I think that the undesired outcome would subtract value from A. It would have a negative value. So there has to be net value for all, otherwise they wouldn't agree to join.

William Harbison: But they may not know that that's A's motive.

Reply: Ah, that's true! In other words A's local constraints may make it impractical for A to reveal the reason why and the total value that it gets, and by the way this is one of the reasons why some of these negotiations might not succeed or might not end.

Mark Lomas: You were saying that if you had enough rounds of negotiation then the parties would learn what their secret constraints are but I can imagine a situation where they don't. Let's take one of your routing examples. I don't want to tell anyone that I'm not willing to give up route 6 but the moment somebody proposes a solution that involves route 6 I have to give a plausible explanation that's consistent with my constraint but doesn't reveal information. So I'm likely to invent more constraints that I didn't have and then I have to make sure that I get a sub-optimum solution.

Reply: True, and the only comment that I have is to suppose that we found out enough routes for it to be obvious to everybody that I have all these wholesale reasons why I don't want route 6 to be shared. At that point it is clear that I don't want to share route 6.

Mark Lomas: But I could produce more reasons, or a wider reason and incorporate it. For example I say, I can't release 4, 5 or 6. My real constraint is I can't release 6 but I don't want to tell anybody that, so I'm actually willing to lose value on the other routes.

Reply: Well in that case what will happen is that negotiations terminate very fast but my suggestion was different. My suggestion was that if the negotiation goes on long enough then you'll find out. If you mask why route 6 would not want to be shared and you mask it by saying I don't want to share 4, 5 and 6, then the constraints are so severe that it will terminate very quickly. Indeed I will not have that information that you don't want to share 6, but my statement was different, if you negotiate long enough eventually more of the constraints become fairly obvious.

Michael Roe: Basically if you enumerate all possibilities and ask every party yes or no then you build up a complete picture.

Bruce Christianson: But Mark is right: if some parties are willing to accept — and indeed negotiate hard for — solutions that are sub-optimal for them, then they can conceal their real constraints.

Mark Lomas: But this might actually depend on the order in which they do their negotiation.

Reply: Let me give you an example to make your point, Mark. When I was a graduate student at Berkeley, we had qualifying exams and in these exams we were supposed to meet with eight professors in their offices at different times in four days. So half an hour with A on Automata Theory, half an hour with B on Finite State Machines, and so on. We were given the choice to select the professors whom we wanted to see. So it turned out that you are better off, because you are more likely to get your choices, if you listed sub-optimal solutions, because the optimal solutions in this case happened to be known to all. In other words there were some professors who were easier on graduate students than others, so you were better off listing those on the next level of difficulty, and getting them all, to make sure you didn't get the worst. By attempting to get the sub-optimal solution, we are hiding the fact that we didn't want to get the worst, and in fact it worked out quite fine.

David Wheeler: Your constraints are static. What happens if they dynamically change with time?

Reply: Clearly they *will* dynamically change in time and then, eventually, you will have to conduct renegotiations. Unfortunately that's more complicated because everyone has to agree to conduct these negotiations at the same time. In other words, everyone has to go to the negotiation server with his or her preference lists

David Wheeler: Yes, but this is a method of concealment of course.

Reply: Oh yes, definitely.

Intrusion-Tolerant Group Management in Enclaves (Transcript of Discussion)

Hassen Saïdi

System Design Laboratory, SRI International

This talk is about intrusion tolerance, and the big idea, the main idea, is that instead of having a one hundred percent security guarantee, you'd like to allow certain attacks to happen and certain parts of the system to be compromised but that this shouldn't compromise the mission of the entire system. This work has been done with my colleagues Bruno Dutertre and Victoria Stavridou.

Enclaves is a lightweight framework for group management. The idea is that you'd like a group of people, perhaps belonging to a certain organisation, to get together and to share a certain application, over a network or infrastructure that is not secure, such as the Internet for instance. But you'd like them to share this application in a secure way so only members of this group should have access to certain information or share the application and you don't want someone who is outside the group to interfere with the activity of the group or, for instance, to know what kind of information is exchanging between the members of the group.

The Enclaves idea was first formulated by Li Gong[1] and there was an early implementation of the system. Their design idea was to have very lightweight frameworks, a very lightweight software solution to implement group management. The main feature of the system is to provide secure communication and a group management service, ensuring that the communications are private among the members of the group and that, in fact, only people who are allowed to be in the group are admitted in the group and can share the application.

A typical Enclaves demo would include things like a whiteboard, or people sharing a file working on some document, or a bunch of engineers working on the same system, or trying to encode or work on some program, that is private to the group, or private to the organisation, to which they belong. The architecture is the following, you have a session Leader and an arbitrary number of members who can contact the Leader and can be for instance admitted in the group and can participate in the application.

The Enclaves software is distributed among the members and the Leader, and if you have a certain application on the top that you want to share, Enclaves provides an API so you can send whatever information you'd like to share to this API. That ensures the group management and the communications are done according to the application, and has a layer of authentication and encryption that ensures the confidentiality of the communication, and ensures also that

[1] Li Gong, "Enclaves: Enabling Secure Collaboration over the Internet", IEEE JSAC 15(3) (1997), 567–575.

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 213–216, 2002.

only members that are allowed to participate in the applications are admitted. Enclaves itself uses the common API to send messages through whatever infrastructure or network you are using, and the Enclaves software is written in Java so you can basically use it on different platforms.

James Malcolm: Is the common API, the lower API, defined as part of Enclaves?

Reply: I guess you have to change a few things if you have some particular infrastructure that is not common, but if you use say TCP/IP then it's pretty standard.

The previous Enclaves protocols that allow an arbitrary member A to communicate with the Leader have four main parts. The first one is sort of pre-authentication part where a member sends just a request for connection and the Leader can decide to allow the member to connect or not. After that there is an authentication part where the Leader sends to the member a group key that it can use to communicate with the different members. The third part is basically regular communications related to the application, and then finally a member A can at any moment decide to disconnect and to leave the group.

Michael Roe: Can I ask a question about the message by the green arrow. That message is signed with the Leader's public key, is that right? I'm thinking that in the second message you're sending the session group key in the clear.

Reply: No, this one is not sent in the clear, but this one is, and these two part-messages sent in the clear, that's actually the problem. You see, we looked at an early implementation of Enclaves and basically this was extracted from the implementation. I've taken the Java code and extracted the protocol, and that's what we wanted to analyse. We took the implementation and it turns out that it had obvious flaws and that's what we wanted to show. It turns out this wasn't actually signed.

So what were the security requirements? The first one is to allow the members and the Leader to authenticate each other and that turns out to be a problem in this particular implementation and the previous protocol. The main problem with it is that it's not intrusion tolerant, in the sense that if a member is compromised then it can actually disturb the group membership apart and even compromise the confidentiality of the communications. A very simple replay attack consists in sending these kind of messages whenever a certain member tries to send the first message to the Leader, tries to establish a connection. So anyone can send this kind of message and create a denial of service attack and prevent a certain member from access to the group.

Another problem is the freshness of messages: there was a part that was missed in the previous protocol when a member leaves the group. The Leader sends to the other members the identity of the member that just left and informs the other members of the group so they can update their membership list. It turns out that this message could be sent by an existing member, to convince another member that someone left the group.

Another problem related to the freshness of messages is that again when a member leaves the group a new group key is issued. An old member who left

the group can now use an old key. So, say an old member B sends an old group key, it can convince a member, an existing and legitimate member A, that this is the new key. Now any message sent by A to the group can be decrypted easily by the intruder who is in this case was a former member of the group.

We want to move from the view that we'd like to protect the members of the group and the Leader from outside attackers and where every member of the group should be trusted to a maybe more realistic view, where you allow certain members to be compromised. The main idea is that this shouldn't compromise the activity of the group, and shouldn't compromise the confidentiality of communication between the Leader and the legitimate members.

So we did a few improvements to the protocol by removing some of the messages that were sent in the clear, and also solved the problem of freshness of messages. We also got rid of the group key, so mainly now a member who wants to join the group uses a certain key, a long term secret key shared between the member and the Leader — actually that was also part of the previous protocol after the pre-authentication part — so he chooses the private or shared key PA which can be for example a simple password known to A. Then A sends that with a nonce, and then the Leader sends back a new session key and a nonce, to show the freshness of the message and at the same time acknowledge the first message, and then the member acknowledges the receipt of the new session key. Another thing is that when a member wants to leave it just has to send a request and doesn't have to wait for any message from the Leader.

The new protocol ensures proper user authentication between the member and the Leader and also proper distribution of group management. We made sure that the messages received by a member had been sent by the Leader, and that they are received in the order they were sent. So there is no duplication of messages and so in the new system, you have to trust the Leader but you don't have to trust the other members. So if you receive all the messages from the Leader then you have that certain properties are ensured.

So as long as the Leader is not compromised these properties are guaranteed. We formalised the protocol in the PVS theorem prover, and we verified these properties in PVS, and we did a new implementation of the system that has the new property.

A few words about the analysis of the protocol itself. We used Paulson's approach to model the behaviour of nontrusted agents, so you consider the executions of the system as traces of states and you try to characterise each state you have by the knowledge of the nontrusted elements and what kind of information they can gain and so on. And the global model consists of the Leader and an arbitrary trusted agent and an arbitrary compromised agent. So the properties to ensure proper authentication and proper group management were started by two simple properties to prove, which are the secrecy of the long term key (the password that is used to join the group) and the secrecy of the session key used by each member. And we used that to ensure that the authentication protocol is correct and that the group management part is correct.

Something about the analysis of the group management and the authentication part once we modelled the protocol. If you want to perform systematic verification or more automatic verification, sometimes it's very hard because these kinds of protocols have multiple instances. In this case also they have an arbitrary number of members and sometimes it's very hard to do this kind of verification in a fully automatic way because you have basically an arbitrary number of cases or arbitrary number of states that the system can be in. But the main idea that we use here is the use of abstraction, that in a formal way reduces the problem to a finite instance. We constructed in a systematic way this abstraction which shows that the protocol cannot be in an infinite number of states and you can analyse this finite number of states and derive all the security properties that you would like to prove.

We also did some limited experiments with model checking, basically that was to find all the flaws in the original protocol. We took a finite number of members and we ran the model checker, and using simple replay attacks you can find easily all the violations to the security properties.

I didn't go into details of exactly how we used the Paulson approach. That's in the full version of the paper[2] and it's quite easy to read, and all the details of the proofs are there.

What's next? We moved from this idea of having an attacker outside the group to the idea where we allow certain members of the group to be compromised, but of course there is still a weak point, a single point of failure, which is the Leader. So we moved to the obvious next step, which is to have hierarchical Enclaves, or distributed Leaders. So we allow different Leaders and we allow also some Leaders to be compromised, and that also led to all sorts of new interesting research ideas in the context of group management and applications to things like PVS and stuff like that.

Larry Paulson: I'm gratified to see other people verifying protocols, even if you happen to use the wrong system. [laughter]. I know John Miller had started doing hand proofs so I wondered whether he'd given up on PVS or not.

Reply: No, the hand proofs now are done in PVS. But we used also our own recipe, the abstraction part and stuff like that. We have some experience in using this technology and usual formal methods, verification of software and things like that, and that works fairly well.

I know last year John Miller had a paper, for proving secrecy we used his method that was basically hand proof, but we did it in PVS. And his method actually used your methods, and at the end we used the abstraction and stuff to prove all the different lemmas that are generated.

[2] Bruno Dutertre, Hassen Saïdi and Victoria Stavridou, "Intrusion-Tolerant Group Management in Enclaves", DSN-2001: International Conference on Dependable Systems and Networks, Sweden, July 2001.

Lightweight Authentication in a Mobile Network (Transcript of Discussion)

James Malcolm

University of Hertfordshire

Bruce does not entirely approve of reports of completed work, and I have no relevant work in progress, so this is the third possibility: work that hasn't really started yet.

I would like to start with a couple of observations. The first is that in a mobile situation, generally speaking a mobile device has a home network which it returns to. This idea is not applicable if the mobile doesn't actually ever return to its home network. When it goes home, when a mobile returns to its home network, then it can be in physical adjacency to the home agent, and if so, it's possible for it to securely share lots of key material. You can think perhaps of a student going home at the end of term; delivers all the dirty washing, collects clean washing, and then goes on its travels once again.

Related to that, the authentication in a mobile network is between the mobile and the home agent. There will be other authentication as well, the application on the mobile device may well want to authenticate itself to a server somewhere, but I'm just thinking about the network level authentication between the mobile device and the home agent.

The second observation, or assertion, is that to prevent denial of service attacks we don't need particularly strong security. Certainly if we have a targeted attack, if we're trying to pick on one individual, then security needs to be rather stronger, but if we're just trying to deal with general malicious denial of service attacks then quite modest authentication between the mobile and its home agent will be enough to prevent such an attack.

John Ioannidis: Are you worried about denial of service against the mobile or launched from the mobile?

Reply: Denial of service from somewhere else in the network saying the mobile has moved over here when in fact it hasn't; trying to ensure, at the point of mobility, that we don't have people able to pretend the mobile has moved when it hasn't.

Now linked to these two observations, we have the requirement that for real time media applications, delay is bad. So can we reduce the round trips for authentication to zero? Do we need to have a challenge response protocol to authenticate?

If we go back to the first observation that the mobile can go home and share lots of key material, we could imagine having a one time pad and every message that the mobile sends could have perhaps just 16 bits of one time pad. It would provide quite good authentication if every single packet sent contained just a small number of bits.

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 217–220, 2002.

Tuomas Aura: It's authentication against messages sent from elsewhere in the network, but not by someone who can listen to the traffic that the mobile sends and intercept the packet and move its bits to another one.

Reply: If you have a one time pad you can observe the traffic up until this point — you don't know what's coming in future. But if you can intercept the packet this doesn't help.

Tuomas Aura: Instead of a one time pad you can use hash chains. The nice thing about hash chains is that you don't need to have any extension headers or anything like that there, because you can actually put them in the host part of the IP address.

Reply: You're talking about IPv6 where you've got plenty of address bits, yes, good point, and in IPv4 there's the packet identifier in the fragmentation field, so there's 16 bits you could use.

So the basic idea is perhaps we can put authentication in every packet and then we don't need exchange and a challenge response to authenticate, every packet has its authentication information.

Maybe a one time pad is not realistic, but only a small number of bits is needed in order to prevent a denial of service attack where somebody says this guy's moved when in fact he hasn't. So basically if we use a hash chain, or some kind of cryptographic random number sequence, then we don't need to use the whole random number, it can be as long as you like, we only need to reveal a relatively small number of bits. It's still possible to cryptoanalyse and work out what the key is but it's not easy.

So for the client mobile, each packet requires one encryption. At the home agent you would have a window of the next 16 acceptable values perhaps, however many packets you might think might get lost, you need at least a window of that size, but that window can be maintained and just increased when the home agent's got spare time.

So, fairly easy to attack, but you do need to gather a lot of traffic. I think this forces the attack to be targeted on a particular mobile and it certainly puts up the cost of doing a denial of service attack on a mobile and it doesn't require any on-line key exchange. And that's it basically.

Tuomas Aura: So you'd have these bits in every packet?

Reply: That's what I had in mind, yes.

Markus Kuhn: How about SecurID[1], because that's almost the exact same protocol, it's just time related but the patent also covers other threats.

Pekka Nikander: The idea that Tuomas and I had yesterday for IPv6 is that maybe you don't even need any keying material. You create a chain of hashes and then you use them in the reverse order. When you send the first packet and start to communicate with somebody you just include the last of the hashes in the IPv6 host address, and whenever you change your prefix you take the previous value from the hash chain so that the recipient can see that probably this is still you, because you knew the previous value of the hash chain.

[1] Mark Lomas, these proceedings.

There are of course some problems; if somebody can eavesdrop the messages and maybe get, intercept, and so on and then it doesn't work anymore.

Tuomas Aura: It protects against most of the script kiddies in a random place ...

Bruce Christianson: Similarly, you use one of these things like the Guy Fawkes protocol where in each message you include something which depends on a nonce and the hash of the next nonce and then you put a hash of that whole thing into a message then the next message must have the nonce corresponding to the hash so there you don't have a fixed number of hashes in advance.

Tuomas Aura: Can you do that? It means you would need twice as long space for this. The problem is, we only have 64 bits.

Bruce Christianson: Oh sure, eventually you either run out of bits or you have to share key material[2].

Larry Paulson: The problem with the hash chain is if you've got one value you can predict all the rest; but you could combine the two ideas, using hashing to stretch out a one time pad so that it lasts a lot longer — so you have a hash chain in which you include say 16 bits of one time pad as a kind of salt and I'd guess that would be fairly hard to attack.

John Ioannidis: I'm fairly sceptical about things that depend on the strict ordering of hash values because there is a lot of packet reordering in any real network and lots of packet losses, and even if you say that you are going to buffer everything and you get enough packets to reconstruct the chain there are many applications where out of order delivery of packets is perfectly acceptable and then you are severely restricting the kind of acceptable packets that you can use.

Reply: Yes, my proposal doesn't require the packets to arrive in order but just that you've got a window long enough to observe that a packet has come from the right source.

John Ioannidis: Right. But before you actually use the packets in the high layer application all of them have to have arrived?

Reply: No. The two ends can independently work out the authenticator on the next twenty or even a hundred packets.

John Ioannidis: Well if a packet arrives out of order can you use that up to higher application. How are you going to know that it's authenticated if you have missed ...

Reply: Because both ends have the one time pad. I mean, it's just an encrypted form of counting; if you send me packet one and I get it and then I get packet seven because the next five have been lost I can still use packet seven, I know it's packet seven because it matches the seventh segment of one time pad.

Tuomas Aura: But with a hash chain, if you get a hash of a packet, let's say three packets ahead ...

Reply: You just do two more calculations and remember the results for the ones that come in later.

Tuomas Aura: You need to use a few bits for a serial number.

[2] For example, if the initial shared secret is S_0, then the authenticator for packet i can be $h(0|S_i) \bmod 2^{64}$ where $S_{i+1} = h(1|S_i)$. See LNCS 2133, p 187.

Markus Kuhn: There are some very related protocols with authentication in car keys, where it's too expensive to have a challenge response. So you want to press a button, and each time you increase a counter. Then you send out the encrypted counter and there is a sliding window of permissible counter values. There is a whole range of mechanisms how you can recover if you lost synchronisation; in Ross's new book[3], the authentication protocol chapter has a list.

Michael Roe: I think authentication of the mobile to its own home agent is the easy part of the problem to solve, I think quite a lot of protocols we've mentioned can solve that. The hard part is when you use route optimisation, this is where you try to avoid going back to the home agent and the mobile talks to another node that had no prior knowledge of it. How do we do authentication in that sort of case, particularly if the mobile's been moving around and has changed its address lots of times. Suddenly the mobile talks to another host it's never talked to before and which has no record of any of its history or where it is in the hash chain, it just suddenly gets a message.

Tuomas Aura: Well the home agent here is a problem. If you didn't have a home agent then you are fine, if you only talk to a correspondent then all you need to prove is that you are the same node that started this connection and that sent the previous packets. But if you want to authenticate the mobile at a certain home, an address, then you need something else.

Bruce Christianson: If you imagine a protocol where effectively the home agent — possibly being tipped off that it's supposed to do it — sets up the two halves of the call, optimises the route and hands it off, then there's no problem.

[3] Ross Anderson, Security Engineering, John Wiley & Sons Inc, April 2001.

Bluetooth Security — Fact or Fiction? (Transcript of Discussion)

Peter Drabwell

Nortel Networks

I guess that this talk is probably appropriate for this workshop, as it is more a collection of ideas related to Bluetooth application concepts that I have worked on in collaboration with Bill Harbison. What Bluetooth is trying to do is replace the tangle of wires around PCs, peripherals, PDAs and cellphones, typically RS232 or 10baseT cables, so the focus really is on interconnect for peripheral devices. However, as the technology becomes pervasive all manner of application ideas are cropping up. I do not intend to go through the finer details of the Bluetooth protocol, I'm basically here to talk about some of the application ideas that have been proposed. Some of these suggestions are a little extreme, but I'll try to concentrate on one of them, namely a LAN access point scenario, and the required security aspects in designing/deploying such devices.

The intention of Bluetooth is to be ubiquitous and this relates to the pricing of the unit. I think a target price is $5 per unit for this: a price in this order of magnitude is fairly crucial to the widespread take-up of the technology. Initial products are pretty expensive. Ericsson has recently come out with some wire-free headsets for their mobile phones, but they currently cost over £200 in the UK. Only when these prices are reduced will we be able to envisage a wider take-up of such devices/applications.

Bluetooth can operate up to eight active devices in each network, and we can have 10 networks per coverage area. We have three power range levels, and operation is designed predominantly for internal use only, although some of the groups that we have been talking to don't appear to be aware of this. So they have been coming up with all manner of concepts, one of which was getting one's car tyres to talk to your car, and to report back the state of the tyres and/or condition of the road, even go as far as making individual car components that can feed back diagnostics.

Tuomas Aura: The original idea was to measure the pressure from inside the car and for a warning to light up if there was a problem.

Reply: Some are keen to take this approach with each individual component, even going as far as linking it to advertising whereby if a device such as a battery is running down, they could advertise the fact that you need to replace that component (with the appropriate brand, of course).

As an aside, one of the ideas that was proposed to us was to use a Bluetooth enabled mobile phone as your ATM input device. The notion being that this could add user privacy and prevent over-the-shoulder snooping upon entering your PIN at a conventional ATM. Using a Bluetooth enabled phone, it was suggested that a user be able to enter their PIN number, and request services such as transfers, chequebooks and statements. However a more problematic

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 221–228, 2002.

suggestion was made, whereby a user could make cash withdrawals from an ATM by using their Bluetooth enabled phone within the vicinity of an ATM. So we were presented with a scenario whereby a user could order cash withdrawals on their phone and then walk past an ATM machine where the money would actually be dispensed. There was all manner of argument over whose money is actually emerging from the ATM as you go by, but this is an absolutely genuine submission.

John Ioannidis: That's what worries me.

Reply: In a business proposition-meeting scenario, you have to maintain a straight face when people come out with such ideas, and discuss them in all seriousness, including the implications of doing such a thing. In addition to that there are some real Dilbert-style instances whereby they've seen Bluetooth, and as it looks exciting, any subsequent product must have Bluetooth in it, regardless. One of the key response questions to such proposals is to ask what it is doing that's unique over 802.11 or IRDA.

I suppose one of the advantages of Bluetooth is that is has some security element built into it as opposed to the mentioned alternatives. But is the security sufficient for the current need or current usage of the technology? As these ideas and concepts become more wild and varied in the Bluetooth environment, I think we have another example of what Mike said earlier[1] about certain protocols that traditionally address the fixed world, and as we move into a mobile world then the game changes rapidly. We have some initial security concepts that were meant to address the concept of a wireless peripheral scenario (your PC talks with the printer, talks to the scanner, etc), but then all manner of ideas are cropping up. I'll talk about one in particular that we were looking at which is the LAN access point.

One of the things that we also took a look at was the characteristics of lightweight clients/PDAs, such as the Palm Pilot. So we've got a small CPU which means that we have a limited amount of computing power to play with. As a PDA's processor is typically small, we have to look at alternative security protocols in terms of the computation times. Some potential examples, and the main one that we've seen was the Certicom VPN[2] using the new elliptic curve technology.

James talked earlier about denial of service attacks[3], and in one of Ross Anderson's papers they talk about Sleep Deprivation[4] because there's only so many times that you need to subject a battery powered device to an attack before it turns itself off. So there are latency issues that relate to battery power in terms of trying to conserve energy. The nodes are going to be off most of the time and they may turn their receivers on periodically. So communicating with such devices involves having to wait until the next wake up, and so we have really an ad-hoc network environment there.

[1] Michael Roe, these proceedings.
[2] `http://www.certicom.com/`
[3] James Malcolm, these proceedings.
[4] LNCS 1796, pp 172–182

LAN access connectivity is the application we were looking at, although we had a number of alternatives suggested to us: the mobile ATM idea was proposed in all seriousness.

Jan Jürjens: Sorry what do you mean exactly by mobile ATM?

Reply: The concept was that instead of actually going up to a bank Automatic Teller Machine and entering a PIN, you'd conduct the transaction on the phone and then go up to the ATM, using the phone to send the transaction details to the ATM.

Markus Kuhn: Where does the cash come out?

John Ioannidis: Maybe you download digital cash onto your phone!

Reply: This was exactly it, it was either that you used the phone as your cash card (to obtain physical cash) or as an electronic wallet concept. The first time they proposed the idea, it was implied that that the physical money would come out ...

John Ioannidis: ... of the phone?

David Wheeler: I like the idea of a little printer printing money in the phone.

Matt Blaze: Where do I get one of these phones?

Reply: Well, I was about to say, you're behind me on the list. So, you're waving a phone about and there's basically just cash coming out of some ATM within range of your phone. In light of which, I don't think they'd looked into this too deeply.

Michael Roe: It solves the false terminal problem, where some crooks put up something that pretends to be an ATM and it gets both your PIN and your card because you put them into it.

Matt Blaze: No it just replaces it with the terminal in the middle problem, which is that I have a Bluetooth repeater that appears to be a cash machine that is sending the message to the real cash machine that I'm standing in front of.

Tuomas Aura: You can put a physical device in front of the ATM that delivers your communication but keeps your cash.

Michael Roe: I can imagine some solution to this whereby you get a challenge on your phone which you then type into your keypad which says what the real ATM has on the screen.

Matt Blaze: But the attack is, instead of putting in a false ATM in which you insert your card, put in a false ATM that has has two Bluetooth devices back to back, one is talking to your phone, the other is talking to the real ATM that I'm in front of and is relaying the messages that are supposed to be appearing on the screen.

Markus Kuhn: It doesn't even have to be local, you can talk with an ATM 2 kilometres away using a microwave repeater.

Michael Roe: You need at least one channel that's not routed through any kind of network and which says that this thing physically in front of me, that's about to spit out the money, really is the ATM.

Matt Blaze: But if people could recognise a real ATM from a fake ATM, then the other problem wouldn't ...

Bruce Christianson: That's right, there's no problem to be solved.

John Ioannidis: On the other hand, we could use Geraint's suggestion[5] that we bound the amount of time that we are allowed to elapse between messages to within a nanosecond, which is the distance from the ATM.

Pekka Nikander: Yes, but that doesn't work with Bluetooth. That's one of the problems in Bluetooth: the message delivery times can vary quite a lot because of the protocol design.

Bruce Christianson: But in theory a very good method of ensuring that somebody is within 30 metres is to do exactly that.

Markus Kuhn: But Bluetooth has collision detection, so it's the same as Ethernet.

Bruce Christianson: Yes, you've got the same non-determinacy that we learnt how to deal with 20 years ago.

Reply: The more serious concept, a Bluetooth LAN access point, has a number of different requirements: Device authentication Bluetooth actually does take care of, the lower level details of this will be contained in a forthcoming paper. Bluetooth doesn't actually deal with user authentication, I think it leaves that to the application levels to take care of. Connection integrity, access control, again that's going to vary depending on what it is you are trying to do — what services you're actually trying to access.

The concept model was of a company conference room, or a meeting room, in which internal employees of the company, and external guests could potentially be involved in discussion and could connect to a network via a ceiling mounted Bluetooth LAN access point terminal. An example use case would be that I'm here selling certain components to Microsoft, and I've come in and I want to get an update on the latest price list. So I use my laptop to connect. In addition to these requirements availability is obviously a key one, although these are not in any particular order of preference.

There appeared to be a difference between the potential users of such an access point who were deemed "internal people", because there seemed to be an additional concept of trust applied to them, even though statistically most violations take place within the internal organisation. There were two different private areas outlined: If operating within a small room with a limited number of users it could be typically be arranged in order that we could set up the guest list, and/or the associated user access list for a particular session. This would allow a meeting organiser to actually book sessions and program them in such a way that from 10 to 12 we know that there will be five guests in that room, and we have the identity of those five guests that are permitted to use the network access service.

This could get pretty hard to administer, and again we are crossing the boundaries of what's secure and what's potentially feasible in terms of usability.

[5] Geraint Price, these proceedings.

In order to get around this, there is a suggestion that you have a host member present or a pre-registered token which you designate to be a chair token. If that token is not present nobody else can successfully access the point. It may well be that I have a specific identity in my Palm Pilot so that when I walk into the room and it acknowledges me, then my guests in that room can use the Bluetooth LAN access point.

The public area issue doesn't actually address that yet, obviously we're coming back into this ATM scenario where anybody can use the service, it's user initiated and as such this host member concept that has been talked about will not always necessarily be there.

This is pretty much a high level sketch: there is a degree of entity authentication and link privacy at the link layer but what Bluetooth doesn't address are the elements of user authentication, message authentication, which I think was touched upon yesterday with the SecurID issue[6] and a three tier model. The secure end-to-end approach is also not yet addressed.

So these are early days. In the fixed or static security world, there are protocols in existence that deal with this, but mobility is opening up all manner of issues and challenges that need to be addressed. Bluetooth seems to have seen this in a very small time-frame and initial entity authentication needs to be looked upon and added to with some user authentication and message authentication as well.

One of the other things that has been interesting over the last few days is the need to consider some lightweight client protocols, and the characteristics of a lightweight client which (at present) place process/power restrictions upon certain more intensive cryptographic and encryption methods. So a complimentary notion is possibly required whereby the concepts of a SecurID mechanism (for example) are used in certain cases.

The security framework itself isn't part of the Bluetooth core protocol suite, as far as we know, and therefore it's not specified to any great level of detail. As a result of this there are all manner of integral issues that could possibly crop up as different vendors find different solutions to these problems. There's also, in amongst the detail, the concept of a shared secret PIN that's used to generate keys for device authentication.

A specification allows the use of a zero link PIN and, as we talked about users opting for the path of least resistance, authentication link keys and encryption keys are going to be vulnerable because a number of users are going to use either very simple PINs or are going to object to using a PIN altogether. In addition to the concept of not having to actually have a PIN, there doesn't appear to be any specification as yet for distributing PINs, which is a potential development problem. In absence of such a scheme, the choice may be to develop proprietary PIN distribution software or PINs would have to be distributed manually, which could also lead to vulnerable PINs.

I think that's just about what we have for the time being. I'd appreciate any feedback or any ideas you may have.

[6] Mark Lomas, these proceedings.

Markus Kuhn: Why do you think there is actually a need for separate specialised security protocols for higher layers, and the things you mentioned on the previous slide? What's wrong with using the existing suite of IPSec, or SSL, or any of the usual things that we use today to talk securely to our laser printers, scanners, etc? If they have the ability to add security to the standard protocols stack, why not just treat Bluetooth as yet another OSI layer two protocol, and leave it at that?

John Ioannidis: "Not Invented Here". Why use Bluetooth in the first place? What's wrong with 802.11 power scaled down? Just use the same technology. Why invent something new with new security problems and a new idiotic link layer? We have enough idiotic protocols already, why invent another one?

William Harbison: I'm afraid it's too late for that now, unfortunately. The genie is out of the bottle.

Reply: I think this is a Dilbert type of scenario, whereby proposals are brought to us along the lines of, "Bluetooth is an exciting new concept, so let's use that", irrespective of the fact that 802.11 may be more appropriate for the project.

Roger Needham: But nobody needs to buy it.

William Harbison: This is true, but I think if you look at the marketing momentum that is behind it, it is probably inevitably going to occur. And as with all these sorts of fancy goodies, if it does occur then the applications that people will try to use it in will run far, far ahead of the capability of the technology to provide any form of secure communications. Honestly I can see that scenario occurring very, very quickly. It's considered to be, in certain circles, a very "sexy technology" at the moment. There are all sorts of people who are already looking to use it as an intermediate communication technology in a much wider broadband system, such as using it as a local collector perhaps and having much higher broadband communications at a concentrated level. I suppose you could say, let's ignore it unless it happens and let's try and make it go away. That is one approach. I think our approach is, it's probably going to happen, it's probably going to be pushed, so let's at least try and anticipate some of the problems that are going to occur, because it's got classically all of the problems of ad-hoc mobile networks and the security problems that are there in general are very interesting and in general have not been solved yet.

Matt Blaze: I think there is a legitimate architectural question of why consider Bluetooth's security separately from other kinds of fundamentally insecure links?

Tuomas Aura: I guess there is a practical reason. It's a link layer thing, and if you think of it as a replacement for the serial cable, then it creates new problems because it's not as secure as the physical serial cable, and you know the protocols that are going to be used on top of that are going to be exactly the same as you use in the average serial cable.

Markus Kuhn: All Bluetooth allows me to do is eavesdrop with cheaper equipment in a less radio quiet environment. In a sufficiently radio quiet environment, I can eavesdrop a serial cable using more expensive receiver equipment at

around ten metres distance. The underlying worst case of security is no different from a serial cable.

Tuomas Aura: But you can buy it in the nearest store and it will cost less than $5.

William Harbison: Denial of service is very easy too; you just buy a cheap microwave oven.

Pekka Nikander: I think there is also a more interesting denial of service problem and that's the denial of service where you just run out of batteries of the device.

Bruce Christianson: Also this is going to go broadband, so it's a replacement for a broadband serial cable not just for Ethernet/802.11. The argument is that that bandwidth is going to go up exponentially, but the sophistication of the protocols is not.

William Harbison: Of course it is aimed primarily at low power portable devices. If you think of that, there are some serious issues in terms of looking at the threat models. Portable/handheld devices have very little ability to compute and their batteries can be run down very, very quickly. So there are certain interesting security issues and threat models involved with Bluetooth which I don't think you see in certain other technologies.

John Ioannidis: I have a very good application for this, a little pocket sized transmitter which blocks everybody's Bluetooth communications in a concert hall or a theatre so that nobody's phone will ring, or we run their batteries down or something.

Markus Kuhn: It would probably be legal, because this is happening in the 2.4GHz range, which has been reserved for things like microwave ovens and other industrial jammers. Bluetooth is just a secondary user in these, and so has to tolerate the jammers.

John Ioannidis: I assume people know the origin of the term, Bluetooth. There was a Danish king in the 10th century named Blaatand, and he managed to unite parts of Denmark, Sweden and Norway which was considered impossible I guess[7]. The other thing of course is that lead smelters get blue teeth from lead metal poisoning so you know, go figure it.

Reply: Returning to the phone jamming concept, there is a group in Australia who are already looking into shutting off phones. Not by running down the battery or jamming them, but merely switching all phones to silent/vibrate mode upon entering a defined area.

John Ioannidis: There is at least one Israeli company that sells portable jammers, for use in theatres. I wish I knew who they were; I would buy lots of stock in them.

Matt Blaze: I just want to buy lots of their products.

John Ioannidis: I'll buy the stock. You buy the products.

Unfortunately CDMA spread-spectrum radios are particularly jam resistant.

Markus Kuhn: It depends on what sort of jammer you use. The usual trick is to use a pulse jammer, as this way the automatic gain control of the receiver

[7] It still is.

goes up and down very frequently and it doesn't resist. Jamming resistance is mostly against continuous wave transmissions such as microwave ovens.

Geraint Price: Do they have to allow emergency services calls through them?

Markus Kuhn: I've seen a mobile handheld jammer which was just wobbling around from 850 to 900MHz continuously; that's the frequency that the base stations transmit on, so you wouldn't get through with an emergency transmission.

Concluding Discussion When Does Confidentiality Harm Security?

Chair: Bruce Christianson

Bruce Christianson: We've just about got to the point now where we can see how real communities could actually achieve genuinely secure end-to-end confidentiality in a heterogeneous dynamic mobile environment. That's still a *slightly* contentious statement. There are some people who say that not all of the details of every protocol you'd need to do that are understood. But I think we're pretty much at the point where we can see what would have to be done.

This isn't as bad as it seems for our future as researchers, for two reasons. The first is that there's no sign that either government or industry has any willingness to do any of the things that it would take actually to achieve this objective. The second reason, which I'd like us to focus on in this discussion, is the fact that if we did have unbreakable end-to-end confidentiality it would probably make security worse rather than better in a number of respects.

Let's draw out some of these unexpected effects, and speculate about what one might do to solve the wave of new problems that secure end-to-end confidentiality would bring.

Tuomas Aura: You said "end-to-end" secrecy. Are you focusing on security of data in transit, or are you just using this word carelessly? I was thinking that it's in data storage and processing where you'd have the problems.

Bruce Christianson: I am using the word "end-to-end" carelessly. The end points may be co-located in space, the plaintext data may be in transit through a time tunnel, starting at the storage device and returning to it some years later.

The ground rule for the discussion is that end-to-end confidentiality is a solved problem, and that in particular the man in the middle has gone away: even when the man in the middle is actually a firewall who is trying to stop viruses from tunnelling into your system.

I stress that I am not claiming that every single one of the technical details in achieving this has been solved, merely that it's now sufficiently clear how we might go about doing it, for it to be interesting to ask what would happen next. If this really were a solved problem, where would be the next set of problems?

David Wheeler: Are the end points human?

Bruce Christianson: Not necessarily, but that's a ground rule not a dogmatic assertion.

William Harbison: Clearly privacy is a very bad thing as far as security is concerned, because it lets people get away with all sorts of things that you don't know about and you can't find out about.

When we look at how we design systems at the moment, a lot of them are designed on the basis that the system is really good and the users are really bad, and we protect the system from the user. I think actually this is the wrong model. The model should be that the users are really good and it's the system

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, pp. 229–238, 2002.

that's really bad, and we should protect the users from the system. Things like confidentiality and privacy mitigate against our ability to do this.

Virgil Gligor: I have come prepared with some concrete examples where confidentiality might in fact hurt security.

I. At the risk of beating a dead horse, I will illustrate three ways in which what we used to call "mandatory access controls" actually hurt security rather than helping.

Firstly, mandatory access controls were initially dreamed up by a number of people to imitate what they thought were military policies guarding national secrets. A number of companies built these multi-level secure operating systems, and one noticed when one had such a system that it invariably breaks applications: you have an application that runs on the top of such a system, and it takes an action which is viewed by the underlying security mechanism as violating the mandatory access controls, in particular the information flow.

So at that point you (as a designer and user) have a choice. Either you don't use the application, or you redesign the application to make it run in a multi-level security environment, or you simply exempt the application from the security checks which broke the application in the first place. So as a matter of course, people would take these applications which broke mandatory access controls, put them in the Trusted Base, bless them as being trusted, and disable most of the other security mechanisms.

As a consequence, the so-called Trusted Base ended up encompassing most of the useful applications. In effect, you ended up with a large piece of the system being almost automatically blessed to be trustworthy.

Clearly this caused a tremendous problem for people who were really interested in trustworthy systems. It was economically infeasible to develop the evidence of trustworthiness for such systems. I think this is one tangible example where confidentiality ends up working against the other security concerns.

The second technical aspect that came to mind, again in the multi-level area, was that a number of very useful security protocols — such as authentication protocols, client server encryption and authentication — would not work unless the end points were decreed *a priori* to operate at the same security level. Consequently things like Kerberos had to be radically redesigned in order to work in a multi-level security environment. This meant that for the class of systems which implement multi-level security, there is a tremendous delay in introducing security mechanisms which are useful in practice, because of the need for these retrofits.

Third, multi-level security gave people a false sense of security. Here is one example, there may be many others. Occasionally one wants to ask the question that Roger mentioned earlier[1]: what are all the accesses of a particular user, or of a particular set of users. I call this per-subject, or per-user, access review as opposed to per-object access review. When I tried to introduce this notion in the US federal criteria, in the early 90s, I got a lot of resistance from people who were in favour of mandatory access controls. Their argument was, you don't

[1] Roger Needham, these proceedings.

need per-subject review, mandatory access controls already tell you all you need to know. In particular, each user has a maximum and a minimum security level, and clearly all the objects that the user could access are between the maximum and minimum level and that's the end of the story. But what you really wanted to know is all objects and all privileges, not security levels.

So mandatory access controls give you a false sense of security. I think the conclusion is unmistakable in this area, these mechanisms and policies were grossly oversold.

II. The second, shorter, point that I want to make is to do with economics — not necessarily the economics of the more technical aspects of security. Basically I think we all realised at some point that security was, is, and will always be, a secondary concern in systems development. Most people build function, because function is what they can sell. Security — particularly security assurances or system structure to support security — does not sell. Security is not recognised very easily by the market as being useful. Otherwise systems like `*deleted on legal advice*` for example, would not be around.

Since security is a secondary concern, and an engineering concern at that, people are faced with development constraints: in particular, cost constraints. So in the past, particularly in the 80s, people such as designers and government put emphasis on these confidentiality-based access controls. There was a lot of development activity going into that area, using this fixed pool of resources. This meant that the security community did not pay any attention to certain other problems that have turned out to be much more important. In other words, the opportunity cost of confidentiality was very high. We ignored throughout the 80s a problem which was very dear to me, namely denial of service: in the literature between 1982 and 1990 no-one except myself published any papers in denial of service. Not that I made a big inroad in the problem, but at least I was able to define it and deny the proposition that denial of service is a subset of integrity concerns.

If you looked at the resource expenditures for building secure systems, they are all channelled towards confidentiality. More important problems in practice, such as denial of service, separation of duty, and integrity in general, were ignored. In that sense the damage was real and tangible.

III. Finally, one last point about privacy. Privacy conflicts with a number of very useful mechanisms, and one obvious one with which it clearly conflicts directly is intrusion detection, or let me call it auditing. Fundamentally, auditing tries to discover what authorised people did, not what outsiders or intruders did. We want this mechanism because we want to find out how Aldrich Ames, making $50,000 a year, ends up buying Jaguars and BMWs, and living in a high style. Once you notice that that has happened, you turn on audit and find out what Aldrich Ames does with CIA secrets. In the process of auditing what Aldrich Ames does you invariably, essentially, violate Aldrich Ames' privacy. There is a tremendous tension here. Auditing is needed to find out what insiders did, which is a major problem. Computer crime is really perpetrated (as we all know) by

insiders not by outsiders, so auditing is really very important. At the same time, auditing can grossly violate privacy concerns.

So if you choose to implement systems which enforce privacy to any very large extent, you have to be very careful how you design auditing, and you can end up having a very hard time auditing at all. I am sure we can find other examples where privacy conflicts with the security mechanisms in place but this is the obvious one.

So to recapitulate the three areas: mandatory access controls are bad for security, high opportunity cost, high cost of retrofit, false sense of security, and increase of trusted bases. Second, the economics of confidentiality is, or used to be, such that it denied development in other areas. And third, privacy conflicts.

Hiroshi Yoshiura: I have a question to Virgil about mandatory access control. A previous access control policy of application systems such as database can often be inconsistent with the access control policy of the platform of operating systems such as mandatory access control. I think the platform system should not be persistent to mandatory access control but to any access control policy of the application system. What do you think about this?

Virgil Gligor: I fully agree, in fact I go further. I say that the access mechanisms and policies of underlying operating systems should be somewhat neutral, in the sense that they should not attempt to solve any application security problem, because applications will invariably try at least to solve their own problems. Clearly database management systems have security policies and security mechanisms in them with good reason, so I no longer see the need to impose very peculiar access control policies, such as mandatory access controls, at the operating system level. In fact what I think you need is some set of neutral mechanisms, that could be used basically as you please. For example you may want to have a separation kernel, in fact I believe that your i-mark goes in that direction. You could have some very simple discretionary access controls as well, and that's about it. You don't want to place too many sophisticated access control mechanisms in the operating system, and certainly not in the kernel.

John Ioannidis: Let me heckle the third point (III) which Virgil raised, that privacy and auditing are mutually incompatible. In areas where you want auditing you are usually in a regulated environment. It's not that privacy is violated, it's that people who are going to be audited should know ahead of time that they have no privacy. In financial institutions, and I'm sure in government institutions, there is no expectation of privacy. For instance, e-mail in financial institutions is not considered private, the sender knows that they may be subject to audit if necessary.

Mark Lomas: I would make a slight distinction. The e-mail is not private, in that it's accessible, but on the other hand you don't sit down reading all the e-mails that somebody has.

Virgil Gligor: I agree completely with JI's point, that you should tell the people that their privacy may be breached. But I would also endorse Mark's distinction between archiving the e-mail and then searching for specific key words (that might end up with one of my e-mails appearing), and going through my e-

mail methodically. The distinction is between a machine going through it, where I've got definite rules as to what it's going to extract and if it doesn't meet any criteria it will not be shown to anybody, and some person ...

Bruce Christianson: You're designing something like Echelon [laughter].

Matt Blaze: You can paint this with a broader brush. E-mail in financial institutions *is* private but it isn't necessarily private from the financial institution itself. The bank's e-mail is private from me.

Virgil Gligor: You've given an example where policy regulates this conflict in favour of one party, but the more general point is that if you are auditing my actions, outside any such context, clearly that conflicts with my desire for privacy. I don't think that there is much dispute about that. Now whether this is legitimate, and whether the conflict can be resolved in favour of one or the other party, or whether there is a compromise, or whether the conflict can be defined away, is really a separate matter. But as a fundamental conflict I think it's still there.

Tuomas Aura: It's not only about regulation. A few days ago I received a telephone bill and it had a list of all the numbers I had called in the last two months. This didn't create a problem for me but, you can imagine that it might. A list of the numbers I called from work probably goes to someone every month. In Finland you cannot get a list of numbers you have called, it's simply impossible. But many people in Finland complain that they want to know which numbers were called because they want to know who created this phone bill. So this is clearly a case where security conflicts with privacy.

Matt Blaze: Imagine if *my* phone bill had a list of the numbers that *you* called.

Michael Roe: I want to redefine the argument here. This really should be "encryption considered harmful" and not "privacy considered harmful". These two things are very different. Despite the titles of protocols such as Pretty Good Privacy and Privacy Enhanced Mail, these were really about encrypting links. Often privacy is more a matter of what the thing on the end of the link does with the information you gave it after it got decrypted. This has nothing to do with the link.

Similarly, a lot of the problems we're seeing where encryption is harmful are when the thing being encrypted is not personal data, it's something that isn't even a secret at all, like protocol sequence numbers. We're just getting gratuitous encryption, because we're encrypting too far down in the stack. It benefits nobody keeping the sequence numbers 1, 2, 3, 4 and 5 secret, but it greatly detriments things which are trying to do link level encryption, that suddenly can no longer do header compression.

A similar example is SSL. Perhaps you'd want to use SSL to keep a credit card number secret from the eavesdropper, but using SSL to make sure that this web page that you got from the site really is an authentic web page is silly, when there's absolutely no need for confidentiality. Anybody in the world who goes to that web server and asks for it can get exactly the same page. All we're

doing is preventing caching from working, we're using "encryption" rather than "signature".

Virgil Gligor: The examples that Mike gave are good, and address the problem where confidentiality implemented by encryption conflicts with things like performance or desirable functionality. But what I'd like to see is some examples where encryption, as a means for confidentiality, conflicts with other security mechanisms and weakens them, as opposed to strengthening them.

Bruce Christianson: Let's have a go at that. There's an important distinction, emerging from what Mike said, between "local privacy", for example me sitting in my office typing a password into a computer, and what is often called "privacy" but is really confidentiality of communication, or the ability to carry out some kind of remote interaction in secret. It's easy to see that local privacy and confidentiality of communication can be in direct conflict in certain cases.

For example, suppose I have a system which can provide me with a really absolutely confidential link to Matt, and there is no possibility of a person in the middle, we've got complete privacy of our communication. But if we also have perfect local privacy for our applications, then it's actually the person (or application) looking over our shoulder at the end-point that is the threat.

Actually Matt is probably a bad choice, because I bet he uses a mailer that he wrote himself.

Matt Blaze: I do.

John Ioannidis: He couldn't figure out how to use Outlook [laughter].

Bruce Christianson: There's a very good reason for writing your own software. Suppose instead I have a connection to somebody less sensible, called Mutt, who uses something like `vi`, which keeps a record of every keystroke that he makes. Now where does that record go? What other software on his system has access to it, and who wrote that software, and what was their agenda? If any of that application software has become part of his trusted computing base, by the process which Virgil described earlier, then the fact that his system has a mechanism to provide perfectly secret communication from end to end will allow that software to tunnel his keystrokes to a third party in perfect secrecy, and there is no way for him, or me, to know.

So, if you *really* implement confidential communication, then you've just lost local privacy.

William Harbison: We all know the examples of firewalls being configured in a way which neatly aids the tunnelling of encrypted viruses through the system. If this virus can subsequently communicate in secret there's an even bigger problem.

Matt Blaze: Here's another example. Back during the cold war, the first cold war with nuclear weapons, the strategic air command messages that were sent out every 15 minutes, about whether or not everyone should start bombing Russia, were deliberately quite heavily authenticated but sent in the clear. Because we wanted the Russians to be reassured that we are not going to bomb them during the period. Encrypting those messages would in fact create uncertainty that might increase the level of threat.

Now you might be able to generalise this a bit further, to protocols in which you achieve security through openness rather than confidentiality, so that evil-doers are more likely to be caught. As an example, anybody with a key can get into this office building quietly, anyone with a sledgehammer can get in noisily and is likely to be seen doing it. The fact that anyone with a sledgehammer can get in is probably useful if, for example, there's a fire and the doors are locked, but it's not useful to an attacker.

There may be other examples where you deliberately want openness because that openness gives you an implicit audit capability.

Larry Paulson: There may be a situation where you have groups, we had a talk earlier on guarding a group working together with possibly a rogue member[2]. Clearly if all members of the group are required to do all their communications in the clear, or at least so other members of the group are able to listen in on these communications, then a rogue member will be detected much more quickly than if they're able to carry on in secret. Especially if you've got a pair of rogues conspiring, then they won't be able to carry out private conversations and so they'll be caught much more quickly.

Bruce Christianson: There's some recent work on protocol analysis, using what are called *honest-looking* participants[3]. The idea is that most people will be honest if being dishonest would mean that they got caught. So most people will comply outwardly with that part of a protocol where non-compliance could be detected. There are other things that protocols call for, like forgetting state, or coming up with a number that really is random where, where if you do it badly, no-one's going to find out. So there's some interest in trying to design protocols which have the property that they work provided the majority of the participants are honest-looking. This route leads you to impose various restrictions on what the boxes are allowed to do. You actually end up with something that looks rather like a gothic version of the White Box talk that Matt gave last year[4]. But it's clear that a focus on encryption for the purpose of confidentiality prevents any of these design techniques from being deployed. Full stop. Absolutely prevents it.

William Harbison: Yes, it's easier obviously to see malfeasance when the system is open. Many systems enable the systems operators to do all sorts of things which are completely hidden from everybody. They can even switch the security logs off, if you will! Whereas I have a model which says, every time a systems administrator changes anything in the system I'd like to see it broadcast to every user of the system. That was mandated so that everybody knew what was changed and when it was changed.

---

[2] Hassen Saïdi, Intrusion-Tolerant Group Management in Enclaves, these proceedings.

[3] See for example Canetti and Ostrovsky, *Secure Computation with Honest-Looking Parties: What if nobody is truly honest?* 31st ACM Symposium on Theory of Computing, Atlanta Georgia, USA, May 1999, pp 255–264.

[4] Matt Blaze, Looking on the Bright Side of Black-Box Cryptography, LNCS 2133, pp 54–61.

Bruce Christianson: JI made the point earlier that in a lot of relationships there is an implied term in the contract, that says certain aspects of your privacy are reduced. This is often the case in commercial contracts as well. The person who lends you the money to buy your house may very well acquire by doing so a right to monitor certain of your financial transactions. Your government also alleges that it has the right to observe certain properties of the social contract to make sure that they're occurring. Very often the software that you're required to run comes from these people.

Therefore there's an imperative, as Bill said earlier, to to treat the system, at least partly, as being the enemy. So the question is, even taking it for granted that local privacy is sometimes a good thing, is there ever a case where we can see non-local privacy — confidentiality of communication — as being a more important requirement than some form of integrity?

Virgil Gligor: Here is an example case about confidentiality and intellectual property protection, and how it actually causes security problems. Clearly if I write a piece of software, then I wish to protect it in such a way that you cannot find out, as a user, what the software actually does and how it's implemented. Assume that there is a spec and that the software does what it should do, but in addition I have put in all sorts of malicious code in it. The tension here is the intellectual property protection — and sometimes the confidentiality — of the software I produce, against your desire to protect yourself from the malicious software which I put in it.

Now this came to be a very important topic in the last two years particularly, because a number of government agencies in the US, among others, noticed that companies and individuals who fixed Year 2000 problems had a slight tendency to insert code which sometimes behaved rather maliciously. So consequently there was a drive to do reverse engineering to find out details of this software provided by these outside companies, to find out what it really does. There are reverse engineering conferences, there are tools being produced nowadays for reverse engineering, and that in itself conflicts with some of the laws that we have in the US, which basically state that you are not allowed to reverse engineer certain other products if you're competition.

I make my piece of software confidential because I have some value that I wish to protect, and of course I may have developed some mistakes too. The software is protected for confidentiality yet you, as a user, have a different interest here, with which my desire for confidentiality conflicts.

Roger Needham: This also comes up in connection with suspicion that software has got things put in it for the benefits of government not the customer. The Danes got very worked up about this because practically all the software they use is American. They wanted to know how they could satisfy themselves that American vendors did not put code in for the aid and comfort of precisely the agencies that the Danes were trying to defeat. The Danes took the view that every Danish individual or organisation should have access to sufficiently good encryption to keep the neighbours out of their communications, the neighbours usually meaning the British and the Americans in this context. They said, people

will always spy on their neighbours and you don't want to get hot under the collar about it. You should just let your folk defend themselves. So they then wanted to know how they could assure themselves that software from a variety of sources, including my employer, was not like that.

John Ioannidis: In a previous life, I spent some time in in Brussels consulting for the European Commission. Invariably I would get personally accused for the dominance of American and, particularly Microsoft, software, as if for some odd reason I was responsible for it. My reply, to which there was never any answer, was always, look Europe is a big economy, why don't you write your own if you don't like it? And they go, oh, oh, and you hear curses in various European languages, and then the issue just died until the next meeting.

Matt Blaze: Linux has some European origins. But at least I'm assured that with Linux there aren't any *secret* problems [laughter]. Most of these export restrictions have evaporated in the last couple of years.

Virgil Gligor: The US government has recently advertised the advanced encryption standard, which is essentially not an American invention, and this modes-of-operation pseudo-contest which may not be an American invention. Clearly anybody can do crypto nowadays to their heart's content.

John Ioannidis: It's interesting that the security subgroup of 3GPP has been extremely reluctant to do anything with AES, for no other reason than that it actually has had the blessing of the American government (even though it's a European algorithm). They certainly want to develop one of their own.

Tuomas Aura: The average politician, the average user, and the average business manager do not really know the difference between encryption and XOR. They just think, it is American, so why would it be trusted now more than a year ago.

Matt Blaze: But helping people who aren't able to behave rationally is a much bigger problem than trying to figure out how to behave rationally, let's solve the second one first before we try to help the irrationals.

Markus Kuhn: There is certainly a lot of personal suspicion in some communities, based on experience from the mid 80s to early 90s. For example, NSA has been extremely active with the ITU and ISO in order to prevent any sort of fax encryption becoming standard, in the mid 90s they then tried to push a rather broken one. Only last minute intervention, from a European player, made sure that the new T30 edition — the fax hand-shaking protocol — has what we would consider decent cryptography in it. Same with all these twists about how long it took to get DES sort of half-accepted by ISO whereas triple-DES still is not an ISO standard as far as I know. A lot of underground activity by the US intelligence community has gone on behind the scenes, and that has spread a lot of mistrust.

Matt Blaze: Most of that activity has focussed on influencing standards bodies, but you know it's a lot harder to poison the open research community.

Bruce Christianson: But, partly as a result, an unhealthy obsession has developed on confidentiality as being the battleground. I would argue that that is actually harming the security enterprise for ordinary people.

Matt Blaze: Right.

William Harbison: If you look at it at the system level, and you want to build a rational large scale distributed system, your first thought isn't going to be, I want to encrypt everything everywhere. That's going to affect my ability to configure it, my ability to keep it going, to do self repair, all of these things. I happen to agree with Bruce in this. We're seeing more and more fixation on encrypting everything that can be, whether it's necessary or not, and I think that's breaking our systems.

Matt Blaze: Yes, and with a false sense of security argument. We see that all the time, websites quite regularly pointing out that you can safely give us your credit card number because we use SSL. And you know, saying nothing about the back end.

William Harbison: I just walk in their front door and steal the disk, yes.

John Ioannidis: I guess this brings us back to the second point that Virgil raised. I would use the metaphor, or the simile, of steel doors and paper walls. Our communication can be iron-clad, but what's perfectly good 128-bit AES encryption going to do for me if somebody can just send me a virus through Linux (let's not always pick on Windows), which breaks in through Linux and grabs my data.

William Harbison: Or masquerades as you.

Bruce Christianson: Getting back to Bill's point about what gets encrypted, I think David's initial question was a very good one in this respect. There's a clear distinction between encrypting data that was provided by a human, and encrypting bit patterns that are generated as part of the system's processing of whatever it is doing.

David Wheeler: The term "security" of course covers two distinct things. One is physical security, is there a fire? Then there's confidentiality security and all that. But the first one is actually reasonably important, because if you have confidentiality and cryptology, recovering from a disaster is usually much harder, and if the system has got very good security, it's probably impossible.

Bruce Christianson: If your backups are really protected by encryption then it's very problematic to verify whether or not they are correct.

Tuomas Aura: Another thing which relates to this is, how long do you want to store data? If you have secret information the easiest way to protect the secrecy is to delete it. But in many cases it's not really that easy to delete things, because you often need a backup, and you don't want that to be deleted by accident.

Virgil Gligor: Once again, I think we must refocus on our subject matter, namely confidentiality affecting security, directly or indirectly. What I would like to do in the future is to distinguish between fundamental conflicts which cannot be resolved short of new policy, perhaps new philosophy, and technological conflicts that perhaps new technologies will resolve. I think it's important to ask people who are mostly interested in technology, to address that issue first: what can we solve? Which of these conflicts that are technological can be solved?

The Last Word

Thucydides

History of the Peloponnesian War**.

In this history I have set down a number of speeches which were delivered at various times just before and during the campaign. I have found it difficult to remember the precise words used in those speeches which I listened to myself, and my various informants have experienced the same difficulty.

The truth has not been easy to discover. Different eye-witnesses give different accounts of the same events, speaking from imperfect recollection, owing to the passage of time, or else out of partiality: to exaggerate the importance of their own theme, or from an interest primarily concerned with capturing the attention of the public.

Nor indeed can we rely upon every detail of the received tradition, if we seek to reach conclusions which are reasonably accurate. Even where authorities can be checked, people are inclined to accept stories of ancient times in an uncritical way, and so the origin of much of the subject matter is now lost in the unreliable stream of mythology.

So my method has been as follows: while keeping as closely as possible to the general sense of the words that were actually used, I have made the speakers say what, in my opinion, was called for by each situation.

The result has met with the general approval of those who were present at the events described, although some feel that my history is less easy to read because of the absence in it of a romantic element.

However, this work is not a piece of writing designed to meet the taste of an immediate public, but has been designed to last forever ...

** Freely translated and arranged by the Editors. As you can see, Thucydides has been plagiarizing our redaction technique for over two thousand years.

B. Christianson et al. (Eds.): Security Protocols, LNCS 2467, p. 239, 2002.

Author Index