Computer Science Tripos part 2 – Security

Lecture 8 – Key Management and Cryptographic Protocols

Ross J. Anderson

1 Introduction

It is usually not the encryption algorithm itself that breaks in the real world, but the way in which it is used. In this lecture, we shall start to look at how algorithms can be incorporated into real systems, and some of the pitfalls for the unwary.

We will usually simplify the discussion by ignoring the algorithm details and using the following notation: if P is the plaintext and K the key, then the ciphertext is given by:

$$C = \{P\}_K$$

We will assume that this gives integrity as well as secrecy, so cut-and-paste attacks on encrypted blocks do not work. You can imagine the operation as consisting of MAC followed by encryption using separate keys, which are concatenated to give K.

One of the hardest problems is key management, and many real-world attacks have exploited key management failures (we mentioned satellite TV; there are many other examples). Key management covers not just how Alice and Bob set up a key when they need one – key distribution – but also how they certify the random number generators which manufacture keys, how they log and audit key use, how they protect keys against physical or software based attacks, how they replace them regularly to make the enemy analysts' work harder, and how they destroy them.

We will start off by looking at the key distribution side of things. How can we go about supplying two principals – conventionally called Alice and Bob – with a good key whenever they want to speak to each other?

2 End-to-end: SWIFT 1

The simplest way to manage keys is to leave the whole problem to the users. This was the approach taken for many years by SWIFT, the network used by banks to send payment instructions to each other internationally. This can be thought of as a secure email system for messages of value. If you get money sent to you by a relative abroad, the chances are that this transfer was made by a SWIFT message saying something like:

> *'To NatWest Bank, London. Please pay from our account with you no. 1234567890 the sum of £1000 to John Smith of 456 Chesterton Road, who has an account with Midland Bank Cambridge no. 301234 4567890123, and notify him that this was for "wedding present from Doreen Smith". From First Fireman's Bank of Santa Barbara, CA, USA. Charges to be paid by us. Authenticator =* 470145D3*'.*

The major concern here is to prevent message manipulation – neither bank wants a hacker to tweak the value from *£1,000* to *£1,000,000* – and the secondary concern is to protect client confidentiality.

Confidentiality depends on line encryption devices between the banks and the local SWIFT node, and between these nodes and the main SWIFT processing sites. Their key management is straightforward: keys are hand carried in EEPROM cartridges between the devices at either end of a leased line.

Integrity depends on the MAC, in this case the eight digits 470145D3. The keys for this MAC used to be managed end-to-end: whenever a bank set up a relationship overseas, the senior manager who negotiated it would exchange keys with his opposite number, whether in a face-to-face meeting or afterwards by post to each others' private addresses. There would typically be two key components to minimise the risk of compromise, with one sent in each direction (on the grounds that even if a bank manager's mail is opened by a criminal, at least posting letters is fairly secure). The key would not be enabled until both banks confirmed that it had been safely received and installed.

SWIFT itself was outside the trust perimeter for the MAC keys; it did however provide a message logging service to assist in dispute resolution. In fact, each local node may be run on contract by a different facilities firm, and logs are kept at each node; so in our example above, an attacker would have to subvert not just SWIFT itself, but also its local contractors in Los Angeles and London in order to delete a log entry. Logs are the primary evidential resource in the event of a dispute between two banks.

SWIFT I ran for twenty years without a single report of external fraud. There have been many internal frauds – such as programmers inserting bogus messages into the SWIFT processing queue – but such attacks are the concern of other computer security mechanisms.

Returning to the cryptology, the problem with end-to-end key management solutions is that they do not scale well. With a thousand correspondents, a typical bank needed two people – a full time administrator and a part time manager – to control all the keys, and with a hundred thousand, the problem would have become intractable. In general, if we have n subscribers to a network, then for each pair of them to share a key we would need $n(n-1)/2$ keys, and there are few applications in which it is economic to manually preposition so much key material.

3 Data Translation — ATM Networks

The second key management architecture which we shall consider is the one used by ATM network operators such as VISA, Mastercard and Cirrus.

The paper enclosed describes how most ATMs operate using some variant of a system developed by IBM. This uses a secret key, called the 'PIN key', to derive the PIN from the account number. The result of this operation is called the 'natural PIN'; an offset can be added to it in order to give the PIN which the customer must enter. The offset has no real cryptographic function; it just enables customers to choose their own PIN. Here is an example of the process:

Account number N:	`8807012345691715`
PIN key KP:	`FEFEFEFEFEFEFEFE`
Result of DES $\{N\}_{KP}$:	`A2CE126C69AEC82D`
$\{N\}_{KP}$ decimalised:	`0224126269042823`
Natural PIN:	`0224`
Offset:	`6565`
Customer PIN:	`6789`

Security depends on keeping the PIN key secret, and the usual strategy is to supply a 'terminal master key' to each ATM in the form of two printed components, which are carried to the branch by two separate officials, input at the ATM keyboard, and combined to form the key. The PIN key, encrypted under this terminal key, is then sent to the ATM by the bank's central facility, as are any keys required for encrypting communications. The underlying security policy is that of dual control: the cards and PINs are sent to the customer via separate channels; all keys are handled in two components by two separate people; and so on.

The problem faced by VISA and Mastercard was to extend this security policy to tens of thousands of banks worldwide. The solution was to insist on standardised encryption units, called 'security modules', which provide a trusted computing base, or TCB, with modules at each member bank and in each switch on the network. The policy they enforce is dual control: it should always take the collusion of two or more staff at any member bank in order to get access to a customer PIN.

When a bank joins a network, this TCB provides a PIN translation facility. It is not feasible for 10,000 banks to share keys in pairs, so each bank connects to a switch provided by VISA or Mastercard, and the security modules in these switches translate the traffic – they decrypt it from the key they share with the originating member and then re-encrypt it under the key shared with the destination. Master keys are set up between banks and switches under dual (or even triple) control; then, each morning, the switch exchanges fresh working keys with each member – encrypted under that member's master key.

As discussed in the paper on ATM fraud, there have been a number of security problems. For example, although authorisation requests from an ATM are encrypted, the authorisation response is usually in the clear, in order to save the cost of the extra security modules, ports, and cpu capacity to support the extra transaction throughput.

However, given that ATM transactions typically have to be completed in 30 seconds, the basic architecture is understandable; during this time, the authorisation request will flow from the ATM through the bank which owns it to its country switch, and to the card issuing bank through its country switch. At each intermediate hop, the local processor invokes a security module transaction to decrypt the PIN under one key and re-encrypt it under another key. The card issuing bank then has to check the available balance, and finally the response has to flow all the way back. Doing all this reliably within 30 seconds, with logs being written at each step, is not trivial.

Message translation services can also be needed by the military. Their problem is that countries cooperating in a joint venture – such as operation Desert Storm – may simply not trust each other enough to share their cryptographic algorithms, key management schemes or even equipment. In order to coordinate their campaign, they may need vehicles containing crypto kit and technicians from both sides, which pass traffic to each other in clear. Such facilities can be a serious bottleneck as well as a security hazard.

4 A More General Solution – Authentication Protocols

In the general case, message translation services are not an appropriate way to manage secure association. Where Alice and Bob want to communicate, they may still use a trusted third party, who is conventionally called Sam, but as an introduction service rather than a translation service; his rôle will be to set up a crypto key on request from any two subscribers. They can then (usually) communicate with each other independently of him. So they do not require him to be online whenever they communicate, and message volumes are much more manageable.

We saw simple authentication protocols in the 1b lectures. However, designing such protocols is not straightforward; there are many examples, both in the literature and among fielded systems, of protocols that turned out to be vulnerable. The opponent can try all sorts of tricks including replaying messages, cutting and splicing, and even reflecting messages back at other players to use them as oracles. For example, during the Angolan war in the late 1980's, Cuban pilots developed an ingenious way to defeat the South African IFF system. Cuban bombers would loiter near the Namibian border until South African bombers raided Luanda; they would then attack Namibia and when they received IFF challenges from the South African air defences, would forward these to the Luanda air defence radars which would fire them at the South African aircraft and relay the responses.

So we will assume that Charlie can manipulate any and all messages between Alice, Bob and Sam; conceptually, we may think of Charlie as being the phone company. This may seem a bit extreme, but in modern distributed systems it is often straightforward to attack routers, so it is prudent to assume that all messages might be diverted through a hostile processor.

We will write A for Alice, B for Bob, S for Sam, C for Charlie, and $\rightarrow$ for the transmission of a message. We will also denote concatenation of message elements by a comma, and a key shared by X and Y as K_{XY}. We will also write N for a number used once (or 'nonce') which we use to guarantee that messages are fresh. For example, in a challenge-response logon or IFF protocol, the challenge can be viewed as a nonce; in many payment protocols, the transaction sequence number acts as a nonce (though extra care is needed if nonces are predictable by the opponent).

4.1 The Needham-Schroder Protocol

Many existing authentication protocols (e.g., Kerberos) are derived from the Needham-Schroder protocol, which appeared in 1978.

Message 1	$A \rightarrow S:$	A, B, N_A
Message 2	$S \rightarrow A:$	$\{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
Message 3	$A \rightarrow B:$	$\{K_{AB}, A\}_{K_{BS}}$
Message 4	$B \rightarrow A:$	$\{N_B\}_{K_{AB}}$
Message 5	$A \rightarrow B:$	$\{N_B - 1\}_{K_{AB}}$

Here Alice takes the initiative, and tells Sam that she wants to talk to Bob. Sam provides her with a session key, and with a certificate to convey this key to Bob. She passes it to him, and he then does a challenge-response to check that she is present and active.

There is one flaw in this protocol – Bob just assumes that the key K_{AB} from Sam is fresh. This need not be so: Alice could have waited a year between steps 2 and 3. In many applications this may not be important; but if Charlie ever got hold of Alice's key, he could use this to set up session keys with many other principals, and could continue to use these even after Alice had set up a new K_{AS} with Sam. In other words, revocation is problematic.

There are a number of ways to fix this: the Otway-Rees protocol involves Bob in the challenge-response, while Kerberos uses timestamps instead of challenges.

4.2 Otway-Rees

The first of these protocols was proposed simultaneously by Otway and Rees and by Needham and Schroder as a fix for the problem above. It has the additional feature of being capable of implementation as two nested remote procedure calls. In an email environment, it may be convenient for Alice to take the initiative, but in more tightly coupled systems, the RPC route may be more suitable.

Message 1	$A \rightarrow B:$	$M, A, B, \{N_A, M, A, B\}_{K_{AS}}$
Message 2	$B \rightarrow S:$	$M, A, B, \{N_A, M, A, B\}_{K_{AS}}, \{N_B, M, A, B\}_{K_{BS}}$
Message 3	$S \rightarrow B:$	$M, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$
Message 4	$B \rightarrow A:$	$M, \{N_A, K_{AB}\}_{K_{AS}}$

Here, M is also a nonce, which we can think of as the message number. It binds together the two parts of the message from B to S.

4.3 Kerberos

The other Needham-Schroder derivative is Kerberos, an authentication protocol developed for MIT's Project Athena and which now forms the basis of authentication in NT. By using timestamps rather than nonces, the old key problem is avoided. The protocol itself is on p25 of the attached article.

Kerberos is much more than just a protocol: it provides a whole mechanism for distributed access control. Users log on to an authentication server; this gives them a key to talk to a ticket granting server which in turn gives them tickets which allow access to various resources such as files. We may talk about some of its other features if there is time.

4.4 Clipper

The Clipper chip, more formally known as the Escrowed Encryption Standard, was introduced by the US government in 1992 as a proposed replacement for DES. It incorporated Skipjack, which was at the time classified secret and only available in tamper-resistant hardware devices, with a key management protocol containing a 'back door' to let the US government decrypt any traffic encrypted by an EES device.

In the words of the government press release:

> A "key-escrow" system will be established to ensure that the "Clipper Chip" is used to protect the privacy of law-abiding Americans. Each device containing the chip will have two unique "keys", numbers that will be needed by authorized government agencies to decode messages encoded by the device. When the device is manufactured, the two keys will be deposited separately in two "key-escrow" data bases that will be established by the Attorney General.

> Access to these keys will be limited to government officials with legal authorization to conduct a wiretap.

In the first version of their mechanism, the sending device constructs a law enforcement access field (LEAF) which contains the session key in use, encrypted with the chip's unique key, as well as the chip's unique ID encrypted under a universal secret 'family key'. The receiving device would decrypt this field and check that it was well formed before decrypting the data.

However, the escrowed encryption standard has run into problems. Firstly, there was a storm of protest from civil libertarians and from users who have invested in other encryption technologies; and secondly, technical faults were found with the scheme. Matt Blaze at Bell Labs showed how a LEAF could be forged – the authentication on the LEAF was only 16 bits, and so bogus but acceptable LEAFs could be found by trial and error. The Clipper chip was a failure in the market, and the NSA subsequently declassified Skipjack in the hope of getting cheaper implementations. Attempts to impose key escrow have continued but have tended to use public key mechanisms, which we will discuss later.

5 Formal Methods

One way to avoid the mistakes that are common in crypto protocol design is to use formal techniques to verify that our protocol does what we think. There are a number of these techniques, and perhaps the best known is the Burrows-Abadi-Needham logic, or BAN logic as it is usually referred to. This is described in the appendix.

Finally, a caveat: even some provably correct protocols can fail as a result of using an inappropriate mode of operation. See example 1.

6 Further Reading

An alternative approach to protocols design is presented in *'Prudent Engineering Practice for Cryptographic Protocols'*, by Abadi and Needham (DEC SRC Research Report 125 — 1/6/94). Kerberos is described in a number of places, such as "Secure Distributed Computing" by Jeff Schiller in the November 1994 *Scientific American*. There is some material on SWIFT in Davies and Price's *'Security for Computer Networks'*, while a comprehensive source on ATMs is Meyer and Matyas' *'Cryptography — A New Dimension in Data Security'*.

7 Problems

1. What happens to the Otway-Rees protocol if the encryption used is the CBC mode of a block cipher, and the protocol element boundaries coincide with the cipher's block boundaries?

2. The Wide Mouthed Frog protocol runs as follows (T_x are timestamps). Does it work?

Message 1	$A \rightarrow S:$	$A, \{T_A, B, K_{AB}\}_{K_{AS}}$
Message 2	$S \rightarrow B:$	$\{T_S, A, K_{AB}\}_{K_{BS}}$

3. The following protocols were put forward in December 1994 as a draft international standard, and are up for voting in the near future. How would you advise the British Standards Institution to cast the UK's vote?

   - ISO DIS 11770 mechanism 7:

Message 1	$A \rightarrow S:$	A, B
Message 2	$S \rightarrow A:$	$\{K_{AB}, B\}_{K_{AS}}, \{K_{AB}, A\}_{K_{BS}}$
Message 3	$A \rightarrow B:$	$\{K_{AB}, A\}_{K_{BS}}$

   - ISO DIS 11770 mechanism 13:

Message 1	$A \rightarrow B:$	$N_A, \{B, K_{AB}\}_{K_{AS}}$
Message 2	$B \rightarrow S:$	$N_B, A, \{B, K_{AB}\}_{K_{AS}}$
Message 3	$S \rightarrow B:$	$\{N_B, K_{AB}, A\}_{K_{BS}}$
Message 4	$B \rightarrow A:$	$\{N_A, N_B\}_{K_{AB}}$
Message 5	$A \rightarrow B:$	$\{N_B, N_A\}_{K_{AB}}$

4. What would be the effect if the clock in the Kerberos authentication server runs fast?

5. Not all authentication protocols are used to set up keys. In the following protocol by Woo and Lam, Bob simply wants to check that Alice is present. Does it work?

Message 1	$A \rightarrow B:$	A
Message 2	$B \rightarrow A:$	N_B
Message 3	$A \rightarrow B:$	$\{N_B\}_{K_{AS}}$
Message 4	$B \rightarrow S:$	$\{A, \{N_B\}_{K_{AS}}\}_{K_{BS}}$
Message 5	$S \rightarrow B:$	$\{N_B\}_{K_{BS}}$

   Is there a general principle of protocol design which you can draw from this example?

6. Design a cryptographic protocol to protect a burglar alarm system (see lecture 5). Assume that the client (the alarm controller) is an eight-bit processor, while the server (in the security company's control room) has a powerful processor.