CHAPTER

4

Protocols

It is impossible to foresee the consequences of being clever.

– CHRISTOPHER STRACHEY

If it's provably secure, it probably isn't.

– LARS KNUDSEN

4.1 Introduction

Passwords are just one example of a more general concept, the security protocol. If security engineering has a core theme, it may be the study of security protocols. They specify the steps that principals use to establish trust relationships. They are where the cryptography and the access controls meet; they are the tools we use to link up human users with remote machines, to synchronise security contexts, and to regulate key applications such as payment. We've come across a few protocols already, including challenge-response authentication and Kerberos. In this chapter, I'll dig down into the details, and give many examples of how protocols fail.

A typical security system consists of a number of principals such as people, companies, phones, computers and card readers, which communicate using a variety of channels including fibre, wifi, the cellular network, bluetooth, 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 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 protocol designs make assumptions about threats. For example, when we get a user to log on by entering a password into a machine, we implicitly assume 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 obvious. Evaluating a protocol thus involves two questions: first, is the threat model realistic? Second, does the protocol deal with it?

Protocols may be very simple, such as swiping a badge through a reader to enter a building. They often involve interaction, and are not necessarily technical. For example, when we order a bottle of fine wine in a restaurant, the standard protocol is that the wine waiter offers us the menu (so that we see the prices but our guests don't); they bring the bottle, so we can check the label, the seal and the temperature; they open it so we can taste it; and then serve it. This has evolved to provide some privacy (our guests don't learn the price), some integrity (we can be sure we got the right bottle and that it wasn't refilled with cheap plonk) and non-repudiation (we can't complain afterwards that the wine was off). Matt Blaze gives other non-technical protocol examples from ticket inspection, aviation security and voting in [261]. Traditional protocols like these often evolved over decades or centuries to meet social expectations as well as technical threats.

At the technical end of things, protocols get a lot more complex, and they don't always get better. As the car industry moved from metal keys to electronic keys with buttons you press, theft fell, since the new keys were harder to copy. But the move to keyless entry has seen car crime rise again, as the bad guys figured out how to build relay devices that would make a key seem closer to the car than it actually was. Another security upgrade that's turned out to be tricky is the move from magnetic-strip cards to smartcards. Europe made this move in the late 2000s while the USA is only catching up in the late 2010s. Fraud against cards issued in Europe actually went up for several years; clones of European cards were used in magnetic-strip cash machines in the USA, as the two systems' protection mechanisms didn't quite mesh. And there was a protocol failure that let a thief use a stolen chipcard in a store even if he didn't know the PIN, which took the banks several years to fix.

So we need to look systematically at security protocols and how they fail.

4.2 Password eavesdropping risks

Passwords and PINs are still the foundation for much of computer security, as the main mechanism used to authenticate humans to machines. We discussed their usability in the last chapter; now let's consider the kinds of technical attack we have to block when designing protocols that operate between one machine and another.

Remote key entry is a good place to start. The early systems, such as the remote control used to open your garage or to unlock cars manufactured up

to the mid-1990's, just broadcast a serial number. The attack that killed them was the 'grabber', a device that would record a code and replay it later. The first grabbers, seemingly from Taiwan, arrived on the market in about 1995; thieves would lurk in parking lots or outside a target's house, record the signal used to lock the car and then replay it once the owner had gone[1].

The first countermeasure was to use separate codes for lock and unlock. But the thief can lurk outside your house and record the unlock code before you drive away in the morning, and then come back at night and help himself. Second, sixteen-bit passwords are too short. Occasionally 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 [309]. And by the mid-1990's, devices appeared that could try all possible codes one after the other. A code will be found on average after about 2^{15} tries, and at ten per second that 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.

The next 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 were still only one or two codes for each car, and grabbers still worked fine.

Using a serial number as a password has a further vulnerability: lots of people 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, and here it's even worse: when a bank buys a burglar alarm, the serial number may appear on the order, the delivery note and the invoice. And banks don't like sending someone out to buy something for cash.

Simple passwords are sometimes the appropriate technology. For example, a monthly season ticket for our local swimming pool simply has a barcode. I'm sure I could make a passable forgery, but as the turnstile attendants get to know the 'regulars', there's no need for anything more expensive. For things that are online, however, static passwords are hazardous; the Mirai botnet got going by recruiting wifi-connected CCTV cameras which had a password that couldn't be changed. And for things people want to steal, like cars, we also need something better. This brings us to cryptographic authentication protocols.

[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. Twelve years after I noted these problems in the second edition of this book, the chip has not been withdrawn. It's now also sold for home security systems and for the remote control of toys.

4.3 Who goes there? – simple authentication

A simple modern authentication device is the token that some multistorey parking garages give subscribers to raise the barrier. The token has a single button; when you press it, it 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 unique to the device, and sent to the garage barrier (typically by radio at 434MHz, though infrared is also used). We will postpone discussion of how to encrypt data to the next chapter, and simply write $\{X\}_K$ for the message X encrypted under the key K.

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

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

This is standard protocol notation, so we'll take it slowly.

The token T sends a message to the garage G consisting of 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. Verification is simple: the garage 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.

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 unique device number) of the token. Another is that once we start discussing attacks on protocols, we may find that a message intended for one principal was intercepted and played back by another. So you might think of the $T \rightarrow G$ to the left of the colon as a hint as to what the protocol designer had in mind.

A *nonce* can be anything that guarantees the freshness of a message. It can be a random number, a counter, a random challenge received from a third party, or even a timestamp. There are subtle differences between them, such as in the level of resistance they offer to various kinds of replay attack, and the ways in which they increase system cost and complexity. In very low-cost systems, random numbers and counters predominate as it's cheaper to communicate 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 just its unique device number encrypted under a global master key KM known to the garage:

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

This is known as *key diversification* or *key derivation*. It's a common way of implementing access tokens, and is widely used in smartcards too. The goal is that someone who compromises a token by drilling into it and extracting the key cannot masquerade as any other token; all he can do is make a copy of one particular subscriber's token. In order to do a complete break of the system, and extract the master key that would enable him to pretend to be any of the system's users, an attacker has to compromise the central server at the garage (which might protect this key in a tamper-resistant smartcard or hardware security module).

But there is still room for error. A common failure mode is for the serial numbers – whether unique device numbers or protocol counters – not to be long enough, so that someone occasionally finds that their remote control works for another car in the car park as well. This can be masked by cryptography. Having 128-bit keys doesn't help if the key is derived by encrypting a 16-bit device number, or by taking a 16-bit key and repeating it eight times. In either case, there are only 2^{16} possible keys, and that's unlikely to be enough even if they appear to be random[2].

Protocol vulnerabilities usually give rise to more, and simpler, attacks than cryptographic weaknesses do. An example comes from the world of prepayment utility meters. Over a million households in the UK, plus over 400 million in developing countries, have an electricity or gas meter that accepts encrypted tokens: the householder buys a magic number and types it into the meter, which then dispenses the purchased quantity of energy. One early meter that was widely used in South Africa checked only that the nonce was different from last time. So the customer could charge their meter indefinitely by buying two low-value power tickets and then feeding them in one after the other; given two valid codes *A* and *B*, the series *ABABAB*... was seen as valid [94].

So the question of whether to use a random number or a counter is not as easy as it looks. If you use random numbers, the lock has to remember a lot of past codes. There's the *valet attack*, where someone with temporary access, such as a valet parking attendant, records some access codes and replays them later to steal your car. In addition, someone might rent a car, record enough unlock codes, and then go back later to the rental lot to steal it. Providing enough nonvolatile memory to remember thousands of old codes might add a few cents to the cost of your lock.

If you opt for counters, the problem is synchronization. The key might be used for more than one lock; it may also be activated repeatedly by accident (I once took an experimental token home where it was gnawed by my dogs). So you need a way to recover after the counter has been incremented hundreds or possibly even thousands of times. One common product uses a sixteen bit

[2]We'll go into this in more detail in section 5.3.1.2 where we discuss the birthday theorem in probability theory.

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. In your application, would you be worried about that?

So designing even a simple token authentication mechanism is not as easy as it looks, and if you assume that your product will only attract low-grade adversaries, this assumption might fail over time. An example 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. All sorts of other industries are getting in on the act, from scientific instruments to games consoles. The cryptographic mechanisms used to support this started off in the 1990s being fairly rudimentary, as vendors thought that any competitor who circumvented them on an industrial scale could be sued or even jailed under copyright law. But then a judge found that while a vendor had the right to hire the best cryptographer they could find to lock their customers in, a competitor also had the right to hire the best cryptanalyst they could find to set them free to buy accessories from elsewhere. This set off a serious arms race, which we'll discuss in section 24.6. Here I'll just remark that security isn't always a good thing. Security mechanisms are used to support many business models, where they're typically stopping the device's owner doing things she wants to rather than protecting her from the bad guys. The effect may be contrary to public policy; one example is cellphone locking, which results in hundreds of millions of handsets ending up in landfills each year, with toxic heavy metals as well as the embedded carbon cost.

4.3.1 Challenge and response

Since 1995, all cars sold in Europe were required to have a 'cryptographically enabled immobiliser' and by 2010, most cars had remote-controlled door unlocking too, though most also have a fallback metal key so you can still get into your car even if the key fob battery is flat. The engine immobiliser is harder to bypass using physical means and uses a two-pass *challenge-response protocol* to 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; this is often done by a separate RFID chip that's powered by the incoming radio signal and so keeps on working even if the battery is flat. The frequency is low (125kHz) so the car can power the transponder directly, and the exchange is also relatively immune to a noisy RF environment.

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:

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

This is sound in theory, but implementations of security mechanisms often fail the first two or three times people try it.

Between 2005 and 2015, all the main remote key entry and immobiliser systems were broken, whether by security researchers, car thieves or both. The attacks involved a combination of protocol errors, poor key management, weak ciphers, and short keys mandated by export control laws.

The first to fall was TI's DST transponder chip, which was used by at least two large car makers and was also the basis of the SpeedPass toll payment system. Stephen Bono and colleagues found in 2005 that it used a block cipher with a 40-bit key, which could be calculated by brute force from just two responses [298]. This was one side-effect of US cryptography export controls, which I discuss in 26.2.7.1. From 2010, Ford, Toyota and Hyundai adopted a successor product, the DST80. The DST80 was broken in turn in 2020 by Lennert Wouters and colleagues, who found that as well as side-channel attacks on the chip, there are serious implementation problems with key management: Hyundai keys have only 24 bits of entropy, while Toyota keys are derived from the device serial number that an attacker can read (Tesla was also vulnerable but unlike the older firms it could fix the problem with a software upgrade) [2050]. Next was Keeloq, which was used for garage door openers as well as by some car makers; in 2007, Eli Biham and others found that given an hour's access to a token they could collect enough data to recover the key [244]. Worse, in some types of car, there is also a protocol bug, in that the key diversification used exclusive-or: $KT = T \bigoplus KM$. So you can rent a car of the type you want to steal and work out the key for any other car of that type.

Also in 2007, someone published the Philips Hitag 2 cipher, which also had a 48-bit secret key. But this cipher is also weak, and as it was attacked by various cryptanalysts, the time needed to extract a key fell from days to hours to minutes. By 2016, attacks took 8 authentication attempts and a minute of computation on a laptop; they worked against cars from all the French and Italian makers, along with Nissan, Mitsubishi and Chevrolet [748].

The last to fall was the Megamos Crypto transponder, used by Volkswagen and others. Car locksmithing tools appeared on the market from 2008, which included the Megamos cipher and were reverse engineered by researchers from Birmingham and Nijmegen – Roel Verdult, Flavio Garcia and Barış Ege – who cracked it [1956]. Although it has a 96-bit secret key, the effective key length is only 49 bits, about the same as Hitag 2. Volkswagen got an injunction in the High Court in London to stop them presenting their work at Usenix 2013, claiming that their trade secrets had been violated. The researchers resisted, arguing that the locksmithing tool supplier had extracted the secrets. After two years of argument, the case settled without admission of liability on either side. Closer study then threw up a number of further problems. There's also a protocol attack as an adversary can rewrite each 16-bit word of the 96-bit key, one after another, and search for the key 16 bits at a time; this reduces the time needed for an attack from days to minutes [1957].

Key management was pervasively bad. A number of Volkswagen implementations did not diversify keys across cars and transponders, but used a fixed global master key for millions of cars at a time. Up till 2009, this used a cipher called AUT64 to generate device keys; thereafter they moved to a stronger cipher called XTEA but kept on using global master keys, which were found in 23 models from the Volkswagen-Audi group up till 2016 [748][3].

It's easy to find out if a car is vulnerable: just try to buy a spare key. If the locksmith companies have figured out how to duplicate the key, your local garage will sell you a spare for a few bucks. We have a spare key for my wife's 2005 Lexus, bought by the previous owner. But when we lost one of the keys for my 2012 Mercedes, we had to go to a main dealer, pay over £200, show my passport and the car log book, have the mechanic photograph the vehicle identification number on the chassis, send it all off to Mercedes and wait for a week. We saw in Chapter 3 that the hard part of designing a password system was recovering from compromise without the recovery mechanism itself becoming either a vulnerability or a nuisance. Exactly the same applies here!

But the worst was still to come: passive keyless entry systems (PKES). Challenge-response seemed so good that car vendors started using it with just a push button on the dashboard to start the car, rather than with a metal key. Then they increased the radio frequency to extend the range, so that it worked not just for short-range authentication once the driver was sitting in the car, but as a keyless entry mechanism. The marketing pitch was that so long as

[3]There are some applications where universal master keys are inevitable, such as in communicating with a heart pacemaker – where a cardiologist may need to tweak the pacemaker of any patient who walks in, regardless of where it was first fitted, and regardless of whether the network's up – so the vendor puts the same key in all its equipment. Another example is the subscriber smartcard in a satellite-TV set-top box, which we'll discuss later. But they often result in a break-once-run-anywhere (BORA) attack. To install universal master keys in valuable assets like cars in a way that facilitated theft and without even using proper tamper-resistant chips to protect them was an egregious error.

you keep the key in your pocket or handbag you don't have to worry about it; the car will unlock when you walk up to it, lock as you walk away, and start automatically when you touch the controls. What's not to like?

Well, now you don't have to press a button to unlock your car, it's easy for thieves to use devices that amplify or relay the signals. The thief sneaks up to your front door with one relay while leaving the other next to your car. If you left your keys on the table in the hall, the car door opens and away he goes. Even if the car is immobilised he can still steal your stuff. And after many years of falling car thefts, the statistics surged in 2017 with 56% more vehicles stolen in the UK, followed by a further 9% in 2018 [824][4].

The takeaway message is that the attempt since about 1990 to use cryptography to make cars harder to steal had some initial success, as immobilisers made cars harder to steal and insurance premiums fell. It has since backfired, as the politicians and then the marketing people got in the way. The politicians said it would be disastrous for law enforcement if people were allowed to use cryptography they couldn't crack, even for stopping car theft. Then the immobiliser vendors' marketing people wanted proprietary algorithms to lock in the car companies, whose own marketing people wanted passive keyless entry as it seemed cool.

What can we do? Well, at least two car makers have put an accelerometer in the key fob, so it won't work unless the key is moving. One of our friends left her key on the car seat while carrying her child indoors, and got locked out. The local police advise us to use old-fashioned metal steering-wheel locks; our residents' association recommends keeping keys in a biscuit tin. As for me, we bought such a car but found that the keyless entry was simply too flaky; my wife got stranded in a supermarket car park when it just wouldn't work at all. So we took that car back, and got a second-hand one with a proper push-button remote lock. There are now chips using AES from NXP, Atmel and TI – of which the Atmel is open source with an open protocol stack.

However crypto by itself can't fix relay attacks; the proper fix is a new radio protocol based on ultrawideband (UWB) with intrinsic ranging, which measures the distance from the key fob to the car with a precision of 10cm up to a range of 150m. This is fairly complex to do properly, and the design of the new 802.15.4z Enhanced Impulse Radio is described by Srdjan Capkun and colleagues [1768]; the first chip became available in 2019, and it will ship in cars from 2020. Such chips have the potential to replace both the Bluetooth and NFC protocols, but they might not all be compatible; there's a low-rate pulse (LRP) mode that has an open design, and a high-rate pulse (HRP) variant that's partly proprietary. Were I advising a car startup, LRP would be my starting point.

[4]To be fair this was not due solely to relay attacks, as about half of the high-value thefts seem to involve connecting a car theft kit to the onboard diagnostic port under the glove box. As it happens, the authentication protocols used on the CAN bus inside the vehicle are also vulnerable in a number of ways [893]. Updating these protocols will take many years because of the huge industry investment.

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 [715]. 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's much better than sending a password in the clear, but like keyless entry it suffers from middleperson attacks (the beneficiaries seem to be mostly intelligence agencies).

4.3.2 Two-factor authentication

The most visible use of challenge-response is probably in *two-factor authentication*. Many organizations issue their staff with password generators to let them log on to corporate computer systems, and many banks give similar devices to customers. They may look like little calculators (and some even work as such) but their main function is as follows. When you want to log in, you are presented with a random nonce 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 4.1. 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 you get in.

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

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

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 new access codes by encrypting a counter or a clock. And they work; the US Defense Department announced in 2007 that an authentication system based on the DoD Common Access Card had cut network intrusions by 46% in the previous year [321].

This was just when crooks started phishing bank customers at scale, so many banks adopted the technology. One of my banks gives me a small calculator that generates a new code for each logon, and also allows me to authenticate new payees by using the last four digits of their account number in place of the challenge. My other bank uses the Chip Authentication Program (CAP), a calculator in which I can insert my bank card to do the crypto.

Figure 4.1: Password generator use

But this still isn't foolproof. In the second edition of this book, I noted 'someone who takes your bank card from you at knifepoint can now verify that you've told them the right PIN', and this now happens. I also noted 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' and this has also become widespread. To see how such attacks work, let's look at a military example.

4.3.3 The MIG-in-the-middle attack

The first use of challenge-response authentication protocols was probably in the military, with 'identify-friend-or-foe' (IFF) systems. 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 risk of pilots shooting down their colleagues by mistake and drove the development of automatic systems to prevent this. These were first fielded in World War II, and 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 US Air Force started a research program to incorporate cryptographic protection in the system. Nowadays, the typical air defense system sends random challenges with its radar signals, and friendly aircraft can identify themselves with correct responses.

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 since become part of the folklore, and it nicely illustrates how attacks can be carried out in real time on challenge-response protocols.

In the late 1980's, South African troops were fighting a war in northern Namibia and southern Angola. Their 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 to the MIGs, who retransmitted them and were allowed through – as in Figure 4.2. According to my informant, this shocked 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.

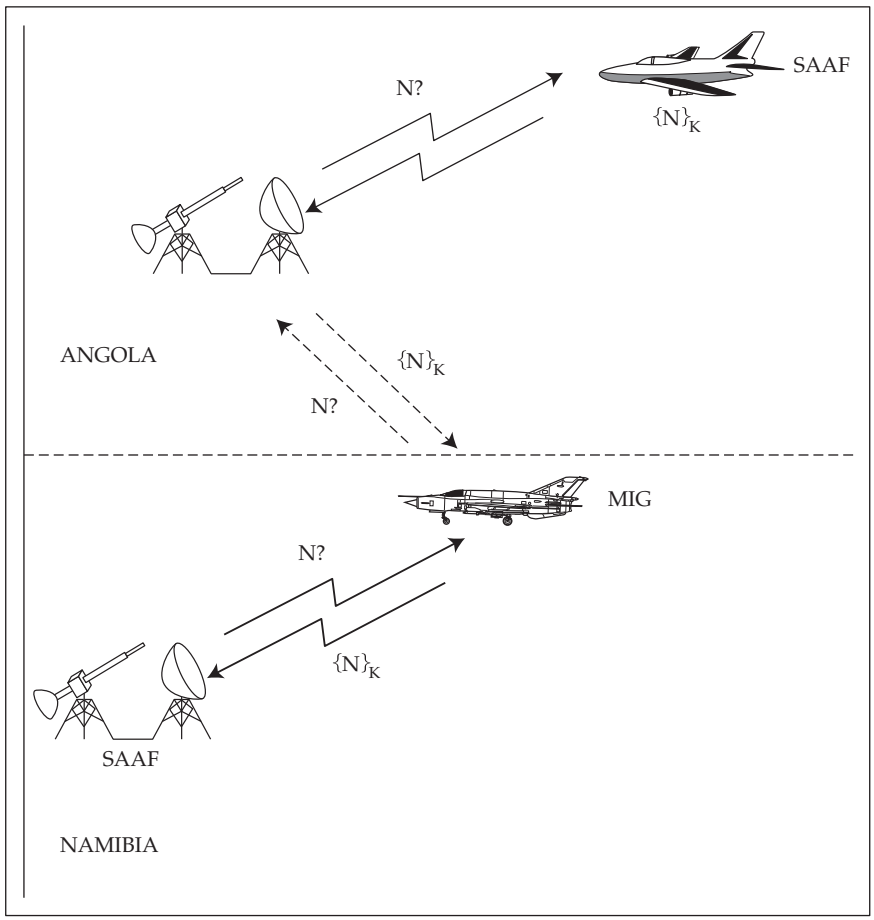

Figure 4.2: The MIG-in-the middle attack

In any case, the tale gives us another illustration of the man-in-the-middle attack. The relay attack against cars is another example. It also works against 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 the response to the bank, and the bank now accepts the phisherman as the mark.

Stopping a middleperson attack is harder than it looks, and may involve multiple layers of defence. Banks typically look for a known machine, a password, a second factor such as an authentication code from a CAP reader, and a risk assessment of the transaction. For high-risk transactions, such as adding a new payee to an account, both my banks demand that I compute an authentication code on the payee account number. But they only authenticate the last four

digits, because of 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 get annoyed and bank elsewhere. Also, the bad guys may be able to exploit any fallback mechanisms, perhaps by spoofing customers into calling phone numbers that run a middleperson attack between the customer and the call center. I'll discuss all this further in the chapter on Banking and Bookkeeping.

We will come across such attacks again and again in applications ranging from Internet security protocols to Bluetooth. They even apply in 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!

4.3.4 Reflection attacks

Further interesting problems arise when two principals have to identify each other. Suppose that a 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 when a fighter challenges an enemy bomber, the bomber might just reflect the challenge back to the fighter's wingman, get a correct response, and then send that back as its own response:

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

There are a number of ways of stopping this, such as including the names of the two parties in the exchange. In the above 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 $\{F', N\}$ from the wingman F' could be detected[5].

This serves to illustrate the subtlety of the trust assumptions that underlie authentication. 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.

[5] And don't forget: you also have to check that the intruder didn't just reflect your own challenge back at you. You must be able to remember or recognise your own messages!

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 careful analysis of trust assumptions and their consequences is at the heart of security protocol design.

By now you might think that we understand all the protocol design aspects of IFF. But we've omitted one of the most important problems – and one which the designers of early IFF systems didn't anticipate. As radar is passive the returns are weak, while IFF is active and so the signal from an IFF transmitter will usually be audible at a much greater range than the same aircraft's radar 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 more 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 made difficult in mode XII using directionality and time-of-flight.

Other IFF design problems include 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. I'll return to IFF in Chapter 23. 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) might not help. 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.

4.4 Manipulating the message

We've now seen a number of middleperson attacks that reflect or spoof the information used to authenticate a participant. However, there are more complex attacks where the attacker doesn't just impersonate someone, but manipulates the message content.

One example we saw already is the prepayment meter that remembers only the last ticket it saw, so it can be recharged without limit by copying in the codes from two tickets *A* and *B* one after another: *ABABAB*.... Another 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 can insert extra pulses, making the taxi appear to have gone further. A truck driver who wants to drive faster or further than regulations allow can use a similar device to discard some pulses, so he seems to have been driving more slowly or not at all. We'll discuss such attacks in the chapter on 'Monitoring Systems', in section 14.3.

As well as monitoring systems, control systems often need to be hardened against message-manipulation attacks. The Intelsat satellites used for international telephone and data traffic have mechanisms to prevent a command being accepted twice – otherwise an attacker could replay control traffic and repeatedly order the same maneuver to be carried out until the satellite ran out of fuel [1529]. We will see lots of examples of protocol attacks involving message manipulation in later chapters on specific applications.

4.5 Changing the environment

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

A nice example comes from the world of cash machine fraud. In 1993, 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 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 [55]. Exactly the same fraud happened in the UK after the move to 'chip and PIN' smartcards in the mid-2000s; a gang wiretapped perhaps 200 filling stations, collected card data from the wire, observed the PINs using CCTV cameras, then made up thousands of magnetic-strip clone cards that were used in countries whose ATMs still used magnetic strip technology. At our local filling station, over 200 customers suddenly found that their cards had been used in ATMs in Thailand.

Why had the system been designed so badly, and why did the design error persist for over a decade through a major technology change? 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 [1303]. (The analogy used was that the magnetic strip was your name and the PIN your password.) 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 destroyed the assumptions behind the original system architecture. Instead of putting a card whose magnetic strip contained no security data into a trusted machine, people were putting a card with clear security data 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.

4.6 Chosen protocol attacks

Governments keen to push ID cards 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?

Some firms are pushing multifunction authentication devices that could be used in a wide range of transactions to save you having to carry around dozens of different cards and keys. A more realistic view of the future may be that people's phones will be used for most private-sector authentication functions.

But this too may not be as simple as it looks. 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 the authentication of bank transactions?

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

smartphones are used to authenticate everything, it would be natural for the porn site to ask the customer to authenticate a random challenge as proof of age. A porn site might then mount a 'Mafia-in-the-middle' attack as shown in Figure 4.3. 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 [1034].

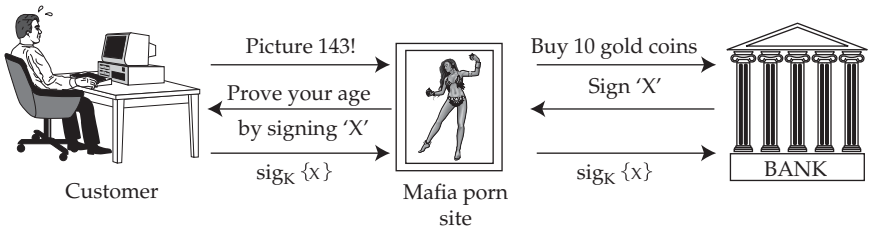

Figure 4.3: The Mafia-in-the-middle attack

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 [1034]. This is why, if we're going to use our phones to authenticate everything, it will be really important to keep the banking apps and the porn apps separate. That will be the subject in Chapter 6 on Access Control.

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. The classic case is where a bank relies for two-factor authentication on sending SMSes to customers as authentication codes. As I discussed in section 3.4.1, the bad guys have learned to attack that system by SIM-swap fraud – pretending to the phone company that they're the target, claiming to have lost their phone, and getting a replacement SIM card.

4.7 Managing encryption keys

The examples of security protocols that we've 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.

4.7.1 The resurrecting duckling

In the Internet of Things, keys can sometimes be managed directly and physically, by local setup and a policy of *trust-on-first-use* or TOFU.

Vehicles provided an early example. I mentioned above that crooked taxi drivers used to put interruptors in the cable from their car's gearbox sensor to the taximeter, to add additional mileage. The same problem happened in reverse with tachographs, the devices used by trucks to monitor drivers' hours and speed. When tachographs went digital in the late 1990s, we decided to encrypt the pulse train from the sensor. But how could keys be managed? The solution was that whenever a new tachograph is powered up after a factory reset, it trusts the first crypto key it receives over the sensor cable. I'll discuss this further in section 14.3.

A second example is Homeplug AV, the standard used to encrypt data communications over domestic power lines, and widely used in LAN extenders. In the default, 'just-works' mode, a new Homeplug device trusts the first key it sees; and if your new wifi extender mates with the neighbour's wifi instead, you just press the reset button and try again. There is also a 'secure mode' where you open a browser to the network management node and manually enter a crypto key printed on the device packaging, but when we designed the Homeplug protocol we realised that most people have no reason to bother with that [1439].

The TOFU approach is also known as the 'resurrecting duckling' after an analysis that Frank Stajano and I did in the context of pairing medical devices [1822]. The idea is that when a baby duckling hatches, it imprints on the first thing it sees that moves and quacks, even if this is the farmer – who can end up being followed everywhere by a duck that thinks he's mummy. If such false imprinting happens with an electronic device, you need a way to kill it and resurrect it into a newborn state – which the reset button does in a device such as a LAN extender.

4.7.2 Remote key management

The more common, and interesting, case is the management of keys in remote devices. The basic technology was developed from the late 1970s to manage keys in distributed computer systems, with cash machines being an early application. In this section we'll discuss shared-key protocols such as Kerberos, leaving public-key protocols such as TLS and SSH until after we've discussed public-key cryptology in Chapter 5.

The basic idea behind key-distribution protocols is that where two principals want to communicate, they may use a trusted third party to introduce them. It's customary to give them human names in order to avoid getting lost in too much algebra. So we will call the two communicating principals 'Alice' and 'Bob', and the trusted third party 'Sam'. Alice, Bob and Sam are likely to be programs running on different devices. (For example, in a protocol to let a car

dealer mate a replacement key with a car, Alice might be the car, Bob the key and Sam the car maker.)

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.

We've seen that replay attacks are a known problem, 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

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

Expanding the notation, Alice calls Sam and says she'd like to talk to Bob. Sam makes up a message consisting of Alice's name, Bob's name, a session 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 session 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 session key.

4.7.3 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 [1428]. It is somewhat similar to the above, but uses nonces rather than timestamps. It runs as follows:

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

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. And if Alice ever got fired, then Sam had better have a list of everyone in the firm to whom he issued a key for communicating with her, to tell them not to believe it any more. 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 40 years later, this example is still controversial. 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 some sympathy) is that this is a protocol failure brought on by shifting assumptions [781, 1493]. 1978 was a kinder, gentler world; computer security then concerned itself with keeping 'bad guys' out, while nowadays we expect the 'enemy' to be among the users of our system. The Needham-Schroeder paper assumed that all principals behave themselves, and that all attacks came from outsiders [1428]. Under those assumptions, the protocol remains sound.

4.7.4 Kerberos

The most important practical derivative of the Needham-Schroeder protocol is Kerberos, a distributed access control system that originated at MIT and is now one of the standard network authentication tools [1829]. It has become part of the basic mechanics of authentication for both Windows and Linux, particularly when machines share resources over a local area network. Instead of a single trusted third party, Kerberos has two kinds: authentication servers to which users log on, and ticket granting servers which give them tickets allowing access to various resources such as files. This enables scalable access management. In a university, for example, one might manage students through their colleges or 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:

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

Translating this into English: Alice asks the ticket granting server for access to B. If this is permissible, the ticket $\{T_S, L, K_{AB}, A\}_{K_{BS}}$ 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 revocation issue with the Needham-Schroeder protocol 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 sync; they might even be desynchronized deliberately as part of a more complex attack.

What's more, Kerberos is a *trusted third-party* (TTP) protocol in that S is trusted: if the police turn up with a warrant, they can get Sam to turn over the keys and read the traffic. Protocols with this feature were favoured during the 'crypto wars' of the 1990s, as I will discuss in section 26.2.7. Protocols that involve no or less trust in a third party generally use public-key cryptography, which I describe in the next chapter.

A rather similar protocol to Kerberos is OAuth, a mechanism to allow secure delegation. For example, if you log into Doodle using Google and allow Doodle to update your Google calendar, Doodle's website redirects you to Google, which gets you to log in (or relies on a master cookie from a previous login) and asks you for consent for Doodle to write to your calendar. Doodle then gives you an access token for the calendar service [864]. I mentioned in section 3.4.9.3 that this poses a cross-site phishing risk. OAuth was not designed for user authentication, and access tokens are not strongly bound to clients. It's a complex framework within which delegation mechanisms can be built, with both short-term and long-term access tokens; the details are tied up with how cookies and web redirects operate and optimised to enable servers to be stateless, so they scale well for modern web services. In the example above, you want to

be able to revoke Doodle's access at Google, so behind the scenes Doodle only gets short-lived access tokens. Because of this complexity, the OpenID Connect protocol is a 'profile' of OAuth which ties down the details for the case where the only service required is authentication. OpenID Connect is what you use when you log into your newspaper using your Google or Facebook account.

4.7.5 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. But there may be encrypted passwords for tens of thousands of staff and keys for large numbers of devices too. 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. You need to 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 [1410]. And things go wrong as applications evolve; it's important to provide headroom to support next year's functionality. It's also important to support recovery from security failure. Yet there are no standard ways of doing either.

Public-key cryptography, which I'll discuss in Chapter 5, can simplify the key-management task slightly. In banking the usual answer is to use dedicated cryptographic processors called hardware security modules, which I'll describe in detail later. Both of these introduce further complexities though, and even more subtle ways of getting things wrong.

4.8 Design assurance

Subtle difficulties of the kind we have seen above, and the many ways in which protection properties depend on subtle assumptions that may be misunderstood, have led researchers to apply formal methods to 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. We often find that the process helps clarify the assumptions that underlie a given protocol.

There are several different approaches to verifying the correctness of protocols. One of the best known is the *logic of belief*, or *BAN logic*, named after its inventors Burrows, Abadi and Needham [352]. It reasons about what a principal might reasonably believe having seen certain messages, timestamps and so

on. Other 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; I described an example in Chapter 3 of the second edition, of how the BAN logic was used to verify a bank card used for stored-value payments. That's still used in Germany as the 'Geldkarte' but elsewhere its use has died out (it was Net1 in South Africa, Proton in Belgium, Moneo in France and a VISA product called COPAC). I've therefore decided to drop the gory details from this edition; the second edition is free online, so you can download and read the details.

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. But they have their limitations, too.

We often find bugs in verified protocols; they're just not in the part that we verified. For example, Larry Paulson verified the SSL/TLS protocol using his Isabelle theorem prover in 1998, and about one security bug has been found every year since then. These have not been flaws in the basic design but exploited additional features that had been added later, and implementation issues such as timing attacks, which we'll discuss later. In this case there was no failure of the formal method; that simply told the attackers where they needn't bother looking.

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. It should not be possible to interpret data in more than one way; so the message formats need to make clear what's a name, what's an address, what's a timestamp, and so on; string formats have to be unambiguous and it should be impossible to use the protocol itself to mount attacks on the software that handles it, such as by buffer overflows. There are other issues concerning the freshness provided by counters, timestamps and random challenges, and on the way encryption is used. If the protocol uses public key cryptography or digital signature mechanisms, there are more subtle attacks and further robustness issues, which we'll start to tackle in the next chapter. To whet your appetite, randomness in protocol often helps robustness at other layers, since it makes it harder to do a whole range of attacks – from those based on mathematical cryptanalysis through those that exploit side-channels such as power consumption and timing to physical attacks that involve microprobes or lasers.

4.9 Summary

Passwords are just one example of a more general concept, the security protocol. Protocols specify the steps that principals use to establish trust relationships in a system, such as authenticating a claim to identity, demonstrating ownership of a credential, or establishing a claim on a resource. Cryptographic authentication protocols are used for a wide range of purposes, from basic entity authentication to providing 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.

Protocols are surprisingly difficult to get right. They can suffer from a number of problems, including middleperson attacks, modification attacks, reflection attacks, and replay attacks. These threats can interact with implementation vulnerabilities and poor cryptography. 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 relevant. The upshot is that attacks are still found frequently on protocols that we've been using for years, and sometimes even on protocols for which we thought we had a security proof. Failures have real consequences, including the rise in car crime worldwide since car makers started adopting passive keyless entry systems without stopping to think about relay attacks. Please don't design your own protocols; get a specialist to help, and ensure that your design is published for thorough peer review by the research community. Even specialists get the first versions of a protocol wrong (I have, more than once). It's a lot cheaper to fix the bugs before the protocol is actually deployed, both in terms of cash and in terms of reputation.

Research problems

At several times during the past 30 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 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. 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.

Much protocol research is problem-driven, but there are still deep questions. How much can we get out of formal methods, for example? And how do we manage the tension between the principle that robust protocols are generally those in which everything is completely specified and checked 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. For the historical background you might read the original Needham-Schroeder paper [1428], the Burrows-Abadi-Needham authentication logic [352], papers on protocol robustness [2, 113] and a survey paper by Anderson and Needham [114]. Beyond that, there are many papers scattered around a wide range of conferences; you might also start by studying the protocols used in a specific application area, such as payments, which we cover in more detail in Part 2. As for remote key entry and other security issues around cars, a good starting point is a tech report by Charlie Miller and Chris Valasek on how to hack a Jeep Cherokee [1318].