The Correctness of Crypto Transaction Sets

Ross Anderson

Cambridge University

In this talk, given in April 2000, I introduced the topic of the correctness of the sets of transactions supported by cryptographic processors.

There has been much work on verifying crypto protocols, which are typically sets of 3-5 transactions exchanged by two principals. Yet when we look at actual implementations, there are typically 50 transactions supported by each principal's cryptographic facility (which might be a smartcard, cryptoprocessor, cryptographic service provider or security software library). Just as there are failures in which an opponent can achieve an undesired result by cutting and pasting valid transactions, so also there are failures in which an opponent uses cryptographic transactions in an order that was not anticipated by the facility's designer.

I can make this clear using an example I discovered shortly after this talk was given, and which is discussed in some detail in my book [1]. It works against many of the *security modules* – hardware cryptoprocessors – used by banks to manage ATM networks in the 1980s.

One of the security modules' principal protection goals was to ensure that no single employee of any bank in the network could learn the clear value of any customer's personal identification number (PIN). If PINs were simply managed in software, then any programmer with an appropriate level of access could learn any customer's PIN, and could therefore masquerade as them. As banks connected their ATMs together into networks, there was not just the risk that any bank's programmers might in theory masquerade as any other bank's customers, but also that a customer might falsely dispute a transaction he had actually made by claiming that some bank insider must be responsible.

So the cryptographic systems used to compute and verify PINs had to support a policy of dual control. No single member of any bank's staff should have access to the clear value of any bank customer's PIN. So key management involved hand carrying two or more *key components* to each ATM when it was initially brought online. These were combined together by exclusive-or to create a *terminal master key* (conventionally known as KMT), and further encryption keys would then be sent to the device encrypted under this master key. The higher-level management of PINs and keys relied on the tamper-resistance properties of the security modules to prevent programmers getting hold of PINs, or of data used to protect them such as the master keys in ATMs (and of course the PIN keys used to derive PINs from account numbers in the first place).

The upshot was that most bank security modules had a transaction to generate a key component and print out its clear value on an attached security printer.

They also returned this value to the calling program, encrypted under a master key KM which was kept in the tamper-resistant hardware:

SM ⟶ printer: KMT_i
SM ⟶ host: $\{KMT_i\}_{KM}$

and another which will combine two of the components to produce a terminal key:

Host ⟶ SM: $\{KMT_1\}_{KM}, \{KMT_2\}_{KM}$
SM ⟶ host: $\{KMT_1 \oplus KMT_2\}_{KM}$

To generate a terminal master key, you'd use the first of these transactions twice followed by the second, giving $KMT = KMT_1 \oplus KMT_2$.

The protocol failure is that the programmer can take any old encrypted key and supply it twice in the second transaction, resulting in a known terminal key (the key of all zeroes, as the key is exclusive-or'ed with itself):

Host ⟶ VSM: $\{KMT_1\}_{KM}, \{KMT_1\}_{KM}$
VSM ⟶ host: $\{KMT_1 \oplus KMT_1\}_{KM}$

The module also has a transaction to verify any PIN supplied encrypted under a key that itself is encrypted under KM:

Host ⟶ SM: *Account number*,$\{KMT_1\}_{KM}, \{PIN\}_{KMT}$
SM ⟶ host: Y / N

So the programmer now uses the zero key to encrypt every possible PIN (0000-9999) and supplies them to the security module along with the target account number and the zero key encrypted under KM, $\{0\}_{KM}$. After about 5000 transactions, on average, he finds the key.

Following this attack, Mike Bond discovered a couple of protocol failures with the IBM 4758, then the only cryptoprocessor evaluated to FIPS 140-1 level 4 – the highest level of tamper-resistance available for unclassified equipment. This startling result is described briefly in [1], and in more detail in a forthcoming paper.

Mike's attacks spurred me to look more closely at the old security module transaction set, and I realised that there is an ever better attack. This uses a transaction that will translate data from one key to another, and can be used to translate almost all the interesting crypto keys in the device to the zero key. The manuals for other security module products have yielded yet more sneaky attacks.

To sum up: the security of cryptographic transaction sets is a new and rapidly growing field of research. This is the talk that kicked it all off.

Much of the talk consisted of an explanation of how the IBM 3848, VISA Security Module and IBM 4758 work at the protocol level. This is rather hard to tidy up into a written article, as I used multicoloured slides to convey the concept of key typing by having 'red keys', 'blue keys', 'green keys' and so on, with each type of keys having a particular kind of function within the system. From section 2.2 onwards, for example, I describe the keys used to protect PINs as 'red keys': a red key should have the property that nothing encrypted under it may ever be output in the clear.

The key typing concepts are made more difficult for the beginner to understand because of the mechanism by which they are usually implemented. This works as follows. The security module has a number of internal master keys, and the working keys supplied in a transaction are encrypted under the master key appropriate to that type. (Thus in the VISA security module, for example, key encryption is done by two-key triple-DES, and what we call a 'red' key is actually a key encrypted in this way under master keys 14 and 15.) The various transactions allow a number of 'approved' operations to be conducted on data that is supplied either in the clear, or encrypted under keys of particular types. Thus, for example, to verify a customer PIN received from another bank's ATM through VISA's network, the programmer calls a security module transaction which accepts a PIN encrypted under a key of the type 'working key shared with VISA', and a PIN verification key; it decrypts the PIN from the first key, checks it with the second, and returns either 'Correct' or 'Incorrect'.

The reader wishing to dive into the brutal details of all this should read the CCA manual supplied with the 4758, as this (more or less) contains the two earlier devices via its backward compatibility modes [7]. The reader wishing for a more abstract view of the problem being addressed should read the following transcript in conjuntcion with chapter 14 of my book [1]. For this reason I've only edited the transcript lightly – adding a couple of illustrations, a couple of equations and a table from my slides.

Ross Anderson
Cambridge, 2nd January 2001

The Correctness of Crypto Transaction Sets

(talk transcript)

This talk follows on more from the talks by Larry Paulson and Giampaolo Bella that we had earlier. The problem I'm going to discuss is, what's the next problem to tackle once we've done crypto protocols? We keep on saying that crypto-protocols appear to be "done" and then some new application comes along to give us more targets to work on – multi-media, escrow, you name it. But sooner or later, it seems reasonable to assume, crypto will be done. What's the next thing to do?

The argument I'm going to make is that we now have to start looking at the interface between crypto and tamper-resistance.

Why do people use tamper resistance? I'm more or less (although not quite) excluding the implementation of tamper resistance that simply has a server sitting in a vault. Although that's functionally equivalent to many more portable kinds of tamper resistance, and although it's the traditional kind of tamper resistance in banking, it's got some extra syntax which becomes most clear when we consider the Regulation of Investigatory Powers (RIP) Bill. When people armed with decryption notices are going to be able to descend on your staff, grab keys, and forbid your staff from telling you, then having these staff working in a Tempest vault doesn't give the necessary protection.

1 Cryptography Involving Mutually Mistrustful Principals

In order to deploy "RIP-stop cryptography" (the phrase we've been using), you need to get guarantees which are mandatory – in the sense of mandatory access control. That is, you need guarantees, that are enforced independently of user action and are therefore subpoena proof, that a key was destroyed at date X, or that a key is not usable on packets over a week old, or that a key is only usable in packets over a week old if it's used with the thumb print of the corporate chief security manager, or whatever. You could do this with something like Kerberos, you just say that tickets can only be decrypted (as opposed to used) if the corporate security manager says so (this is maybe the cheap and cheerful way of getting RIP-stop into Windows). Alternatively, you could impose a requirement that a key should be unusable in the same key packet twice – possible if you've got a device like the 4758 with a reasonable amount of memory – or you could have a key which is unusable without a correct biometric. All of these give you ways of defeating RIP.

Now when we look at the larger canvas, at the applications that really use hardware tamper-resistance, we find that most of them are used where there is mutual mistrust. Often all the principals in a system mistrust each other.

The classic example, that Simmons discussed in the 80's, was the business of nuclear treaty verification, and indeed command and control generally [10]. The Americans don't trust the Russians, and the Pentagon doesn't trust the commanders in the field not to use the weapons, and so you end up with this enormous hierarchy of tamper-resistant boxes.

Pre-payment electricity meters provide another application that we have discussed at previous workshops [3]. There you have a central electricity authority, and you have hundreds of regional electricity vendors who sell tokens which operate meters. How do you balance power and cash, and stop the various vendors running off with the cash – or selling tokens that they then don't own up to? The only known solution is using some kind of hardware tamper-resistance, at least in the vending stations.

Then you've got bank security modules, which are basically used because banks don't want to trust each others' programmers not to get at customer PINs. Matt asked yesterday: "How could you possibly use tamper-resistance where you have got mutual mistrust?" In practice, tamper-resistance is used precisely where there is mutual mistrust, and this is what makes it difficult and interesting.

A brief history of tamper-resistance includes weighted code-books, sigaba cypher machines with thermite charges, and so on – all fairly familiar. GSM is fairly low-level stuff: your SIM contains a customer key K_c, and this key is *you*, for practical purposes, in the network; it is used to respond to random authentication challenges. You have no real motive to break into the SIM; it's just convenient to use a smartcard to hold K_c (although it does prevent some kinds of cloning such as cloning a phone in a rental car).

Pay-TV becomes more serious, and Markus Kuhn has talked about it at some length [4]. There the mutual mistrust is between the pay-TV operator and the customer. The customer has the master key on his premises, and the pay-TV operator doesn't trust the customer not to use the master key. Hence the need for tamper-resistance.

Banking is an issue that I've written about a lot [2], and the kind of security modules that I worked with (and indeed built some of) are basically PCs in steel cases with lid switches. Open the lid, and bang goes the memory. This is a very simple way of building stuff. You can make it more complex by putting in seismometers, and photo-diodes, and all sorts of other sensors, but the basic idea is simple. When you take it to its logical conclusion you get the IBM 4758, where you have got a DES chip, microprocessor, battery and so on, potted in a tamper-sensing core with alarm circuits around it.

What are the vulnerabilities of such products?

Many of the kinds of vulnerability we had in the 80's were to do with maintenance access. The maintenance engineer would go in to change the battery; he could then disable the tamper-sensing; and he could then go back on his next visit and take out the keys. That got fixed. The way to fix it was to take the batteries outside the device, as you can see in the photo of the 4758 in figure 1.

Then there is nothing user-serviceable inside the device, so as soon as you penetrate the tamper-sensing membrane the thing dies irrevocably – it becomes a door-stop.

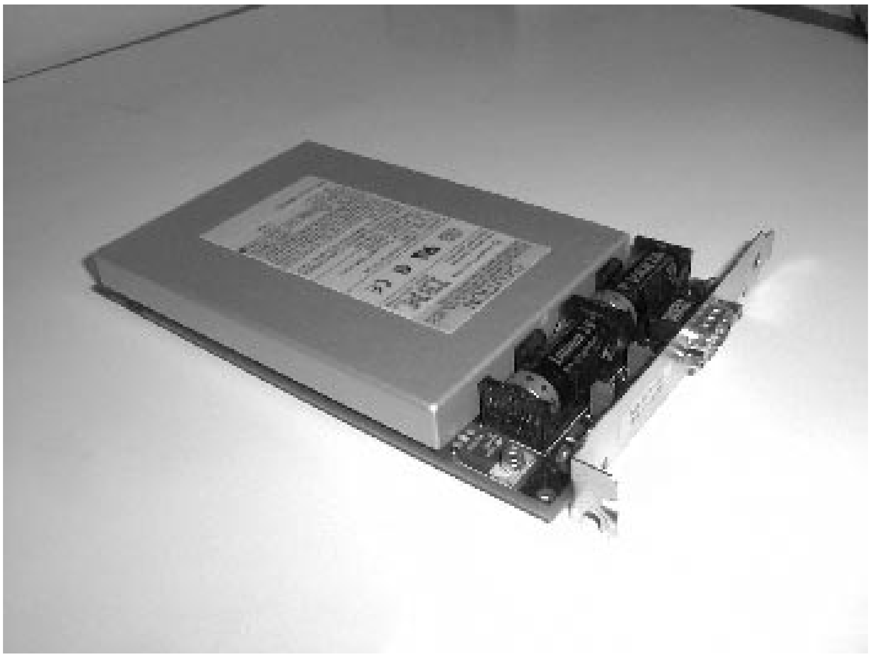

Fig. 1. – the IBM 4758 cryptoprocessor (courtesy of Steve Weingart)

We've more or less got to the point that the hardware can be trusted. We deal with the problem of whether the software can be trusted by getting FIPS 140-1 evaluation done by independent laboratories that sign a non-disclosure agreement and then go through the source code. That may be enough in some cases and not enough in others, but that isn't the subject of this talk.

2 Cryptoprocessor Transaction Sets

What I'm interested in is the command set that the cryptographic device itself uses. We have up till now, in this community, been looking at what happens in protocols where Alice and Bob exchange messages, and you need to contemplate only three or four possible types of message.

In the real world, things are much harder. If Alice and Bob are using tamper-resistant hardware, then they have got boxes that may support dozens or even hundreds of transactions. And we are assuming that Alice and Bob are untrustworthy – perhaps not all the time, perhaps you're just worried about a virus taking over Alice's PC – but perhaps Alice is simultaneously your customer and your enemy, in which case you can expect a more or less continuous threat.

2.1 Early transaction sets – CUSP

Let me review quickly how cryptographic processor instruction sets developed. The first one that's widely documented is IBM's CUSP, which came in in 1980 in the PCF product, and shortly after that in the 3848 [8]. This was a device the size of a water-softener that hung off your mainframe on a channel cable. It was designed to do things like bulk file encryption, using keys that were protected in tamper-sensing memory.

What IBM wanted to do was to provide some useful protection for these keys. If you merely have a device which will, on demand, encrypt or decrypt with a key that is held in its memory, then it doesn't seem to do you much good. Anybody who can access the device can do the encrypting and the decrypting.

So there began to be a realization that we want, in some way or another, to limit what various keys can be used for. Although the details are slightly more complex than the subset I'm giving here, the idea that IBM had was that you can divide keys into different types, and some keys can be local keys and other keys can be remote keys. How they did this was to have three master keys for the device – basically one master key and two offsets that would be XORed with it to create dependent master-keys. A key that was encrypted with the main master key, you can think of that as a blue key, and a key that was encrypted with variant KMH_0 you might think of as a green key, and a key that was encrypted with variant KMH_1 you might think of as a red key.

Now few people go through this documentation [7], because the IBM terminology and notation in it are absolutely horrible. But by starting to think in terms of key typing, or key colours, you can start to make much progress.

One of the goals that IBM tried to achieve with the 3848 was to protect for host-terminal communications using a session key which is never available in the clear, and which therefore has no street value. (This was an NSA concern at the time, because of their various staff members who had been selling key material to the Russians.) So how do you protect communication between a mainframe and its terminal?

The implementation was that you had a host transaction ECPH, which would take a green key – that is a session key encyphered under KMH0 – and a message m, and it would, in the trusted hardware, get its hands on this green key by decyphering it under KMH0, and then apply it to encypher the message m.

ECPH: $\{KS\}_{KMHO}, m \longrightarrow \{m\}_{KS}$

So you had a means of using the green key to encypher messages, and 'can encypher with green key' is a property of CUSP. Similarly, you can decypher with a green key, because can supply K_S encyphered under KMH_0 and a message encyphered under K_S and you will get back m.

How do you go about generating keys?

The technique is to supply a random number, or a serial number – or any value that you like, in fact – and use the device to decypher that under KMH0 to get a green key. In other words, the external encyphered keys that you hold on your database are just numbers that you thought up somehow. They are then 'decyphered' and used as keys.

Then you've got a more sensitive transaction, 'reformat to master key' (RFMK), which will take a master-key for a terminal – which is set up by an out-of-band technique – and K_S encrypted under KMH0, and it will encypher the session key under the master-key for the terminal.

RFMK: $\{KS\}_{KMHO}, KMT \longrightarrow \{KS\}_{KMT}$

So we've got another type of key, a red key I've called it, which is assumed to be held in the terminal. At the terminal, which is a different piece of hardware with different syntax, you've got a transaction DMK which will take a session key encyphered under KMT, and the clear value of KMT which is held in the hardware; it will give you a value of K_S which can then be used to do the decryption. (There isn't any hardware protection on the syntax of crypto in the terminal, because it is assumed that if you have physical access to the terminal then you can use the keys in it.)

Now this isn't all of the implementation, because there are other things that you need to do. You need to be able to manage keys, and (most confusingly of all) you need to be able to manage peer-to-peer communication between mainframes, which was being brought in at the time. So you end up having transactions to set up a key that is a local key at one mainframe and a remote key at the other mainframe. The IBM view at the time was that this made public-key cryptography unnecessary, because DES was so much faster and you needed tamper-resistant hardware anyway, so whay not just have local keys and remote keys? They have got the same functionality.

What went wrong with this? Well, it was very difficult to understand or explain, because the terminology they used is very complex. All sorts of implementation errors resulted. People found that it was simpler just to use blue keys which are all-powerful, rather than to mess around trying to separate keys into red and green and then finding that various things they wanted to do couldn't be done.

Nobody really figured out how to use this CUSP system to protect PINs properly as they were being sent from one bank to another. For example, a PIN from another bank's customer would come in from your cash machine, so you would decypher it and then re-encypher it with a key you shared with VISA for onward transmission. There were one or two hacks with which people tried to prevent PINs being available in clear in the host mainframe – such as treating PINs as keys – but for various reasons they didn't really work.

2.2 Banking security modules

So the next generation of security hardware to come along. The VISA security module, for example, takes the concept a bit further. They've actually got about five or six different colours of key.

The basic idea is that the security module is generating the PIN for a cash machine, by encrypting the account number using a key known as the PIN key. The result is then decimalised, and either given to the customer directly or added to an 'offset' on the customer's card to give the PIN that they must actually enter:

`Account number:`	`8807012345691715`
`PIN key:`	`FEFEFEFEFEFEFEFE`
`Result of DES:`	`A2CE126C69AEC82D`
`Result decimalised:`	`0224126269042823`
`Natural PIN:`	`0224`
`Offset:`	`6565`
`Customer PIN:`	`6789`

So you start off with keys which *never* decrypt, such as the PIN key KP, and you also have keys which *can* decrypt. Now the PIN key is a 'red key', a key which can never decrypt, and keys of this type are also used to protect PINs locally. (For example, when a PIN is encrypted at a cash machine for central verification, then the key used to perform this encryption is treated as a red key by the security module.) You can pass the security module a PIN which has been encrypted under a red key and it will say whether the PIN was right or not. The green key operates like a normal crypto key; you use it for computing MACs on messages and stuff like that.

So you use a red key to compute the PIN from the primary account number (PAN), and you use a red key to encipher the PIN from the ATM to the host security module. The whole thing works because nothing that's ever been encrypted under a red key is supposed to leak out – except the fraction of a bit per time that comes out from the PIN verification step. Then you add various support transactions, like 'generate terminal master key component', 'XOR two terminal master key components together to get a terminal master key' and so on and so forth.

What's the security policy enforced by the set of transactions that the security module provides? Well, it doesn't quite do Bell-LaPadula, because if you think of the red keys as being High, it allows an information flow from High to Low – about whether a PIN at High is right or not. So you need an external mechanism to track how many PIN retries have there been on an account, and if there's been more than three or 12 or whatever your limit is, then you freeze the account.

It doesn't quite do Clark-Wilson either, because you can generate master key components and XOR them together, and there's no system level protection for separation of duties on these two master key components; that's part of manual procedure. But the security module is moving somewhat in the direction of Bell-LaPadula and Clark-Wilson (although its evolution went down a different path).

Once you network many banks together, you've got further requirements. You've got one type of key used to encrypt a PIN on the link between the cash machine and the security module of the acquiring bank; you've got another type of key being used between the acquiring bank and VISA; another type of key being used between VISA and the issuing bank; then you've got the top level master keys which VISA shares with banks, and so on. You can think of these as being all of different colours.

So you've got all these various types of key, and you've got more and more complex transactions; the VISA security module now has about 60 different transactions in its transaction set. You've got support transactions, such as translating different PIN formats. You also have got potential hacks which I will come to later.

So the concern with something like the VISA box is: "Is there some combination of these 60-odd transactions, which if issued in the right order will actually spit out a clear-text PIN?" The designers' goal was that there were supposed to be one or two – in order to generate PINs for customers and keys for ATMs, for example – but they all involve trusted transactions that involve entering a supervisor password or turning a metal key in a lock. So the issue of trusted subjects is supposedly tied down.

2.3 The IBM 4758 security module product

Now we come to the more modern IBM product, the 4758. There's another picture of a 4758 in figure 2 with the shielding slightly removed. You can see a protective mesh here, a circuit board inside it behind a metal can, and the sensors here will alarm if you try to cut through (you can see the little sensor lines there). So what does this actually achieve? Well, assuming that the tamper-protection works (and it appears to), the software most people run on it – IBM's Common Cryptographic Architecture (CCA) – introduces a quite general concept of typing, namely control vectors.

A control vector can be thought of as a string bound to each key. The physical implementation of this is that you encipher the key under a different variant of a master key, but that's irrelevant at the level of abstraction that's useful to think about here. You can think of each key as going into the machine with a little tag attached to it saying, "I am a green key", "I am a red key", "I am a key of type 3126", or whatever. By default, CCA provides ten different types:

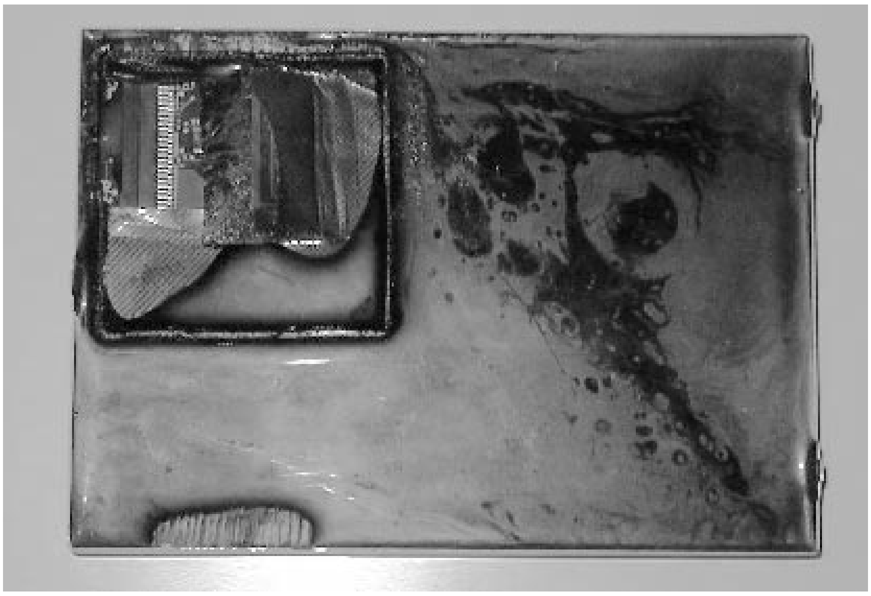

Fig. 2. – the 4758 partially opened – showing (from top left downwards) the circuitry, aluminium electromagnetic shielding, tamper sensing mesh and potting material (courtesy of Frank Stajano)

there is a data key; there's a data translation key; there's a MAC key; there's a PIN generation key; there's an incoming PIN encryption key; and so on. There are somewhat over 100 different transactions supplied with CCA that operate using these key types. In addition, you can define your own key types and roll your own transactions. You can download the details from the Web, all 400 and whatever pages of it [7], so the target is public domain.

Even if you use the standard default types, there are some very obscure issues that are left to the designer. There are some possible hacks, or at least unclear things, such as if people decrypt rubbish and use it, or if people use the wrong type of key, then can you gain some kind of advantage? Well keys supposedly have parity, so you have to decrypt on average about 32,000 'wrong' keys before you get one that works, but there have been attacks in the past which have used this [6]. Do any of these attacks work on the 4758?

Also, in the typing structure, we find that some of these types have got fairly explicit limitations (such as whether export is allowed or not), and you've got some types of types. But there's also a load of stuff that appears to be left more or less implicit, or at the very least has no supplied formal model.

You have got backward-compatibility modes. The 4758 will do everything that the old 3848 would do: it will do the ECPH, DCPH, re-format to master key, all that stuff, so you get free documentation of the obsolete equipment in the new equipment. You have things that approximate to the VISA transactions.

You also have a claim that the instruction set is comprehensive. Now this makes me feel rather uneasy, because the whole reason that I'm paying out all this money for a security processor is that its instruction set should *not* be comprehensive. There should be some things that it is simply not possible to do with it, and these things should be clearly understood.

So how did the current design come about? Well it should now be clear. We started off with the 3848, and that was not enough for banking so someone (actually Carl Campbell) invented the VISA security module. Then IBM saw itself losing market share to the VSM, so it does the embrace-and-expand play and incorporates the VSM functionality in the 4753. This evolves into the 4758, which was then used for things like prepayment electricity meters [3]. People then invented i-buttons and smartcards, yet more applications with which IBM's product line had to be compatible. Roger's phrase, "the inevitable evolution of the Swiss army knife", is very appropriate.

3 Verifying Crypto Transaction Sets

So how can you be sure that there isn't some chain of 17 transactions which will leak a clear key? I've actually got some experience of this, because with a security module that I designed there was one particular banking customer who said, we need a transaction to translate PINs. They were upgrading all of their customers to longer account numbers, and they didn't want to change the customers' PINs because they were afraid that this would decrease the number of people using their credit cards in cash machines. So they said, "We'd like to supply two account numbers, an old account number and a new account number, plus a PIN, and you return the difference between the old PIN and the new PIN so we can write that as an offset onto the card track."

So I said, "Fine – we'll code that up. But be warned: this transaction is dangerous. Remove this version of the software as soon as you've done that batch job." Of course they didn't, and about a year and a half later one of their programmers realised: "Hey – if I put in my account number and the MD's account number and my PIN, the machine then tells me what I've got to subtract from my PIN in order to get the MD's PIN!" Luckily for the bank, he went to the boss and told all; to which our response was, "We told you, why didn't you do as you were told?" (This episode is described in [5].)

So managing this kind of thing isn't as straightforward as you might think. As I mentioned, we have got no proper formal model of what goes on in a device like this. Although it has some flavours of Bell-LaPadula and some flavours of Clark-Wilson, it doesn't really behave according to either model; there are leaks at the edges. This brings us to all the research opportunities, which aren't necessarily 'insurmountable' but are, I suspect, a step change more difficult than the attacks on protocols that people have being doing up to now.

The first opportunity is to find an attack on the 4758. Find some property of the manual which even IBM will be forced to admit was not something they had in mind when they designed it. Alternatively prove that it's secure (though given the history of provable security, that's perhaps just as damning). Or, more practically, provide a tool that enables the designer to use the 4758 safely. Up until now, the IBM philosophy appears to have been: "We're selling you an F15 and you had jolly well better let us sell you a pilot and a maintenance crew and all the rest of it at the same time." They are not alone in this respect; BBN and all the other vendors appear to take the same view.

So you find that you can go and buy a set of tamper-resistant hardware boxes for maybe $2,000 each, but if you are a bank and you are building an actual installation using them, such as a certification authority, the outlay at the end of the day is more likely to be a quarter of a million dollars. By the time you've paid for the all the consultancy, and the machines they security modules will be attached to, and the firewalls to protect these machines from the network, and the procedures, and the training, and all the rest of it, the cost of the tamper-resistant box itself is essentially trivial.

So if anybody else is interested in getting into the systems integration business and getting that $240,000 instead of funneling it to IBM, then the competitive advantage might consist in having a suitable tool. Now it's not just the 4758! I use that as the target because we don't know how to break its physical tamper resistance, but there are many other architectures where you get a defined set of crypto transactions that may in fact be quite extensible. Those that come to mind are Microsoft CAPI, Intel CDSA – there's dozens out there.

This is definitely more difficult than doing verification of crypto protocols – and more interesting I think – because this is the real world problem. Somebody who has got a penetration of your corporate network is not trying to attack Kerberos, he's trying to attack the boxes that contain the Kerberos keys. Those boxes will do 50 transactions, of which you might normally use only three or four. But the others are there and available, unless you find some way of switching them off in a particular application, which again brings together a whole new set of design issues. So what's the nature of a crypto transaction set in general, and how do you go about verifying it?

Next question: how do you relate its properties to a higher level security policy model, such as Bell-LaPadula? And where this I suppose leads to, from the theory and semantics point of view, is whether you can get a general theory of restricted computation of computers that can do some things but cannot – absolutely, definitely, provably, cannot – do other things. Is there such a language that is complete in the sense that you can express any worthwhile crypto transaction set in it? What do you have to add to mathematics in order to get the expressiveness of metal? In the RIP-stop application it may very well be that provable destruction of a key is all you need, and anything else can be built on top of that given various external services such as perhaps secure time. But provable destruction of a key doesn't seem obviously related to what's going on in banking, because a typical bank is still running with the same ATM keys that they set up in 1985 when VISA set up the ACDS network. Key expiration just doesn't come into their universe.

And finally, if you're going to develop such a language, it would be useful if it could deal with public key stuff as well, such as homomorphic attacks, blinding, and so on. Because if you're going to build a machine which will, for example, not decrypt the same PGP message twice, then you have got somehow to make sure that the bad guys don't send you a message which is an old message which has been blinded so it looks like a new message.

Matt Blaze: I was surprised by your early comment – which I realise was a throwaway – that tamper-resistant hardware appears to be almost done, so let's move on to the next thing. It seems to me that tamper-resistant hardware if anything has become increasingly tenuous of late, particularly with respect to the use of out-of-band attacks such as power analysis and timing attacks. Things have been made even more difficult by putting the power supply outside of the box. Do you think once Paul Kocher gets his hands on one of these IBM boxes and hooks up his spectrum analyser to it that that's the end?

Reply: In the case of the IBM product I don't think so, because they've got a reasonably convincing proof (in the electrical engineering sense) that this can't happen. They filter the power lines into the device in such a way that all the interesting signals, given the frequencies that they're at inside the box, are attenuated more than 100 dBs. I agree that there is a serious problem with smartcards, but we've got a project going on that, we believe it to be fixable. We discuss this in a paper at a VLSI conference recently [9]. What we're proposing to do is to use self-timed dual-rail logic, so that the current that is consumed by doing a computation is independent of the data that is being operated on. You can do various other things as well; you can see to it that no single transistor failure will result in an output, and so on. In fact the NSA already uses dual-rail for resilience rather than power-analysis reasons. So there are technologies coming along that can fix the power analysis problem.

When I say that hardware tamper-resistance has been done, what I'm actually saying is that we can see our way to products on the shelf in five or seven years time that will fix the kind of things that Paul Kocher has been talking about.

John Ioannidis: The kind of tamper-resistance you seem to analyse here is not the sort of tamper resistance I'm likely to find in my wallet or my key-ring. It's more like the sort of tamper-resistance you find in a big box somewhere. Maybe that big box is sitting on my desk, or under my desk, but it's not going to be sitting on my person.

Reply: Wrong. The logical complexity that I've described here is independent of the physical size of the device. You consider the kind of SIM card that's likely to appear, with added-on transactions to support an electronic purse, and you're already looking at the same scale of complexity as a VISA security module. You're looking at much more difficult management problems, because you have got independent threads of trust: the phone company trusts its K_c and the bank trusts its EMV keys, and how do these devices interact within the smartcard itself? It suddenly becomes very hairy and we don't at present have the tools to deal with it.

Virgil Gligor: How many 4758s did IBM sell, do you know what proportion were penetrated relative to NT or Windows 98, or any of the things that you have on your desk?

Reply: They're selling of the order of a couple of thousand a year, I'm aware of only one penetration which occurred – during testing as a result of a manufacturing defect.

I assume that an NT box, or for that matter a Linux box, can be penetrated by somebody who knows what he's doing, and so you must assume that the host to which your security module is attached belongs to the enemy, even if only from time to time.

(Indistinct question)

Reply: It's useful in security to separate design assurance from implementation assurance and from strength of mechanism. Now the strength of mechanism of the tamper-resistance mechanisms in 4758s are very much higher than a smartcard, and a smartcard is very much higher than an NT server, but that's a separate lecture course.

What I'm concerned about here is the logical complexity of the set of commands which you issue to your tamper resistant device, be it a smartcard, or an NT server, or a 4758, because if you screwed up somehow in the design of those, then it doesn't matter how much steel and how many capacitors and how many seismometers and how many men with guns you've got. You're dead.

(Indistinct question)

Reply: We brought the designer of the 4758 over and we grilled him for a day, and we've looked at the things partially dismantled, and we've got one or two off-the-wall ideas about how you might conceivably, if you spent a million bucks building equipment, have a go at it. But we don't have any practical ideas that we could offer to a corporation and say, "Give us a million bucks and with a better than 50% chance we'll have the keys out in a year." The 4758 is hard!

John Ioannidis: What kind of primitives am I going to be offloading on to tamper-proof hardware so that no matter what kind of penetration my NT box has, there won't be an order for a billion dollars to be transferred from somewhere else to my account without my consent, so I can't get framed for fraud.

Reply: The sort of thing that hardware boxes are traditionally used for is to protect the PIN key, the key that generates all the PINs of the eleven million customers in your bank, because if a bad guy gets hold of that you've had it, you have to re-issue your cards and PINs. The sort of thing that you can use a 4758 for in a high-value corporate environment is to see to it that a transaction will only go out signed with the great seal of Barclays Bank (or whatever) provided – let's say – in the case of a $100m transaction, at least two managers of grade

4 or above have signed it using the signing keys and their smartcards. You can set up a system involving the 4758 and the smartcards which will distribute the keys, manage the certificates, and then will basically enforce your Clark-Wilson policy.

John Ioannidis: I'm not a grade 4 manager at Goldman Sachs, or whatever they're called, Suppose I'm a poor AT&T employee, and I have a box on my desk and I do my banking transactions over it, and I have somewhere a key given me by the bank, or I created my own, or both, and I have a tamper-resistant device which I want to use to secure my transaction. Now the question is, what kind of operations should that thing support, and what kind of interface to me, that does not go through NT or Linux, so that a transaction could not be made without my personal, physical approval – not my electronic approval. This is also a hard problem.

Reply: Agreed. But it's not the problem I'm addressing. Many people have talked about signature tablets that will display text; you put your thumb print, for example, to OK this text and this is supposed to give you a high degree of non-repudiation. But that's a different problem; it's not the problem that interests me in the context of this talk.

Michael Roe: What Ross is deliberately ignoring is the effect of successful tamper-resistance. If you have a very secure transaction in a very physically secure processor then the attacker is going to attack the software that goes in between the user interface and the secure box.

Reply: In this talk what I'm interested in is the command set that you use to drive a tamper resistant device and how you go about defining that in a way that's sensible – and assuring yourself that the design holds up. (I'm talking about design assurance not about the human computer interface aspects.)

(Indistinct question)

Reply: There have been attacks on early pay TV systems where there were protocol failures in the smartcard. Now if you believe Gemplus figures that by 2003 there will be 1.4 billion microprocessor cards sold a year (and as the biggest OEM they should know about that) then clearly it is important in engineering and industrial and economic terms, as well as being a problem that's directly related to protocol verification. That's the reason why I bring it to this audience. It is a protocol problem; but it's the next step up in difficulty. And it's sufficiently different that there's an awful lot of new research for people to do, and, hopefully, new theories to discover.

(Indistinct question)

Reply: I'm interested in Clark-Wilson-type objects and Bell-LaPadula-type objects, because that's where the real world is. The usual objection to Clark-Wilson is, "Where's the TCB?" Now as an example application, banking security modules are 90% of the way towards a Clark-Wilson TCB, so this should also be of scientific interest.

(Indistinct question)

Reply: An awful lot of design effort is believed to be about to go into multifunction smartcards, where you've got a SIM card sitting in your WAP device, and into which signed applets can be downloaded by your bank, by your insurance company, by your health care provider, etc.

(Indistinct question)

Reply: Now separability is not the only issue here, there's the correctness of the transactions that people can issue, and whether the applications themselves can be sabotaged by somebody just issuing the right commands in order.

(Indistinct question)

Reply: Even if you can download these applets safely, the applets will have to talk to each other in many cases for it to be useful. For example, if you're going to use your WAP device to pay for hospital treatment, by transferring money from your electronic purse to your health card, then that involves having interfaces that allow transactions between the two, and that's hard to do.

References

1. RJ Anderson, *'Security Engineering – a Guide to Building Dependable Distributed Systems'*, Wiley (2001) ISBN 0-471-38922-6
2. RJ Anderson, "Why Cryptosystems Fail" in *Communications of the ACM* vol 37 no 11 (November 1994) pp 32–40; earlier version at `http://www.cl.cam.ac.uk/users/rja14/wcf.html`
3. RJ Anderson, SJ Bezuidenhoudt, "On the Reliability of Electronic Payment Systems", in *IEEE Transactions on Software Engineering* vol 22 no 5 (May 1996) pp 294–301; `http://www.cl.cam.ac.uk/ftp/users/rja14/meters.ps.gz`
4. RJ Anderson, MG Kuhn, "Tamper Resistance – a Cautionary Note", in *Proceedings of the Second Usenix Workshop on Electronic Commerce* (Nov 96) pp 1–11; `http://www.cl.cam.ac.uk/users/rja14/tamper.html`
5. RJ Anderson, MG Kuhn, "Low Cost Attacks on Tamper Resistant Devices", in *Security Protocols – Proceedings of the 5th International Workshop* (1997) Springer LNCS vol 1361 pp 125–136
6. M Blaze, "Protocol Failure in the Escrowed Encryption Standard", in *Second ACM Conference on Computer and Communications Security*, 2–4 November 1994, Fairfax, Va; proceedings published by the ACM ISBN 0-89791-732-4, pp 59–67; at `http://www.crypto.com/papers/`
7. IBM, *'IBM 4758 PCI Cryptographic Coprocessor – CCA Basic Services Reference and Guide*, Release 1.31 for the IBM 4758-001, available through `http://www.ibm.com/security/cryptocards/`
8. CH Meyer, SM Matyas, *'Cryptography: A New Dimension in Computer Data Security'*, Wiley, 1982
9. SW Moore, RJ Anderson, MG Kuhn, "Improving Smartcard Security using Self-timed Circuit Technology", Fourth ACiD-WG Workshop, Grenoble, ISBN 2-913329-44-6, 2000; at `http://www.g3card.com/`
10. GJ Simmons, "How to Insure that Data Acquired to Verify Treaty Compliance are Trustworthy", GJ Simmons, *Proceedings of the IEEE* v 76 no 5 (1988)