CHAPTER

3

Protocols

It is impossible to foresee the consequences of being clever.

— Christopher Strachey

Every thing secret degenerates, even the administration of justice; nothing is safe that does not show how it can bear discussion and publicity.

— Lord Acton

3.1 Introduction

If security engineering has a deep unifying theme, it is the study of security protocols. We've come across a few protocols informally already — I've mentioned challenge-response authentication and Kerberos. In this chapter, I'll dig down into the details. Rather than starting off with a formal definition of a security protocol, I will give a rough indication and then refine it using a number of examples. As this is an engineering book, I will also give many examples of how protocols fail.

A typical security system consists of a number of principals such as people, companies, computers and magnetic card readers, which communicate using a variety of channels including phones, email, radio, infrared, and by carrying data on physical devices such as bank cards and transport tickets. The security protocols are the rules that govern these communications. They are typically designed so that the system will survive malicious acts such as people telling lies on the phone, hostile governments jamming radio, or forgers altering the data on train tickets. Protection against all possible attacks is often too expensive, so protocols are typically designed under certain assumptions about the threats. For example, the logon protocol that consists of a user

entering a password into a machine assumes that she can enter it into the right machine. In the old days of hard-wired terminals in the workplace, this was reasonable; now that people log on to websites over the Internet, it is much less so. Evaluating a protocol thus involves answering two questions: first, is the threat model realistic? Second, does the protocol deal with it?

Protocols may be extremely simple, such as swiping a badge through a reader in order to enter a building. They often involve interaction, and do not necessarily involve technical measures like cryptography. For example, when we order a bottle of fine wine in a restaurant, the standard wine-waiter protocol provides some privacy (the other diners at our table don't learn the price), some integrity (we can be sure we got the right bottle and that it wasn't switched for, or refilled with, cheap plonk) and non-repudiation (it's hard for the diner to complain afterwards that the wine was off). Blaze gives other examples from applications as diverse as ticket inspection, aviation security and voting in [185].

At the technical end of things, protocols can be much more complex. The world's bank card payment system has dozens of protocols specifying how customers interact with cash machines and retail terminals, how a cash machine or terminal talks to the bank that operates it, how the bank communicates with the network operator, how money gets settled between banks, how encryption keys are set up between the various cards and machines, and what sort of alarm messages may be transmitted (such as instructions to capture a card). All these protocols have to work together in a large and complex system.

Often a seemingly innocuous design feature opens up a serious flaw. For example, a number of banks encrypted the customer's PIN using a key known only to their central computers and cash machines, and wrote it to the card magnetic strip. The idea was to let the cash machine verify PINs locally, which saved on communications and even allowed a limited service to be provided when the cash machine was offline. After this system had been used for many years without incident, a programmer (who was playing around with a card reader used in a building access control system) discovered that he could alter the magnetic strip of his own bank card by substituting his wife's bank account number for his own. He could then take money out of her account using the modified card and his own PIN. He realised that this enabled him to loot any other customer's account too, and went on to steal hundreds of thousands over a period of years. The affected banks had to spend millions on changing their systems. And some security upgrades can take years; at the time of writing, much of Europe has moved from magnetic-strip cards to smartcards, while America has not. Old and new systems have to work side by side so that European cardholders can buy from American stores and vice versa. This also opens up opportunities for the crooks; clones of European cards are often used in magnetic-strip cash machines in other countries, as the two systems' protection mechanisms don't quite mesh.

So we need to look systematically at security protocols and how they fail. As they are widely deployed and often very badly designed, I will give a number of examples from different applications.

3.2 Password Eavesdropping Risks

Passwords and PINs are still the foundation on which much of computer security rests, as they are the main mechanism used to authenticate humans to machines. I discussed their usability and 'human interface' problems of passwords in the last chapter. Now let us consider some more technical attacks, of the kind that we have to consider when designing more general protocols that operate between one machine and another. A good case study comes from simple embedded systems, such as the remote control used to open your garage or to unlock the doors of cars manufactured up to the mid-1990's. These primitive remote controls just broadcast their serial number, which also acts as the password.

An attack that became common was to use a 'grabber', a device that would record a code broadcast locally and replay it later. These devices, seemingly from Taiwan, arrived on the market in about 1995; they enabled thieves lurking in parking lots to record the signal used to lock a car door and then replay it to unlock the car once the owner had left[1].

One countermeasure was to use separate codes for lock and unlock. But this is still not ideal. First, the thief can lurk outside your house and record the unlock code before you drive away in the morning; he can then come back at night and help himself. Second, sixteen-bit passwords are too short. It occasionally happened that people found they could unlock the wrong car by mistake (or even set the alarm on a car whose owner didn't know he had one [217]). And by the mid-1990's, devices appeared which could try all possible codes one after the other. A code will be found on average after about 2^{15} tries, which at ten per second takes under an hour. A thief operating in a parking lot with a hundred vehicles within range would be rewarded in less than a minute with a car helpfully flashing its lights.

So another countermeasure was to double the length of the password from 16 to 32 bits. The manufacturers proudly advertised 'over 4 billion codes'. But this only showed they hadn't really understood the problem. There was still

[1]With garage doors it's even worse. A common chip is the Princeton PT2262, which uses 12 tri-state pins to encode 3^{12} or 531,441 address codes. However implementers often don't read the data sheet carefully enough to understand tri-state inputs and treat them as binary instead, getting 2^{12}. Many of them only use eight inputs, as the other four are on the other side of the chip. And as the chip has no retry-lockout logic, an attacker can cycle through the combinations quickly and open your garage door after 2^7 attempts on average.

only one code (or two codes) for each car, and although guessing was now impractical, grabbers still worked fine.

Using a serial number as a password has a further vulnerability: there may be many people with access to it. In the case of a car, this might mean all the dealer staff, and perhaps the state motor vehicle registration agency. Some burglar alarms have also used serial numbers as master passwords, and here it's even worse: the serial number may appear on the order, the delivery note, the invoice and all the other standard commercial paperwork.

Simple passwords are sometimes the appropriate technology, even when they double as serial numbers. For example, my monthly season ticket for the swimming pool simply has a barcode. I'm sure I could make a passable forgery with our photocopier and laminating machine, but as the turnstile is attended and the attendants get to know the 'regulars', there is no need for anything more expensive. My card keys for getting into the laboratory where I work are slightly harder to forge: the one for student areas uses an infrared barcode, while the card for staff areas has an RFID chip that states its serial number when interrogated over short-range radio. Again, these are probably quite adequate — our more expensive equipment is in rooms with fairly good mechanical door locks. But for things that lots of people want to steal, like cars, a better technology is needed. This brings us to cryptographic authentication protocols.

3.3 Who Goes There? — Simple Authentication

A simple example of an authentication device is an infrared token used in some multistorey parking garages to enable subscribers to raise the barrier. This first transmits its serial number and then sends an authentication block consisting of the same serial number, followed by a random number, all encrypted using a key which is unique to the device. We will postpone discussion of how to encrypt data and what properties the cipher should have; we will simply use the notation $\{X\}_K$ for the message X encrypted under the key K.

Then the protocol between the access token in the car and the parking garage can be written as:

$$T \rightarrow G : T, \{T, N\}_{KT}$$

This is the standard protocol engineering notation, and can be a bit confusing at first, so we'll take it slowly.

The in-car token sends its name T followed by the encrypted value of T concatenated with N, where N stands for 'number used once', or *nonce*. Everything within the braces is encrypted, and the encryption binds T and N together as well as obscuring their values. The purpose of the nonce is to assure the recipient that the message is *fresh*, that is, it is not a replay of

an old message that an attacker observed. Verification is simple: the parking garage server reads T, gets the corresponding key KT, deciphers the rest of the message, checks that the nonce N has not been seen before, and finally that the plaintext contains T (which stops a thief in a car park from attacking all the cars in parallel with successive guessed ciphertexts).

One reason many people get confused is that to the left of the colon, T identifies one of the principals (the token which represents the subscriber) whereas to the right it means the name (that is, the serial number) of the token. Another is that once we start discussing attacks on protocols, we can suddenly start finding that the token T's message intended for the parking garage G was actually intercepted by the freeloader F and played back at some later time. So the notation is unfortunate, but it's too well entrenched now to change easily. Professionals often think of the $T \rightarrow G$ to the left of the colon is simply a hint as to what the protocol designer had in mind.

The term *nonce* can mean anything that guarantees the freshness of a message. A nonce can, according to the context, be a random number, a serial number, a random challenge received from a third party, or even a timestamp. There are subtle differences between these approaches, such as in the level of resistance they offer to various kinds of replay attack, and they increase system complexity in different ways. But in very low-cost systems, the first two predominate as it tends to be cheaper to have a communication channel in one direction only, and cheap devices usually don't have clocks.

Key management in such devices can be very simple. In a typical garage token product, each token's key is simply its serial number encrypted under a global master key KM known to the central server:

$$KT = \{T\}_{KM}$$

This is known as *key diversification*. It's a common way of implementing access tokens, and is very widely used in smartcard-based systems as well. But there is still plenty of room for error. One old failure mode that seems to have returned is for the serial numbers not to be long enough, so that someone occasionally finds that their remote control works for another car in the car park as well. Having 128-bit keys doesn't help if the key is derived by encrypting a 16-bit serial number.

Weak ciphers also turn up. One token technology used by a number of car makers in their door locks and immobilisers employs a block cipher known as Keeloq, which was designed in the late 1980s to use the minimum number of gates; it consists of a large number of iterations of a simple round function. However in recent years an attack has been found on ciphers of this type, and it works against Keeloq; it takes about an hour's access to your key to collect enough data for the attack, and then about a day on a PC to process it and recover the embedded cryptographic key [172]. You might not think this a practical attack, as someone who gets access to your key can just drive off with

your car. However, in some implementations, there is also a terrible protocol vulnerability, in that the key diversification is not done using the block cipher itself, but using exclusive-or: $KT = T \oplus KM$. So once you have broken a single vehicle key for that type of car, you can immediately work out the key for any other car of that type. The researchers who found this attack suggested 'Soon, cryptographers will drive expensive cars.'

Indeed protocol vulnerabilities usually give rise to more, and simpler, attacks than cryptographic weaknesses do. At least two manufacturers have made the mistake of only checking that the nonce is different from last time, so that given two valid codes A and B, the series $ABABAB...$ was interpreted as a series of independently valid codes. A thief could open a car by replaying the last-but-one code. A further example comes from the world of prepayment utility meters. Over a million households in the UK, plus many millions in developing countries, have an electricity or gas meter that accepts encrypted tokens; the householder buys a token, takes it home and inserts it into the meter, which then dispenses the purchased quantity of energy. One electricity meter widely used in South Africa checked only that the nonce in the decrypted command was different from last time. So the customer could charge the meter up to the limit by buying two low-value power tickets and then repeatedly feeding them in one after the other [59].

So the question of whether to use a random number or a counter is not as easy as it might seem [316]. If you use random numbers, the lock has to remember a reasonable number of past codes. You might want to remember enough of them to defeat the *valet attack*. Here, someone who has temporary access to the token — such as a valet parking attendant — can record a number of access codes and replay them later to steal your car. Providing enough nonvolatile memory to remember hundreds or even thousands of old codes might push you to a more expensive microcontroller, and add a few cents to the cost of your lock.

If you opt for counters, the problem is synchronization. The key may be used for more than one lock; it may also be activated repeatedly by jostling against something in your pocket (I once took an experimental token home where it was gnawed by my dogs). So there has to be a way to recover after the counter has been incremented hundreds or possibly even thousands of times. This can be turned to advantage by allowing the lock to 'learn', or synchronise on, a key under certain conditions; but the details are not always designed thoughtfully. One common product uses a sixteen bit counter, and allows access when the deciphered counter value is the last valid code incremented by no more than sixteen. To cope with cases where the token has been used more than sixteen times elsewhere (or gnawed by a family pet), the lock will open on a second press provided that the counter value has been incremented

between 17 and 32,767 times since a valid code was entered (the counter rolls over so that 0 is the successor of 65,535). This is fine in many applications, but a thief who can get six well-chosen access codes — say for values 0, 1, 20,000, 20,001, 40,000 and 40,001 — can break the system completely. So you would have to think hard about whether your threat model includes a valet able to get access codes corresponding to chosen counter values, either by patience or by hardware hacking.

A recent example of design failure comes from TinyOS, an operating system used in sensor networks based on the IEEE 802.15.4 ad-hoc networking standard. The TinySec library commonly used for security protocols contains not one, but three counters. The first is lost as the radio chip driver overwrites it, the second isn't remembered by the receiver, and although the third is functional, it's used for reliability rather than security. So if someone monkeys with the traffic, the outcome is 'error' rather than 'alarm', and the network will resynchronise itself on a bad counter [340].

So designing even a simple token authentication mechanism is not at all straightforward. There are many attacks that do not involve 'breaking' the encryption. Such attacks are likely to become more common as cryptographic authentication mechanisms proliferate, many of them designed by programmers who thought the problem was easy and never bothered to read a book like this one. And there are capable agencies trying to find ways to defeat these remote key entry systems; in Thailand, for example, Muslim insurgents use them to detonate bombs, and the army has responded by deploying jammers [1000].

Another important example of authentication, and one that's politically contentious for different reasons, is 'accessory control'. Many printer companies embed authentication mechanisms in printers to ensure that genuine toner cartridges are used. If a competitor's product is loaded instead, the printer may quietly downgrade from 1200 dpi to 300 dpi, or simply refuse to work at all. Mobile phone vendors make a lot of money from replacement batteries, and now use authentication protocols to spot competitors' products so they can be blocked or even drained more quickly. All sorts of other industries are getting in on the act; there's talk in the motor trade of cars that authenticate their major spare parts. I'll discuss this in more detail in Chapter 22 along with copyright and rights management generally. Suffice it to say here that security mechanisms are used more and more to support business models, by accessory control, rights management, product tying and bundling. It is wrong to assume blindly that security protocols exist to keep 'bad' guys 'out'. They are increasingly used to constrain the lawful owner of the equipment in which they are built; their purpose may be of questionable legality or contrary to public policy.

3.3.1 Challenge and Response

Most cars nowadays have remote-controlled door unlocking, though most also have a fallback metal key to ensure that you can still get into your car even if the RF environment is noisy. Many also use a more sophisticated two-pass protocol, called challenge-response, to actually authorise engine start. As the car key is inserted into the steering lock, the engine controller sends a challenge consisting of a random n-bit number to the key using short-range radio. The car key computes a response by encrypting the challenge. So, writing E for the engine controller, T for the transponder in the car key, K for the cryptographic key shared between the transponder and the engine controller, and N for the random challenge, the protocol may look something like:

$$\begin{aligned} E &\rightarrow T: \quad N \\ T &\rightarrow E: \quad \{T, N\}_K \end{aligned}$$

This is still not bulletproof.

In one system, the random numbers generated by the engine management unit turned out to be predictable, so it was possible for a thief to interrogate the key in the car owner's pocket, as he passed, with the anticipated next challenge. In fact, many products that incorporate encryption have been broken at some time or another because their random number generators weren't random enough [533, 395]. The fix varies from one application to another. It's possible to build hardware random number generators using radioactive decay, but this isn't common because of health and safety concerns. There are various sources of usable randomness in large systems such as PCs, such as the small variations in the rotation speed of the hard disk caused by air turbulence [358]. PC software products often mix together the randomness from a number of environmental sources such as network traffic and keystroke timing and from internal system sources [567]; and the way these sources are combined is often critical [703]. But in a typical embedded system such as a car lock, the random challenge is generated by encrypting a counter using a special key which is kept inside the device and not used for any other purpose.

Locks are not the only application of challenge-response protocols. In HTTP Digest Authentication, a web server challenges a client or proxy, with whom it shares a password, by sending it a nonce. The response consists of the hash of the nonce, the password, and the requested URI [493]. This provides a mechanism that's not vulnerable to password snooping. It's used, for example, to authenticate clients and servers in SIP, the protocol for Voice-Over-IP (VOIP) telephony. It is much better than sending a password in the clear, but suffers from various weaknesses — the most serious being middleperson attacks, which I'll discuss shortly.

A much more visible use of challenge-response is in *two-factor authentication*. Many organizations issue their staff with password generators to let them log on to corporate computer systems [1354]. These may look like calculators (and some even function as calculators) but their main function is as follows. When you want to log in to a machine on the network, you call up a logon screen and are presented with a random challenge of maybe seven digits. You key this into your password generator, together with a PIN of maybe four digits. The device encrypts these eleven digits using a secret key shared with the corporate security server, and displays the first seven digits of the result. You enter these seven digits as your password. This protocol is illustrated in Figure 3.1. If you had a password generator with the right secret key, and you entered the PIN right, and you typed in the result correctly, then the corporate computer system lets you in. But if you do not have a genuine password generator for which you know the PIN, your chance of logging on is small.

Formally, with S for the server, P for the password generator, PIN for the user's Personal Identification Number that bootstraps the password generator, U for the user and N for the random nonce:

$$\begin{aligned} S &\rightarrow U: \quad N \\ U &\rightarrow P: \quad N, PIN \\ P &\rightarrow U: \quad \{N, PIN\}_K \\ U &\rightarrow S: \quad \{N, PIN\}_K \end{aligned}$$

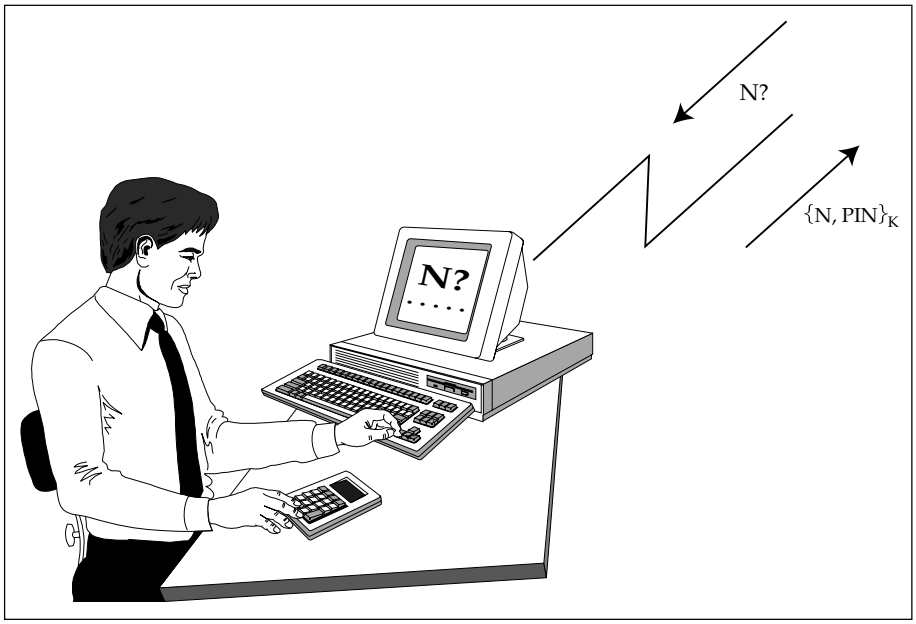

Figure 3.1: Password generator use

These devices appeared from the early 1980s and caught on first with phone companies, then in the 1990s with banks for use by staff. There are simplified versions that don't have a keyboard, but just generate a new access code every minute or so by encrypting a counter: the RSA SecurID is the best known. One sector after another has been adopting authentication tokens of one kind or another to replace or supplement passwords; the US Defense Department announced in 2007 that the introduction of an authentication system based on the DoD Common Access Card had cut network intrusions by 46% in the previous year [225].

The technology is now starting to spread to the customer side of things. By 2001, password generators were used by some exclusive private banks, such as Coutts, to authenticate their online customers. These banks never suffered any phishing fraud. By 2006, some banks in the Netherlands and Scandinavia had rolled out the technology to all their millions of customers; then the frauds started. The phishermen typically use real-time man-in-the-middle attacks (which I'll describe in the next section) to take over a session once the user has authenticated herself to the bank. As of late 2007, some banks in the UK and elsewhere in Europe have been introducing the Chip Authentication Program (CAP), which is implemented by giving bank customers a calculator that uses their bank card to do crypto[2]. This calculator, when loaded with a bank card, will ask for the customer's PIN and, if it's entered correctly, will compute a response code based on either a counter (as a one-off authentication code for a card transaction, or a one-step logon to a banking website) or a challenge (for a two-step logon). There is also a third mode of operation: if session takeover becomes a problem, the CAP calculator can also be used to authenticate transaction data. In this case, it's planned to have the customer enter the amount and the last eight digits of the payee account number into her CAP calculator.

But the result might not be as good in banking as it has been in the armed forces. First, when your wallet is stolen the thief might be able to read your PIN digits from the calculator — they will be the dirty and worn keys. If you just use one bank card, then the thief's chance of guessing your PIN in 3 tries has just come down from about 1 in 3000 to about 1 in 10. Second, when you use your card in a Mafia-owned shop (or in a shop whose terminals have been quietly reprogrammed without the owner's knowledge), the bad guys have everything they need to loot your account. Not only that — they can compute a series of CAP codes to give them access in the future, and use your account for wicked purposes such as money laundering. Third, someone who takes your bank card from you at knifepoint can now verify that you've told them

[2]Bank cards in many European countries have an EMV smartcard chip on them, and new UK bank cards have software to compute authentication codes as well as to operate ATMs and shop terminals.

the right PIN. A further problem is that the mechanisms can be used in a range of protocols; if you have to give a one-off authentication code over the phone to buy a book with your bank card, and the bookseller can then use that code to log on to your bank, it's clearly a bad thing. A deeper problem is that once lots of banks use one-time passwords, the phishermen will just rewrite their scripts to do real-time man-in-the-middle attacks. These have already been used against the early adopter banks in the Netherlands and Scandinavia. To see how they work, we will now look at a military example.

3.3.2 The MIG-in-the-Middle Attack

The ever-increasing speeds of warplanes in the 1930s and 1940s, together with the invention of the jet engine, radar and rocketry, made it ever more difficult for air defence forces to tell their own craft apart from the enemy's. This led to a serious risk of 'fratricide' — people shooting down their colleagues by mistake — and drove the development of systems to 'identify-friend-or-foe' (IFF). These were first fielded in World War II, and in their early form enabled an airplane illuminated by radar to broadcast an identifying number to signal friendly intent. In 1952, this system was adopted to identify civil aircraft to air traffic controllers and, worried about the loss of security once it became widely used, the U.S. Air Force started a research programme to incorporate cryptographic protection in the system. Nowadays, the typical air defense system sends random challenges with its radar signals, and friendly aircraft have equipment and keys that enable them to identify themselves with correct responses. The chapter on electronic warfare has more details on modern systems.

It's tricky to design a good IFF system. One of the problems is illustrated by the following story, which I heard from an officer in the South African Air Force (SAAF). After it was published in the first edition of this book, the story was disputed — as I'll discuss below. Be that as it may, similar games have been played with other electronic warfare systems since World War 2. The 'Mig-in-the-middle' story has in any event become part of the folklore, and it nicely illustrates how attacks can be carried out in real time on challenge-response authentication protocols.

In the late 1980's, South African troops were fighting a war in northern Namibia and southern Angola. The goals were to keep Namibia under white rule, and impose a client government (UNITA) on Angola. Because the South African Defence Force consisted largely of conscripts from a small white population, it was important to limit casualties, so most South African soldiers remained in Namibia on policing duties while the fighting to the north was done by UNITA troops. The role of the SAAF was twofold: to provide tactical support to UNITA by bombing targets in Angola, and to ensure that the Angolans and their Cuban allies did not return the compliment in Namibia.

Suddenly, the Cubans broke through the South African air defenses and carried out a bombing raid on a South African camp in northern Namibia, killing a number of white conscripts. This proof that their air supremacy had been lost helped the Pretoria government decide to hand over Namibia to the insurgents — itself a huge step on the road to majority rule in South Africa several years later. The raid may also have been the last successful military operation ever carried out by Soviet bloc forces.

Some years afterwards, a SAAF officer told me how the Cubans had pulled it off. Several MIGs had loitered in southern Angola, just north of the South African air defense belt, until a flight of SAAF Impala bombers raided a target in Angola. Then the MIGs turned sharply and flew openly through the SAAF's air defenses, which sent IFF challenges. The MIGs relayed them to the Angolan air defense batteries, which transmitted them at a SAAF bomber; the responses were relayed back in real time to the MIGs, who retransmitted them and were allowed through — as in Figure 3.2. According to my informant, this had a significant effect on the general staff in Pretoria. Being not only outfought by black opponents, but actually outsmarted, was not consistent with the world view they had held up till then.

After this tale was published in the first edition of my book, I was contacted by a former officer in SA Communications Security Agency who disputed the story's details. He said that their IFF equipment did not use cryptography yet at the time of the Angolan war, and was always switched off over enemy territory. Thus, he said, any electronic trickery must have been of a more primitive kind. However, others tell me that 'Mig-in-the-middle' tricks were significant in Korea, Vietnam and various Middle Eastern conflicts.

In any case, the tale illustrates the basic idea behind an attack known to the cryptographic community as the *man-in-the-middle* or (more recently) the *middleperson* attack. It applies in a straightforward way to the challenge-response authentication performed by password calculators: the phishing site invites the mark to log on and simultaneously opens a logon session with his bank. The bank sends a challenge; the phisherman relays this to the mark, who uses his device to respond to it; the phisherman relays it to the bank, and is now authenticated to the bank as the mark. This is why, as I discussed above, European banks are introducing not just a simple response to a single challenge, but an authentication code based on input fields such as the amount, the payee account number and a transaction sequence number.

However, once the protocol-level vulnerabilities are fixed by including all the transaction data, the big problem will be usability. If it takes two minutes and the entry of dozens of digits to make a payment, then a lot of customers will get digits wrong, give up, and then either call the call center or send paper checks — undermining the cost savings of online banking. Also, the bad guys will be able to exploit the fallback mechanisms, perhaps by spoofing customers

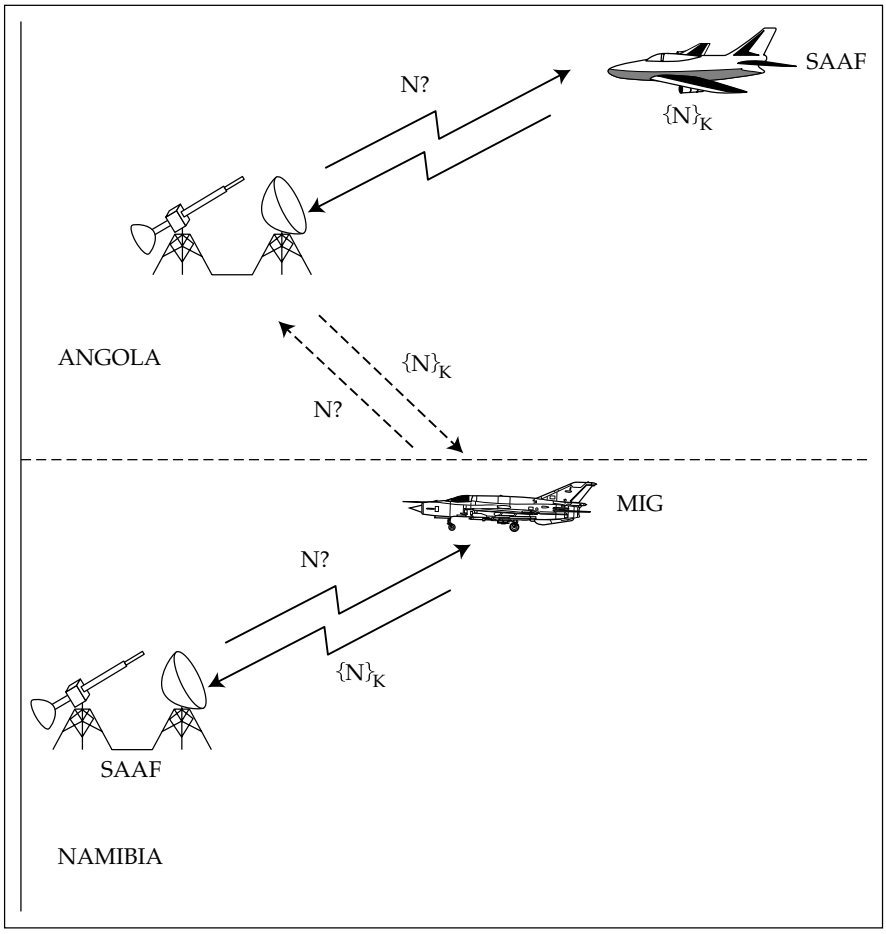

Figure 3.2: The MIG-in-the middle attack

into calling voice phishing phone numbers that run a middleperson attack between the customer and the call center.

We will come across the man-in-the-middle attack again and again in applications ranging from pay-TV to Internet security protocols. It even applies in online gaming. As the mathematician John Conway once remarked, it's easy to get at least a draw against a grandmaster at postal chess: just play two grandmasters at once, one as white and the other as black, and relay the moves between them!

In many cases, middleperson attacks are possible but not economic. In the case of car keys, it should certainly be possible to steal a car by having an accomplice follow the driver and electronically relay the radio challenge to you as you work the lock. (One of our students has actually demonstrated

this for our RFID door locks.) But, for the average car thief, it would be a lot simpler to just pick the target's pocket or mug him.

In early 2007, it became clear that there is a practical middleperson attack on the protocols used by the EMV smartcards issued to bank customers in Europe. A bad man could build a wicked terminal that masqueraded, for example, as a parking meter; when you entered your card and PIN to pay a £2.50 parking fee, the transaction could be relayed to a crook loitering near a self-service terminal in a hardware store, who would use a card emulator to order goods. When you get your statement, you might find you've been debited £2,500 for a wide-screen TV [915]. The basic problem here is the lack of a trustworthy user interface on the card; the cardholder doesn't really know which terminal his card is doing business with. I'll discuss such attacks further in the chapter on Banking and Bookkeeping.

3.3.3 Reflection Attacks

Further interesting problems arise with mutual authentication, that is, when two principals have to identify each other. Suppose, for example, that a simple challenge-response IFF system designed to prevent anti-aircraft gunners attacking friendly aircraft had to be deployed in a fighter-bomber too. Now suppose that the air force simply installed one of their air gunners' challenge units in each aircraft and connected it to the fire-control radar. But now an enemy bomber might reflect a challenge back at our fighter, get a correct response, and then reflect that back as its own response:

$$F \longrightarrow B : N$$

$$B \longrightarrow F : N$$

$$F \longrightarrow B : \{N\}_K$$

$$B \longrightarrow F : \{N\}_K$$

So we will want to integrate the challenge system with the response generator. It is still not enough just for the two units to be connected and share a list of outstanding challenges, as an enemy attacked by two of our aircraft might reflect a challenge from one of them to be answered by the other. It might also not be acceptable to switch manually from 'attack' to 'defense' during combat.

There are a number of ways of stopping this 'reflection attack': in many cases, it is sufficient to include the names of the two parties in the authentication exchange. In the above example, we might require a friendly bomber to reply to the challenge:

$$F \longrightarrow B : N$$

with a response such as:

$$B \longrightarrow F : \{B, N\}_K$$

Thus a reflected response $\{F, N\}$ (or even $\{F', N\}$ from the fighter pilot's wingman) could be detected.

This is a much simplified account of IFF, but it serves to illustrate the subtelty of the trust assumptions that underlie an authentication protocol. If you send out a challenge N and receive, within 20 milliseconds, a response $\{N\}_K$, then — since light can travel a bit under 3,730 miles in 20 ms — you know that there is someone with the key K within 2000 miles. But that's all you know. If you can be sure that the response was not computed using your own equipment, you now know that there is someone *else* with the key K within two thousand miles. If you make the further assumption that all copies of the key K are securely held in equipment which may be trusted to operate properly, and you see $\{B, N\}_K$, you might be justified in deducing that the aircraft with callsign B is within 2000 miles. A clear understanding of trust assumptions and their consequences is at the heart of security protocol design.

By now you might think that the protocol design aspects of IFF have been exhaustively discussed. But we've omitted one of the most important problems — and one which the designers of early IFF systems did not anticipate. As radar returns are weak, the signal from the IFF transmitter on board an aircraft will often be audible at a much greater range than the return. The Allies learned this the hard way; in January 1944, decrypts of Enigma messages revealed that the Germans were plotting British and American bombers at twice the normal radar range by interrogating their IFF. So many modern systems authenticate the challenge as well as the response. The NATO mode XII, for example, has a 32 bit encrypted challenge, and a different valid challenge is generated for every interrogation signal, of which there are typically 250 per second. Theoretically there is no need to switch off over enemy territory, but in practice an enemy who can record valid challenges can replay them as part of an attack. Relays are also possible, as with the Mig in the middle.

Many other IFF design problems are less protocol-related, such as the difficulties posed by neutrals, error rates in dense operational environments, how to deal with equipment failure, how to manage keys, and how to cope with multinational coalitions such as that put together for Operation Desert Storm. I'll return to IFF in Chapter 19. For now, the spurious-challenge problem serves to reinforce an important point: that the correctness of a security protocol depends on the assumptions made about the requirements. A protocol that can protect against one kind of attack (being shot down by your own side) but which increases the exposure to an even more likely attack (being shot down by the other side) does more harm than good. In fact, the spurious-challenge problem became so serious in World War II that some experts advocated abandoning IFF altogether, rather than taking the risk that one bomber pilot

in a formation of hundreds would ignore orders and leave his IFF switched on while over enemy territory.

3.4 Manipulating the Message

We've now seen a number of middleperson attacks that reflect or spoof the information used to authenticate a participant's identity — from ATM cards that could be reprogrammed to 'identify' the wrong customer, to attacks on IFF. However, there are more complex attacks where the attacker does not just obtain false identification, but manipulates the message content in some way.

An example is when dishonest cabbies insert pulse generators in the cable that connects their taximeter to a sensor in their taxi's gearbox. The sensor sends pulses as the prop shaft turns, which lets the meter work out how far the taxi has gone. A pirate device, which inserts extra pulses, makes the taxi appear to have gone further. We'll discuss such attacks at much greater length in the chapter on 'Monitoring Systems', in section 12.3.

Another example is a key log attack which defeated many pay-TV systems in Europe in the 1990s and still appears to work in China. The attack is also known as *delayed data transfer*, or DDT. First-generation pay-TV equipment has a decoder, which deciphers the video signal, and a customer smartcard which generates the deciphering keys. These keys are recomputed every few hundred milliseconds by using a one-way encryption function applied to various 'entitlement control messages' that appear in the signal. Such systems can be very elaborate (and we'll discuss some more complex attacks on them later) but there is a very simple attack which works against a lot of them. If the messages that pass between the smartcard and the decoder are the same for all decoders (which is usually the case) then a subscriber can log all the keys sent by his card to his decoder and post it online somewhere. People without a subscription, but who have video-recorded the enciphered program, can then download the key log and use it to decipher the tape.

Changing pay-TV protocols to prevent DDT attacks can be difficult. The base of installed equipment is huge, and many of the obvious countermeasures have an adverse effect on legitimate customers (such as by preventing them videotaping movies). Pay-TV companies generally ignore this attack, since connecting a PC up to a satellite TV decoder through a special hardware adaptor is something only hobbyists do; it is too inconvenient to be a real threat to their revenue stream. In the rare cases where it becomes a nuisance, the strategy is usually to identify the troublesome subscribers and send entitlement control messages that deactivate their cards.

Message-manipulation attacks aren't limited to 'consumer' grade systems. The Intelsat satellites used for international telephone and data traffic have robust mechanisms to prevent a command being accepted twice — otherwise

an attacker could repeatedly order the same manoever to be carried out until the satellite ran out of fuel [1027].

3.5 Changing the Environment

A very common cause of protocol failure is that the environment changes, so that assumptions which were originally true no longer hold and the security protocols cannot cope with the new threats.

One nice example comes from the ticketing systems used by the urban transport authority in London. In the early 1980's, passengers devised a number of scams to cut the cost of commuting. For example, a passenger who commuted a long distance from a suburban station to downtown might buy two cheaper, short distance season tickets — one between his suburban station and a nearby one, and the other between his destination and another downtown station. These would let him get through the barriers, and on the rare occasions he was challenged by an inspector in between, he would claim that he'd boarded at a rural station which had a broken ticket machine.

A large investment later, the system had all the features necessary to stop such scams: all barriers were automatic, tickets could retain state, and the laws had been changed so that people caught without tickets got fined on the spot.

But suddenly the whole environment changed, as the national transport system was privatized to create dozens of rail and bus companies. Some of the new operating companies started cheating each other, and there was nothing the system could do about it! For example, when a one-day travel pass was sold, the revenue was distributed between the various bus, train and subway operators using a formula that depended on where it was sold. Suddenly, the train companies had a motive to book all their ticket sales through the outlet that let them keep the largest percentage. As well as bad outsiders (passengers), we now had bad insiders (rail companies), and the design just hadn't allowed for them. Chaos and litigation ensued.

The transport system's problem was not new; it had been observed in the Italian ski resort of Val di Fassa in the mid-1970's. There, one could buy a monthly pass for all the ski lifts in the valley. An attendant at one of the lifts was observed with a deck of cards, one of which he swiped through the reader between each of the guests. It turned out that the revenue was divided up between the various lift operators according to the number of people who had passed their turnstiles. So each operator sought to inflate its own figures as much as it could [1217].

Another nice example comes from the world of cash machine fraud. In 1993 and 1994, Holland suffered an epidemic of 'phantom withdrawals'; there was much controversy in the press, with the banks claiming that their systems were secure while many people wrote in to the papers claiming to have been

cheated. Eventually the banks were shamed into actively investigating the claims, and noticed that many of the victims had used their bank cards at a certain filling station near Utrecht. This was staked out and one of the staff was arrested. It turned out that he had tapped the line from the card reader to the PC that controlled it; his tap recorded the magnetic stripe details from their cards while he used his eyeballs to capture their PINs [33].

Why had the system been designed so badly? Well, when the standards for managing magnetic stripe cards and PINs were developed in the early 1980's by organizations such as IBM and VISA, the engineers had made two assumptions. The first was that the contents of the magnetic strip — the card number, version number and expiration date — were not secret, while the PIN was [880]. (The analogy used was that the magnetic strip was your name and the PIN your password. I will have more to say on the difficulties of naming below.) The second assumption was that bank card equipment would only be operated in trustworthy environments, such as in a physically robust automatic teller machine, or by a bank clerk at a teller station. So it was 'clearly' only necessary to encrypt the PIN, on its way from the PIN pad to the server; the magnetic strip data could be sent in clear from the card reader.

Both of these assumptions had changed by 1993. An epidemic of card forgery, mostly in the Far East in the late 1980's, drove banks to introduce authentication codes on the magnetic strips. Also, the commercial success of the bank card industry led banks in many countries to extend the use of debit cards from ATMs to terminals in all manner of shops. The combination of these two environmental changes undermined the original system design: instead of putting a card whose magnetic strip contained no security data into a trusted machine, people were putting a card with security data in clear on the strip into an untrusted machine. These changes had come about so gradually, and over such a long period, that the industry didn't see the problem coming.

3.6 Chosen Protocol Attacks

Some firms are trying to sell the idea of a 'multifunction smartcard' — an authentication device that could be used in a wide range of transactions to save you having to carry around dozens of different cards and keys. Governments keen to push ID cards in the wake of 9/11 have tried to get them used for many other transactions; some want a single card to be used for ID, banking and even transport ticketing. Singapore went so far as to experiment with a bank card that doubled as military ID. This introduced some interesting new risks: if a Navy captain tries to withdraw some cash from an ATM after a good dinner and forgets his PIN, will he be unable to take his ship to sea until Monday morning when they open the bank and give him his card back?

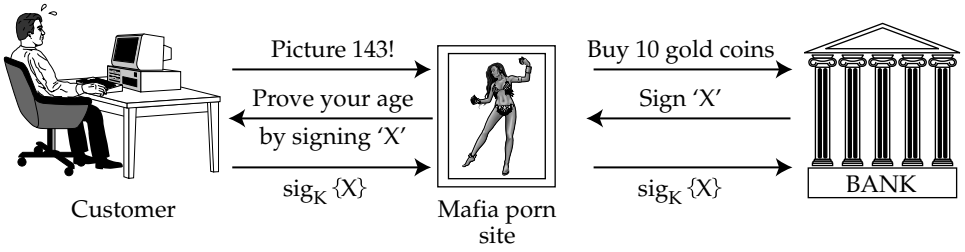

Figure 3.3: The Mafia-in-the-middle attack

Suppose that the banks in Europe were to introduce the CAP protocol to get their customers to authenticate themselves to electronic banking websites, but rather than forcing their customers to fiddle about with a calculator device they just issued all customers with smartcard readers that could be attached to their PC. This would certainly improve convenience and usability. You might think it would improve security too; the EMV protocol enables the card to calculate a message authentication code (MAC) on transaction data such as the amount, merchant number, date and transaction serial number. Message manipulation attacks against electronic banking payments would be prevented.

Or would they? The idea behind the 'Chosen Protocol Attack' is that given a target protocol, you design a new protocol that will attack it if the users can be inveigled into reusing the same token or crypto key. So how might the Mafia design a protocol to attack CAP?

Here's one approach. It used to be common for people visiting a porn website to be asked for 'proof of age,' which usually involves giving a credit card number, whether to the site itself or to an age checking service. If credit and debit cards become usable in PCs, it would be natural for the porn site to ask the customer to authenticate a random challenge as proof of age. A porn site can then mount a 'Mafia-in-the-middle' attack as shown in Figure 3.3. They wait until an unsuspecting customer visits their site, then order something resellable (such as gold coins) from a dealer, playing the role of the coin dealer's customer. When the coin dealer sends them the transaction data for authentication, they relay it through their porn site to the waiting customer. The poor man OKs it, the Mafia gets the gold coins, and when thousands of people suddenly complain about the huge charges to their cards at the end of the month, the porn site has vanished — along with the gold [702].

This is a more extreme variant on the Utrecht scam, and in the 1990s a vulnerability of this kind found its way into international standards: the standards for digital signature and authentication could be run back-to-back in this way. It has since been shown that many protocols, though secure in themselves, can be broken if their users can be inveigled into reusing the same keys in other applications [702]. This is why, for CAP to be secure, it may

well have to be implemented in a stand-alone device into which the customer enters all the transaction parameters directly. Even so, some way has to be found to make it hard for the phishermen to trick the customer into computing an authentication code on data that they supply to the victim. The use of the customer's bank card in the CAP calculator may at least help to bring home that a banking transaction is being done.

In general, using crypto keys (or other authentication mechanisms) in more than one application is dangerous, while letting other people bootstrap their own application security off yours can be downright foolish. If a bank lets its smartcards also be used to load credit into prepayment electricity meters, it would have to worry very hard about whether bad software could be used in electricity vending stations (or even electricity meters) to steal money. Even if those risks could be controlled somehow, liability issues can arise from unplanned or emergent dependencies. A bank that changed its card specification might break the metering system — leaving its customers literally in the dark and risking a lawsuit from the power company. If the bank heeds these risks and tests system changes properly with all the dependant systems, then changes will be much more expensive. Crooks who hack the bank could black out the neighbourhood. The bank might still want to take this risk, though, reckoning that power company customers would be locked in more tightly to the bank, enabling it to charge them more. Security dependencies can have all sorts of strange effects, and we will return to this subject again and again later.

3.7 Managing Encryption Keys

The examples of security protocols that we have discussed so far are mostly about authenticating a principal's name, or application data such as the impulses driving a taximeter. There is one further class of authentication protocols that is very important — the protocols used to manage cryptographic keys. Until recently, such protocols were largely used in the background to support other operations; much of the technology was developed to manage the keys used by cash machines and banks to communicate with each other. But now, systems such as pay-TV use key management to control access to the system directly.

Authentication protocols are now also used in distributed computer systems for general key management purposes, and are therefore becoming ever more important. Kerberos was the first such system to come into widespread use, and a variant of it is used in Windows. I'll now lay the foundations for an understanding of Kerberos.

3.7.1 Basic Key Management

The basic idea behind key distribution protocols is that where two principals want to communicate, they may use a trusted third party to effect an introduction.

When discussing authentication protocols, it is conventional to give the principals human names in order to avoid getting lost in too much algebraic notation. So we will call the two communicating principals 'Alice' and 'Bob', and the trusted third party 'Sam'. But please don't assume that we are talking about human principals. Alice and Bob are likely to be programs while Sam is a server; for example, Alice might be a program in a taximeter, Bob the program in a gearbox sensor and Sam the computer at the taxi inspection station.

Anyway, a simple authentication protocol could run as follows.

1. Alice first calls Sam and asks for a key for communicating with Bob.
2. Sam responds by sending Alice a pair of certificates. Each contains a copy of a key, the first encrypted so only Alice can read it, and the second encrypted so only Bob can read it.
3. Alice then calls Bob and presents the second certificate as her introduction. Each of them decrypts the appropriate certificate under the key they share with Sam and thereby gets access to the new key. Alice can now use the key to send encrypted messages to Bob, and to receive messages from him in return.

Replay attacks are a known problem with authentication protocols, so in order that both Bob and Alice can check that the certificates are fresh, Sam may include a timestamp in each of them. If certificates never expire, there might be serious problems dealing with users whose privileges have been revoked.

Using our protocol notation, we could describe this as

$$\begin{aligned} A \to S &: \quad A, B \\ S \to A &: \quad \{A, B, K_{AB}, T\}_{K_{AS}}, \{A, B, K_{AB}, T\}_{K_{BS}} \\ A \to B &: \quad \{A, B, K_{AB}, T\}_{K_{BS}}, \{M\}_{K_{AB}} \end{aligned}$$

Expanding the notation, Alice calls Sam and says she'd like to talk to Bob. Sam makes up a session key message consisting of Alice's name, Bob's name, a key for them to use, and a timestamp. He encrypts all this under the key he shares with Alice, and he encrypts another copy of it under the key he shares with Bob. He gives both ciphertexts to Alice. Alice retrieves the key from the ciphertext that was encrypted to her, and passes on to Bob the ciphertext encrypted for him. She now sends him whatever message she wanted to send, encrypted using this key.

3.7.2 The Needham-Schroeder Protocol

Many things can go wrong, and here is a famous historical example. Many existing key distribution protocols are derived from the Needham-Schroeder protocol, which appeared in 1978 [960]. It is somewhat similar to the above, but uses nonces rather than timestamps. It runs as follows:

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

Here Alice takes the initiative, and tells Sam: 'I'm Alice, I want to talk to Bob, and my random nonce is N_A.' Sam provides her with a session key, encrypted using the key she shares with him. This ciphertext also contains her nonce so she can confirm it's not a replay. He also gives her a certificate to convey this key to Bob. She passes it to Bob, who then does a challenge-response to check that she is present and alert.

There is a subtle problem with this protocol — Bob has to assume that the key K_{AB} he receives from Sam (via Alice) is fresh. This is not necessarily so: Alice could have waited a year between steps 2 and 3. In many applications this may not be important; it might even help Alice to cache keys against possible server failures. But if an opponent — say Charlie — ever got hold of Alice's key, he could use it to set up session keys with many other principals.

Suppose, for example, that Alice had also asked for and received a key to communicate with Dave, and after Charlie stole her key he sent messages to Sam pretending to be Alice and got keys for Freddie and Ginger. He might also have observed message 2 in her protocol exchanges with Dave. So now Charlie could impersonate Alice to Dave, Freddie and Ginger. So when Alice finds out that her key has been stolen, perhaps by comparing message logs with Dave, she'd have to get Sam to contact everyone for whom she'd ever been issued a key, and tell them that her old key was no longer valid. She could not do this herself as she doesn't know anything about Freddie and Ginger. In other words, revocation is a problem: Sam may have to keep complete logs of everything he's ever done, and these logs would grow in size forever unless the principals' names expired at some fixed time in the future.

Almost 30 years later, this example still generates controversy in the security protocols community. The simplistic view is that Needham and Schroeder just got it wrong; the view argued by Susan Pancho and Dieter Gollmann (for which I have much sympathy) is that this is one more example of a protocol failure brought on by shifting assumptions [538, 1002]. 1978 was a kinder, gentler world; computer security then concerned itself with keeping 'bad guys' out, while nowadays we expect the 'enemy' to be the users of the

system. The Needham-Schroeder paper explicitly assumes that all principals behave themselves, and that all attacks come from outsiders [960]. With these assumptions, the protocol remains sound.

3.7.3 Kerberos

An important practical derivative of the Needham-Schroeder protocol may be found in Kerberos, a distributed access control system that originated at MIT and is now one of the standard authentication tools in Windows [1224]. Instead of a single trusted third party, Kerberos has two kinds: an authentication server to which users log on, and a ticket granting server which gives them tickets allowing access to various resources such as files. This enables more scalable access management. In a university, for example, one might manage students through their halls of residence but manage file servers by departments; in a company, the personnel people might register users to the payroll system while departmental administrators manage resources such as servers and printers.

First, Alice logs on to the authentication server using a password. The client software in her PC fetches a ticket from this server that is encrypted under her password and that contains a session key K_{AS}. Assuming she gets the password right, she now controls K_{AS} and to get access to a resource B controlled by the ticket granting server S, the following protocol takes place. Its outcome is a key K_{AB} with timestamp T_S and lifetime L, which will be used to authenticate Alice's subsequent traffic with that resource:

$$
\begin{array}{ll}
A \to S: & A, B \\
S \to A: & \{T_S, L, K_{AB}, B, \{T_S, L, K_{AB}, A\}_{K_{BS}}\}_{K_{AS}} \\
A \to B: & \{T_S, L, K_{AB}, A\}_{K_{BS}}, \{A, T_A\}_{K_{AB}} \\
B \to A: & \{T_A + 1\}_{K_{AB}}
\end{array}
$$

Translating this into English: Alice asks the ticket granting server for access to B. If this is permissible, the ticket $\{T_S, L, K_{AB}, A\}_{K_{BS}}$ is created containing a suitable key K_{AB} and given to Alice to use. She also gets a copy of the key in a form readable by her, namely encrypted under K_{AS}. She now verifies the ticket by sending a timestamp T_A to the resource, which confirms it's alive by sending back the timestamp incremented by one (this shows it was able to decrypt the ticket correctly and extract the key K_{AB}).

The vulnerability of Needham-Schroeder has been fixed by introducing timestamps rather than random nonces. But, as in most of life, we get little in security for free. There is now a new vulnerability, namely that the clocks on our various clients and servers might get out of synch; they might even be desynchronized deliberately as part of a more complex attack.

3.7.4 Practical Key Management

So we can use a protocol like Kerberos to set up and manage working keys between users given that each user shares one or more long-term keys with a server that acts as a key distribution centre. I'll describe a number of other similar protocols later; for example, in the chapter on 'Banking and Bookkeeping' I'll discuss how a bank can set up a long-term key with each of its ATMs and with each of the interbank networks with which it's associated. The bank then uses protocols not too unlike Kerberos to establish a 'key of the day' with each ATM and with each network switch; so when you turn up at the ATM belonging to a foreign bank and ask for money from your own bank via the Cirrus network, the ATM will encrypt the transaction using the working key it shares with the bank that owns it, and the bank will then pass on the transaction to Cirrus encrypted with the key of the day for that network.

So far so good. But a moment's thought will reveal that the bank has to maintain several keys for each of the several hundred ATMs that it owns — a long-term master key, plus perhaps an encryption key and an authentication key; several keys for each of the several dozen bank networks of which it's a member; passwords and other security information for each of several million electronic banking customers, and perhaps keys for them as well if they're given client software that uses cryptography. Oh, and there may be encrypted passwords for each of several thousand employees, which might also take the form of Kerberos keys encrypted under user passwords. That's a lot of key material. How is it to be managed?

Key management is a complex and difficult business and is often got wrong because it's left as an afterthought. A good engineer will sit down and think about how many keys are needed, how they're to be generated, how long they need to remain in service and how they'll eventually be destroyed. There is a much longer list of concerns — many of them articulated in the Federal Information Processing Standard for key management [948]. In addition, things go wrong as applications evolve; it's important to provide extra keys to support next year's functionality, so that you don't compromise your existing ones by reusing them in protocols that turn out to be incompatible. It's also important to support recovery from security failure. Yet there are no standard ways of doing either.

As for practical strategies, there are a number — none of them straightforward. Public-key crypto, which I'll discuss in Chapter 5, can slightly simplify the key management task. Long-term keys can be split into a private part and a public part; you don't have to keep the public part secret (as its name implies) but you do have to guarantee its integrity. In banking the usual answer is to use dedicated cryptographic processors called security modules, which I'll describe in detail in the chapter on 'Tamper Resistance'. These do all the cryptography and contain internal keys with which application keys are protected.

Thus you get your security module to generate master keys for each of your ATMs; you store their encrypted values in your ATM master file. Whenever a transaction comes in from that ATM, you retrieve the encrypted key from the file and pass it to the security module along with the encrypted data. The module then does what's necessary: it decrypts the PIN and verifies it, perhaps against an encrypted value kept locally. Unfortunately, the protocols used to set all this up are also liable to failure. Many attacks have been found that exploit the application programming interface, or API, of the security module, where these protocols are exposed. I will describe these attacks in detail in the chapter on API Security. For now, it's enough to note that getting security protocols right is hard. You should not design them at home, any more than you design your own explosives.

3.8 Getting Formal

Subtle difficulties of the kind we have seen with the above protocols, and the many ways in which protection properties depend on quite subtle starting assumptions that protocol designers may get wrong (or that may be misunderstood later), have led researchers to apply formal methods to key distribution protocols. The goal of this exercise was originally to decide whether a protocol was right or wrong: it should either be proved correct, or an attack should be exhibited. More recently this has expanded to clarifying the assumptions that underlie a given protocol.

There are a number of different approaches to verifying the correctness of protocols. The best known is the *logic of belief*, or *BAN logic*, named after its inventors Burrows, Abadi and Needham [249]. It reasons about what a principal might reasonably believe having seen of certain messages, timestamps and so on. A second is the *random oracle model*, which I touch on in the chapter on cryptology and which is favored by people working on the theory of cryptography; this appears less expressive than logics of belief, but can tie protocol properties to the properties of the underlying encryption algorithms. Finally, a number of researchers have applied mainstream formal methods such as CSP and verification tools such as Isabelle.

Some history exists of flaws being found in protocols that had been proved correct using formal methods; the following subsection offers a typical example.

3.8.1 A Typical Smartcard Banking Protocol

The COPAC system is an electronic purse used by VISA in countries with poor telecommunications [48]. It was the first live financial system whose underlying protocol suite was designed and verified using such formal techniques, and

in particular a variant of the BAN logic. A similar protocol is now used in the 'Geldkarte,' an electronic purse issued by banks in Germany, and adopted also by French banks as 'Moneo'. There's also a system in Belgium called 'Proton'. The European applications focus on low-value transactions with devices such as parking meters and vending machines for which it may not be economical to provide a network connection.

Transactions take place from a customer smartcard to a merchant smartcard (which in the case of a vending machine is kept in the machine and changed when it's replenished). The customer gives the merchant an electronic check with two authentication codes on it; one that can be checked by the network, and one that can be checked by the customer's bank. A simplified version of the protocol is as follows.

$$\begin{array}{ll} C \longrightarrow R: & \{C, N_C\}_K \\ R \longrightarrow C: & \{R, N_R, C, N_C\}_K \\ C \longrightarrow R: & \{C, N_C, R, N_R, X\}_K \end{array}$$

In English: the customer and the retailer share a key K. Using this key, the customer encrypts a message containing its account number C and a customer transaction serial number N_C. The retailer confirms its own account number R and his own transaction serial number N_R, as well as the information it's just received from the customer. The customer now sends the electronic check X, along with all the data exchanged so far in the protocol. One can think of the electronic check as being stapled to a payment advice with the customer's and retailer's account numbers and their respective reference numbers. (The reason for repeating all previous data in each message is to prevent message manipulation attacks using cut-and-paste.)

3.8.2 The BAN Logic

The BAN logic provides a formal method for reasoning about the beliefs of principals in cryptographic protocols. Its underlying idea is that we will believe that a message is authentic if it is encrypted with a relevant key and it is also fresh (that is, generated during the current run of the protocol). Further assumptions include that principals will only assert statements they believe in, and that some principals are authorities for certain kinds of statement. This is formalized using a notation which includes:

$A \mid\equiv X$ *A believes* X, or, more accurately, that A is entitled to believe X;

$A \mid\sim X$ *A once said* X (without implying that this utterance was recent or not);

$A \mid\Rightarrow X$ *A has jurisdiction over* X, in other words A is the authority on X and is to be trusted on it;

$A \triangleleft X$ *A sees X*, that is, someone sent a message to A containing X in such a way that he can read and repeat it;

$\sharp X$ *X is fresh*, that is, contains a current timestamp or some information showing that it was uttered by the relevant principal during the current run of the protocol;

$\{X\}_K$ *X encrypted under the key K*, as in the rest of this chapter;

$A \leftrightarrow^K B$ *A and B share the key K*, in other words it is an appropriate key for them to use to communicate.

There are further symbols dealing, for example, with public key operations and with passwords, that need not concern us here.

These symbols are manipulated using a set of postulates which include:

the message meaning rule states that if A sees a message encrypted under K, and K is a good key for communicating with B, then he will believe that the message was once said by B. (We assume that each principal can recognize and ignore his or her own messages.) Formally, $\dfrac{A \mid\equiv A \leftrightarrow^K B, A \triangleleft \{X\}_K}{A \mid\equiv B \mid\sim X}$

the nonce-verification rule states that if a principal once said a message, and the message is fresh, then that principal still believes it. Formally, $\dfrac{A \mid\equiv \sharp X, A \mid\equiv B \mid\sim X}{A \mid\equiv B \mid\equiv X}$

the jurisdiction rule states that if a principal believes something, and is an authority on the matter, then he or she should be believed. Formally, we write that $\dfrac{A \mid\equiv B \mid\Rightarrow X, A \mid\equiv B \mid\equiv X}{A \mid\equiv X}$

In this notation, the statements on the top are the conditions, and the one on the bottom is the result. There are a number of further rules to cover the more mechanical aspects of manipulation; for example, if A sees a statement then he sees its components provided he knows the necessary keys, and if part of a formula is known to be fresh, then the whole formula must be.

3.8.3 Verifying the Payment Protocol

Assuming that the key K is only available to principals who can be trusted to execute the protocol faithfully, formal verification is now straightforward. The trick is to start from the desired result and work backwards. In this case, we wish to prove that the retailer should trust the check, i.e., $R \mid\equiv X$ (the syntax of checks and cryptographic keys is similar for our purposes here; a check is good if and only if it is genuine and the date on it is sufficiently recent).

Now $R \mid\equiv X$ will follow under the jurisdiction rule from $R \mid\equiv C \mid\Rightarrow X$ (R believes C has jurisdiction over X) and $R \mid\equiv C \mid\equiv X$ (R believes C believes X).

The former condition follows from the hardware constraint, that no-one except C could have uttered a text of the form $\{C, \ldots\}_K$.

The latter, that $R \mid\equiv C \mid\equiv X$, must be deduced using the nonce verification rule from $\sharp X$ (X is fresh) and $R \mid\equiv C \mid\sim X$ (R believes C uttered X).

$\sharp X$ follows from its occurrence in $\{C, N_C, R, N_R, X\}_K$ which contains the sequence number N_R, while $R \mid\equiv C \mid\sim X$ follows from the hardware constraint.

The above summary of the proof is, of necessity, telegraphic. If you want to understand logics of authentication in detail, you should consult the original papers [48] and see the recommendations for further reading at the end of this chapter.

3.8.4 Limitations of Formal Verification

Formal methods can be an excellent way of finding bugs in security protocol designs as they force the designer to make everything explicit and thus confront difficult design choices that might otherwise be fudged. However, they have their limitations, too.

One problem is in the external assumptions we make. For example, we assumed that the key wasn't available to anyone who might use it in an unauthorized manner. In practice, this is not always true. Although our purse protocol is executed in tamper-resistant smartcards, their software can have bugs, and in any case the tamper-resistance they offer is never complete. (I'll discuss this in the chapter on Tamper Resistance.) So the system has various fallback mechanisms to detect and react to card forgery, such as shadow accounts which track the amount of money that should be on each card and which are updated as transactions are cleared. It also has lists of hot cards that are distributed to terminals; these are needed anyway for stolen cards, and can be used for forged cards too.

Second, there are often problems with the idealisation of the protocol. An interesting flaw was found in an early version of this system. The key K actually consisted of two keys — the encryption was done first with a 'transaction key' which was diversified (that is, each card had its own variant) and then again with a 'bank key', which was not diversified. The former was done by the network operator, and the latter by the bank which issued the card. The reasons for this included dual control, and to ensure that even if an attacker managed to drill the keys out of a single card, he would only be able to forge that card, not make forgeries which would pass as other cards (and thus defeat the hot card mechanism). But since the bank key was not diversified, it must be assumed to be known to any attacker who has broken a card. This means that he can undo the outer wrapping of encryption, and in some circumstances message replay was possible. (The bank key was diversified in a later version before any villains discovered and exploited the flaw.)

In this case there was no failure of the formal method, as no attempt was ever made to verify the diversification mechanism. But it does illustrate a common problem in security engineering — that vulnerabilities arise at the boundary between two protection technologies. In this case, there were three technologies: the hardware tamper resistance, the authentication protocol and the shadow account / hot card list mechanisms. Different protection technologies are often the domain of different experts who don't completely understand the assumptions made by the others. (In fact, that's one reason security engineers need a book such as this one: to help subject specialists understand each others' tools and communicate with each other more effectively.)

For these reasons, people have explored alternative ways of assuring the design of authentication protocols, including the idea of *protocol robustness*. Just as structured programming techniques aim to ensure that software is designed methodically and nothing of importance is left out, so robust protocol design is largely about explicitness. Robustness principles include that the interpretation of a protocol should depend only on its content, not its context; so everything of importance (such as principals' names) should be stated explicitly in the messages. There are other issues concerning the freshness provided by serial numbers, timestamps and random challenges, and on the way encryption is used. If the protocol uses public key cryptography or digital signature mechanisms, there are further more technical robustness issues.

3.9 Summary

Passwords are just one (simple) example of a more general concept, the security protocol. Protocols specify the series of steps that principals use to establish trust relationships in a system, such as authenticating a claim to identity, demonstrating ownership of a credential, or granting a claim on a resource. Cryptographic authentication protocols, whether one-pass (e.g., using random nonces) or two-pass (challenge-response) are used for a wide range of such purposes, from basic entity authentication to provide infrastructure for distributed systems that allows trust to be taken from where it exists to where it is needed. Security protocols are fielded in all sorts of systems from remote car door locks through military IFF systems to authentication in distributed computer systems.

It is difficult to design effective security protocols. They suffer from a number of potential problems, including middleperson attacks, modification attacks, reflection attacks, and replay attacks. These threats can interact with implementation vulnerabilities such as poor random number generators. Using mathematical techniques to verify the correctness of protocols can help, but it won't catch all the bugs. Some of the most pernicious failures are caused by creeping changes in the environment for which a protocol was designed, so that the protection it gives is no longer adequate.

Research Problems

At several times during the past 20 years, some people have thought that protocols had been 'done' and that we should turn to new research topics. They have been repeatedly proved wrong by the emergence of new protocol applications with a new crop of errors and attacks to be explored. Formal methods blossomed in the early 1990s, then key management protocols; during the mid-1990's the flood of proposals for electronic commerce mechanisms kept us busy; and in the later 1990's a whole series of mechanisms proposed for protecting copyright on the Internet provided us with targets. Since 2000, one strand of protocol research has acquired an economic flavour as security mechanisms are used more and more to support business models; the designer's 'enemy' is often a commercial competitor, or even the customer. Another has applied protocol analysis tools to look at the security of application programming interfaces (APIs), a topic to which I'll return later.

Will people continue to develop faulty protocols which other people attack, or will we manage to develop a methodology for designing them right first time? What are the exact uses and limitations of formal methods, and other mathematical approaches such as the random oracle model?

At the system level, how do we manage the tension between the principle that robust protocols are generally those in which everything is completely specified and checked (principals' names, roles, security policy statement, protocol version, time, date, sequence number, security context, maker of grandmother's kitchen sink) and the system engineering principle that a good specification should not overconstrain the implementer?

Further Reading

Research papers on security protocols are scattered fairly widely throughout the literature. The main introductory papers to read are probably the original Needham-Schroeder paper [960]; the Burrows-Abadi-Needham authentication logic [249]; papers by Abadi and Needham, and Anderson and Needham, on protocol robustness [2, 73]; and there is a survey paper by Anderson and Needham [74]. In [707] there is an analysis of a defective security protocol, carried out using three different formal methods. Beyond that, the proceedings of the security protocols workshops [290, 291] provide leads to current research, and there are many papers scattered around a wide range of conferences.