CHAPTER

2

Protocols

It is impossible to foresee the consequences of being clever.

—CHRISTOPHER STRACHEY

If security engineering has a unifying theme, it is the study of security protocols. Rather than starting off with a formal definition of a security protocol, I will give a rough indication, then refine it using a number of examples. As this is an engineering book, I will also give several 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. 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; or they may be very complex. The world's networks of cash machines have dozens of protocols specifying how a cash machine interacts with customers, how it 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 principals, 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, in the past, 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 realized that this also enabled him to loot any other customer's account, and he went on to steal hundreds of thousands over a period of years. The affected banks had to spend millions on changing their systems.

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

2.1 Password Eavesdropping Risks

Passwords are still the foundation on which much of computer security rests, as they are the main mechanism used to authenticate human users to computer systems. In the form of PINs, they are also used in many embedded systems, from cash machines through mobile phones to burglar alarms. They raise many problems, such as the difficulty people have in choosing passwords that are difficult to guess, or remembering passwords generated randomly by the system.

We discuss the "human interface" problems of passwords in the next chapter. For now, let us consider the limitations of embedded systems that use passwords. The typical application is the remote control used to open your garage or to unlock the doors of cars manufactured up to the mid-1990s. These primitive remote controls just broadcast their 16-bit 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 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.

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, 16-bit passwords are too short. In the mid-1990s, devices appeared that could try all possible codes one after the other. A code would be found on average after about 2^{15} tries, which at 10 per second would take less than an hour. A thief operating in a parking lot with a hundred vehicles within range could be rewarded in less than a minute with a car helpfully flashing its lights.

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 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 in that there may be many people who have 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; this is even worse, as the serial number may appear on the order, the delivery note, the invoice, and all the other standard 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 the turnstile is attended and the attendants get to know the "regulars," so there is no need for anything more expensive. My cardkey for getting into the laboratory where I work is slightly harder to forge, as it uses an infrared barcode. Again, this is probably quite adequate—our more expensive equipment is in rooms with additional door locks. We'll discuss passwords in more detail in Chapter 3. But for things that lots of people want to steal, like cars, a better technology is needed. This brings us to cryptographic authentication protocols.

2.2 Who Goes There? Simple Authentication

A simple example of an authentication device is an infrared token used in some multi-storey parking garages to enable subscribers to raise the barrier. This first transmits its serial number and then transmits an authentication block that consists of the same serial number, followed by a random number, all encrypted using a key that is unique to the device.

I will postpone discussion of how to encrypt data and what properties the cipher should have; here, I 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*. The purpose of 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 plaintext contains T, and, finally, that the nonce N has not been seen before.

One reason many people get confused is that to the left of the colon, T identifies one of the principals (the token that represents the subscriber), whereas to the right it means the name (that is, the serial number) of the token. Another cause of confusion 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 thoroughly entrenched now to change easily. Professionals often think of the $T \rightarrow G$ to the left of the colon as simply a hint of what the protocol designer had in mind.

The term *nonce* can mean anything that guarantees the freshness of a message. A nonce may, according to the context, be a random number, a serial number, or a random challenge received from a third party. There are subtle differences between the three approaches, such as in the level of resistance they offer to various kinds of replay attack. (We'll discuss these later.) 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.

Key management in such devices can be simple. A typical garage token's key *KT* 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 gives a very simple 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.

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. In one car lock, the thief could open the door by replaying the last-but-one code. Another example comes from the world of prepayment utility meters. Over a million households in the United Kingdom, plus many millions in developing countries, have an electricity or gas meter designed so that they can purchase encrypted tokens to take home and insert into the meter, which then dispenses the purchased quantity of electricity or gas. 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 [39].

The question of whether to use a random number or a counter is not as easy as it might seem [195]. With random numbers, the lock has to remember a reasonable number of past codes. There's also the *valet attack*; 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.

The problem with counters is maintaining synchronization. A key may be used for more than one lock, and may also be activated 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 synchronize on, a key under certain conditions; but the details are not always designed thoughtfully. One common product uses a 16-bit counter, and allows access when the counter value that is deciphered is the last valid code incremented by no more than 16. To cope with cases where the token has been used more than 16 times elsewhere (or chewed 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 opens it to a replay attack, because someone only needs six access codes—say for values 0, 1, 20,000, 20,001, 40,000 and 40,001 to break the system completely.

So designing even a simple token authentication mechanism is not 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.

An example that may become contentious is *accessory control.* It is common for the makers of games consoles to build in challenge-response protocols to prevent software cartridges or other accessories being used with their product unless a license fee is paid. This practice is spreading. According to one vendor of authentication chips, some printer companies have begun to embed authentication in printers to ensure that genuine toner cartridges are used. If a competitor's product is loaded instead, the printer will quietly downgrade from 1200 dpi to 300 dpi. In mobile phones, much of the profit is made on batteries, and authentication protocols can be used to spot competitors' products so they can be drained more quickly. (I wonder how long it will be before the research that toner cartridge and battery manufactures will do to defeat these systems will hit the street in the form of better car theft tools?)

2.2.1 Challenge and Response

The most modern car door locks use a more sophisticated two-pass protocol, often called *challenge-response.* As the car key is inserted into the steering lock, the engine management unit sends a challenge, consisting of a random *n*-bit number to the key using a short-range radio signal. The car key computes a response by encrypting the challenge. In this way, 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:

$$E \rightarrow T : N$$
$$T \rightarrow E : \{T, N\}_K$$

This is still not bulletproof. In one system, the random numbers generated by the engine management unit turned out to be rather 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, most of the widely used software products that incorporate encryption—including Kerberos, Netscape, and PGP—have been broken at some time or another because their random-number generators weren't random enough [340, 256]. The fix used 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 environmental concerns. There are various sources of randomness that can be used in large systems such as PCs; for example, it's possible to use the small variations in the rotational velocity of the hard disk caused by air turbulence [225]. Practical systems for PCs often mix the randomness available from a number of environmental sources, such as network traffic and keystroke timing, and from internal system sources [363]; the way these sources are combined is often critical [447]. But in a typical embedded system such as a car lock, the random challenge is generated by encrypting a counter using a special key that is kept inside the device, and not used for any other purpose.

Locks are not the only application of challenge-response protocols. Many organizations—including most U.S. banks, many phone companies, and a number of defense agencies—issue their staff password generators that enable them to log on to corporate computer systems [808]. These may look like calculators (and 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 11 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. (See Figure 2.1.) If you have a password generator with the right secret key, and you enter the PIN right, and you type 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.

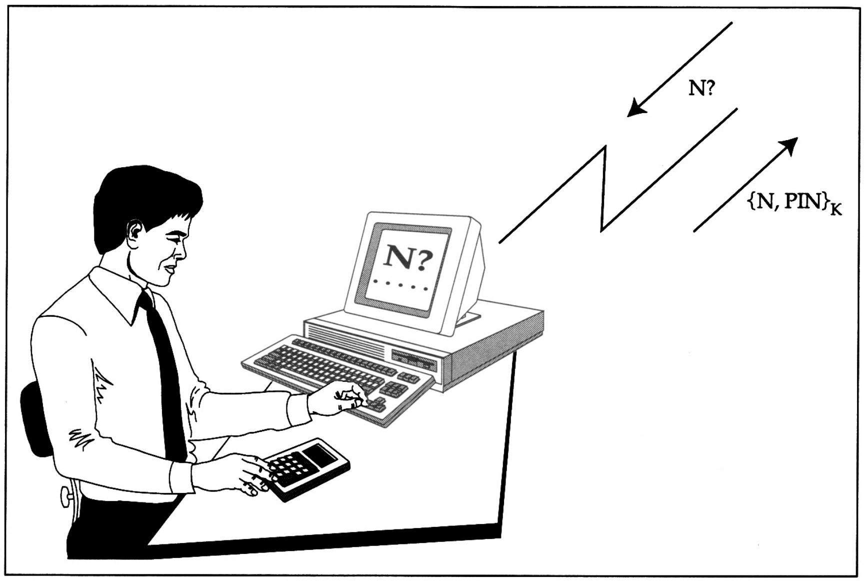

Figure 2.1 Password generator use.

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:

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

(For a more detailed description of one of the more popular challenge-response products, see [15, p. 211 ff].)

The encryption in challenge-response protocols does not always need to be invertible, and so in general it can be accomplished using a "one-way function" or "cryptographic hash function," which has the property that it's less subject to export restrictions than are encryption algorithms. (For its technical properties, see Chapter 5, "Cryptology.")

2.2.2 The MIG-in-the-Middle Attack

There is an interesting attack on challenge-response systems that appears to have played a role in bringing peace to Southern Africa.

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 *identify-friend-or-foe* (IFF) systems. 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.

U.S. aircraft use an IFF system called 'Mode XII,' and systems are under development for ground troops too. But the South African Air Force (SAAF) had been cut off from Western arms supplies by sanctions and had to design its own system.

In the late 1980s, South African troops were fighting a war in northern Namibia and southern Angola. The goals were to keep Namibia under white rule and to impose a client government (UNITA) on Angola. Because the South African Defense Force consisted largely of conscripts from a small, white population, it was essential to limit casualties. So, most South African troops 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, Cuban aircraft broke through the South African air defenses and bombed a South African camp in northern Namibia, killing a number of white conscripts. This proof that its 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 have been the last successful military operation ever carried out by Soviet bloc forces.

Some years afterward, a former 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, which retransmitted them and were allowed through (see Figure 2.2). 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 until then.

I have no independent confirmation on this story from the Angolan or Cuban side. But the basic technique is at least as old as World War II, and illustrates the basic idea behind an attack known to the cryptographic community as the *man-in-the-middle* or (more recently) the *middleperson* attack. We will come across it 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 beat 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!

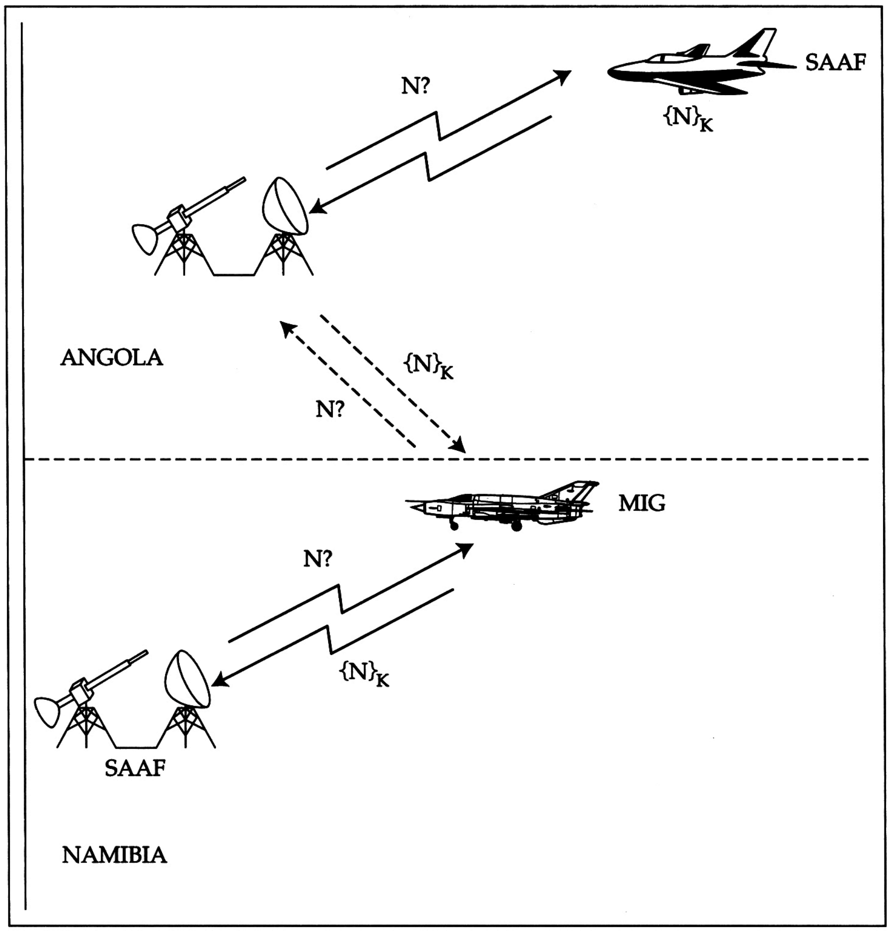

Figure 2.2 The MIG-in-the middle attack.

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. But it would be a lot simpler to just pick the driver's pocket or mug him.

2.2.3 Reflection Attacks

Other interesting problems arise with *mutual authentication*, that is, when two principals have to identify each other. Suppose, that a simple challenge-response IFF system designed to prevent anti-aircraft gunners attacking friendly aircraft also had to be deployed in a fighter-bomber. Now suppose that the air force simply installed one of its 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:

$$\begin{aligned} F &\rightarrow B : N \\ B &\rightarrow F : N \\ F &\rightarrow B : \{N\}_K \\ B &\rightarrow F : \{N\}_K \end{aligned}$$

So we will want to integrate the challenge system with the response generator. It is still not enough 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. Likewise, it might not be acceptable to switch manually from "attack" to "defense" mode during air 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 previous example, we might require a friendly bomber to reply to the challenge:

$$F \rightarrow B : N$$

with a response such as:

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

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 different trust assumptions that underlie an authentication protocol. If you send out a challenge N and receive, within 20 milliseconds, a response $\{N\}_K$, then, because light can travel a bit under 3,730 miles in 20 ms, you know that there is someone with the key K within 2,000 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 2,000 miles. If you make the further assumption that all copies of the key K are securely held in equipment that 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 2,000 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.

There are many other aspects of IFF which 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. We'll return to IFF in Chapter 16. 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.

2.3 Manipulating the Message

One kind of middleperson attack is often treated as a separate category of attack. This is where the attacker does not just reflect identification information, but manipulates the message content in some way. We saw an example at the beginning of this chapter: ATM cards designed for offline operation could be manipulated in order to steal money. In effect, the magnetic card acted as a store-and-forward communication channel between the bank's mainframe computer and its cash machines whenever the phone lines (or the mainframe) were down.

Another 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 let 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 Chapter 10, "Monitoring Systems." Section 10.4.

However, many application-level message manipulation attacks are really just variants on the replay attack, which we saw previously. They aren't limited to low-grade systems, such as remote door locks that can be defeated by recording and replaying a fixed password. 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 maneuver to be carried out until the satellite ran out of fuel [617].

Another example is a key log attack, which defeats many pay-TV systems (it's also known as *delayed data transfer*, or DDT). Typical pay-TV equipment has a decoder which deciphers the video signal and a customer smartcard which generates the deciphering keys. These keys are recomputed several times a second 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 that 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 subscribers can record logs of all the keys sent by their cards to their decoders, and post them to the Net. Someone without a subscription, but who has 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 from videotaping movies). Pay-TV companies generally ignore this attack, since connecting a PC 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.

2.4 Changing the Environment

A very common cause of protocol failure is that the environment changes, so that assumptions that 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 London Transport. In the early 1980s, 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 their suburban station and a nearby one, and the other between their destination and another downtown station. These would let them get through the barriers; on the rare occasions they were challenged by an inspector in between, they would claim that they'd boarded at a rural station that 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 then the whole environment changed, as parts of the system were 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. 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-1970s. 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 [730].

Another relevant 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 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 [19].

Why had the system been designed so badly? Well, when the standards for managing magnetic stripe cards and PINs were developed in the early 1980s, 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—was not secret, while the PIN was [548]. (The analogy used was that the magnetic strip was the holder's name and the PIN their password. We will have more to say on the subtleties of naming later.) 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 1980s, 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 that did rely on security data in 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.

2.5 Chosen Protocol Attacks

Some people 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 users having to carry around dozens of different cards and keys.

This introduces some interesting new risks. Suppose that you use your card to sign bank transactions; a common way of doing this would be to have the card compute a digital signature on the transaction data. In fact, to save on computation, the signature is usually computed on a random-looking 20-byte digest of the transaction. (We'll discuss in Chapter 5 how to compute such digests.) Now suppose that this card can be used by any other application that anyone cares to design. How might the Mafia design a protocol to attack it?

Here's one example. At present people visiting a Web porn site are often 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 cards become able to do digital signatures, it would be natural for the porn site to ask the customer to sign a random challenge as proof of age. A porn site could then mount a "*Mafia-in-the-middle*" attack, as shown in Figure 2.3. The perpetrators 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 digest for signature, they relay it through their porn site to the waiting customer in the form of a random challenge. The customer signs 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 [446].

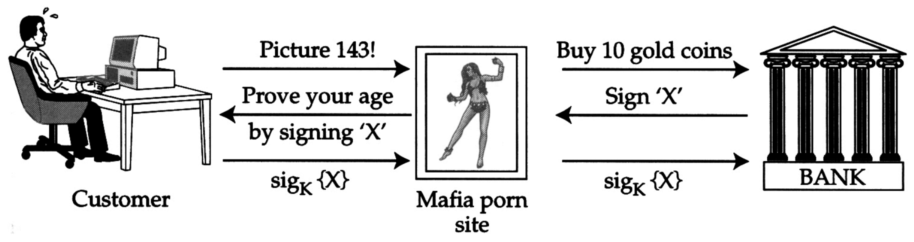

Figure 2.3 The Mafia-in-the-middle attack.

This is a more extreme variant on the Utrecht scam. There are several lessons: using crypto keys (or other authentication mechanisms) in more than one application can be dangerous; and letting other people bootstrap their own application security off yours can be downright foolish.

2.6 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, therefore are going to be very important. Kerberos was the first such system to come into widespread use, and a variant of it is used in Windows 2000. I'll now lay the foundations for an understanding of Kerberos.

2.6.1 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.

I remarked that in the literature on authentication protocols, it is conventional to give the principals human names to avoid getting lost in too much algebraic notation. So I 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; 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.

I mentioned that 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 could well be serious problems dealing with users whose privileges have been revoked.

Using our protocol notation, we could describe this as:

$$A \rightarrow S : A, B$$
$$S \rightarrow A : \{A, B, K_{AB}, T\}_{K_{AS}}, \{A, B, K_{AB}, T\}_{K_{BS}}$$
$$A \rightarrow B : \{A, B, K_{AB}, T\}_{K_{BS}}, \{M\}_{K_{AB}}$$

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. Sam encrypts this under the key he shares with Alice, and with 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.

2.6.2 The Needham-Schroeder Protocol

Many things can go wrong. We will see plenty of examples later; for now, a famous historical example will suffice. Many existing key distribution protocols are derived from a protocol invented by Roger Needham and Mike Schroeder in 1978 [589]. It is somewhat similar to the one we've just discussed, but uses nonces rather than timestamps. It runs as follows:

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: "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 the certificate 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 K_{AS} 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 Dorothy, 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 Dorothy, that is, when Sam sent her a key for communicating with Dorothy, encrypted under the key K_{AS} which is now compromised. So now Charlie could impersonate Alice to Dorothy, and also to Freddie and Ginger. So when Alice finds out that her key has been stolen, perhaps by comparing message logs with Dorothy, she'd have to get Sam 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 will probably have to keep complete logs of everything he has ever done, and these logs would grow in size forever unless the principals' names expired at some fixed time in the future.

Over 20 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 [345, 600]. 1978 was a kinder, gentler world; computer security then concerned itself with keeping the bad guys out, while nowadays we expect the bad guys to be users of the system. The Needham-Schroeder paper explicitly assumes that all principals behave themselves, and that attacks come only from outsiders [589]. Under these assumptions, the protocol remains sound.

2.6.3 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 the default authentication option in Windows 2000 [735]. Instead of a single trusted third party, Kerberos has two kinds: an authentication server to which users log on, and a ticket-granting server that 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, which is encrypted under her password and which contains a session key K_{AS}. Assuming she gets the password right, she now controls K_{AS}; 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:

$$
\begin{aligned}
&A \square\ S : A, B \\
&S \square\ A : \{T_S, L, K_{AB}, B, \{T_S, L, K_{AB}, A\}_{KBS}\}_{KAS} \\
&A \square\ B : \{T_S, L, K_{AB}, A\}_{KBS}, \{A, T_A\}_{KAB} \\
&B \square\ A : \{T_A{+}1\}_{KAB}
\end{aligned}
$$

Translating this into English: Alice asks the ticket-granting server for access to B. If this is permissible, the ticket $\{T_S, L, K_{AB}, A\}_{KBS}$ 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 its aliveness by sending back the timestamp incremented by one (this is a convention to indicate that the resource 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.

2.7 Getting Formal

Subtle difficulties of the kind we have seen with the protocols just discussed, and the many ways in which protection properties depend on quite narrow (and often unobvious) starting assumptions, 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. Either it should 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 a logic of belief, the *BAN logic*, named after its inventors Mike Burrows, Martín Abadi, and Roger Needham [148]. It reasons about what is reasonable for a principal to believe, given sight of certain messages, timestamps, and so on. A second is the *random oracle model*, which I touch on in Chapter 5, and which is favored by many mathematicians working at the theoretical end of the subject; this appears to be less expressive than logies of belief, but can tie protocol properties down to the properties of the underlying encryption algorithms. Finally, a number of researchers have applied mainstream formal methods such as CSP and Lotos.

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

2.7.1 A Typical Smartcard Banking Protocol

This system, currently called COPAC, is an electronic purse system used by VISA in countries with poor telecommunications [35]. 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 very similar protocol is now also used in the "Geldkarte," an electronic purse issued by banks in Germany to their clients.

Transactions take place from a customer smartcard to a merchant smartcard. 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:

$$\begin{aligned} C &\rightarrow R : \{C, N_C\}_K \\ R &\rightarrow C : \{R, N_R, C, N_C\}_K \\ C &\rightarrow R : \{C, N_C, R, N_R, X\}_K \end{aligned}$$

In English: the customer and the retailer share a key, K. Using this key, the customer card encrypts a message containing its account number, C, and a customer transaction serial number, N_C. The retailer confirms its name, R, and its transaction serial number, N_R, as well as the information it has 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 names, 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.)

2.7.2 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 that 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 A can read and repeat it.

- $\#X$: *X is fresh*; that is, X 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.
- $A \square^{K} B$: *A and B share the key K*; in other words, it is an appropriate key for them to use to communicate.

Other symbols deal, for example, with public key operations and with passwords, but they do 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 A 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:

$$\frac{A \mid\equiv A \square^{K} B,\ A \lhd \{X\}_K}{A \mid\equiv B \mid\square X}$$

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:

$$\frac{A \mid\equiv \#X,\ A \mid\equiv B \mid\square X}{A \mid\equiv B \mid\equiv X}$$

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:

$$\frac{A \mid\equiv B \mid\square\ X,\ A \mid\equiv B \mid\equiv X}{A \mid\equiv X}$$

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

2.7.3 Verifying the Payment Protocol

Assuming that the key, K, is available only 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 backward. In this case, we wish to prove that the retailer should trust the check; that is, $R \mid\equiv X$ (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 fresh).

Now $R \mid\equiv X$ will follow under the jurisdiction rule from $R \mid\equiv C \mid\square\ X$ (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 $\#X$ (X is fresh) and $R \mid\equiv C \mid\square X$ (R believes C uttered X).

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

This summary of the proof is, of necessity, telegraphic. If you want to understand logics of authentication in detail, you should consult the original papers, and refer to the recommendations for further reading at the end of this chapter.

2.7.4 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 the COPAC 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 explain this in Chapter 14, "Physical Tamper Resistance.") So the system has various fallback mechanisms for detecting and reacting 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 idealization of the protocol. A well-known example comes from the application of the BAN logic to protocols using public key cryptography; a version of the message meaning rule which only applies to digital signature was erroneously thought to apply to decryption as well, leading to a positive verification of a flawed protocol. Another example is given by a flaw found in an early version of the COPAC system. There 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), then again with a "bank key," which was not diversified. The former was done by the network operator and the latter by the bank that issued the card. The reasons for this included dual control and to ensure that an attacker who managed to drill the keys out of a single card would only be able to forge that card, not make forgeries that would pass as other cards (and thus defeat the hot card mechanism). But since the bank key was not diversified, it would be known to any attacker who has broken a card. This means that the attacker could undo the outer wrapping of encryption; and in some circumstances, message replay was possible. (The bank key was diversified in later versions 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. (That's one reason security engineers need a book such as this one: to help subject specialists understand each others' tools and to 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 that 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; thus, 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.

2.8 Summary

Passwords are not always an adequate means of protection, especially if they have to be used more than once over an open communications channel. Simple authentication protocols, whether one-pass (e.g., using random nonces) or two-pass (challenge-response) are appropriate in many cases, and 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

During the past few years, some people have thought that protocols had been "done," and that we should turn to new research topics. These people have been repeatedly proved wrong by the emergence of new protocol applications, with a new crop of errors and attacks to be explored. Key management protocols were a focus of research in the early 1990s; during the mid-1990s, the flood of proposals for electronic commerce mechanisms kept us busy; and in the later 1990s, a whole series of mechanisms proposed for protecting copyright on the Internet provided us with targets.

Will we continue to develop faulty protocols that 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 [589]; the Burrows-Abadi-Needham authentication logic [148]; papers by Martín Abadi and Roger Needham, and by Roger Needham and myself on protocol robustness [2, 47]. There is also a survey paper which Roger and I wrote, and which introduced the phrase 'programming Satan's computer' (discussed by Bruce Schneier in the foreword) as a metaphor for security protocol design [48]. In [449] 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 [183, 184] provide leads to current research; and many papers appear in a wide range of conferences.