**Chapter 7**

**Distributed Systems**

**A distributed system is one in which the failure of a computer you**  
 **didn’t even know existed can render your own computer unusable.**

– LESLIE LAMPORT [1123]

**What’s in a name? That which we call a rose**

**by any other name would smell as sweet**

– WILLIAM SHAKESPEARE

**7.1** **Introduction**

We need a lot more than authentication, access control and cryptography to  
 build a robust distributed system of any size. Some things need to happen

quickly, or in the right order, and matters that are trivial to deal with for a few  
 machines become a big deal once we have hyperscale data centres with complex  
 arrangements for resilience. Everyone must have noticed that when you update  
 your address book with an online service provider, the update might appear a  
 second later on another device, or perhaps only hours later.

Over the last 50 years, we’ve learned a lot about issues such as concurrency,

failure recovery and naming as we’ve built things ranging from phone systems  
 and payment networks to the Internet itself. We have solid theory, and a lot  
 of hard-won experience. These issues are central to the design of robust secure  
 systems but are often handled rather badly. I’ve already described attacks on  
 protocols that arise as concurrency failures. If we replicate data to make a

system fault-tolerant, then we may increase the risk of data theft. Finally,

naming can be a thorny problem. There are complex interactions of people

and objects with accounts, sessions, documents, ﬁles, pointers, keys and other  
 ways of naming stuff. Many organisations are trying to build larger, ﬂatter

namespaces – whether using identity cards to track citizens or using device ID  
 to track objects – but there are limits to what we can practically do. Big data  
 means dealing with lots of identiﬁers, many of which are ambiguous or even  
 changing, and a lot of things can go wrong.

236

*7.2. CONCURRENCY*

**7.2** **Concurrency**

Processes are called *concurrent* if they can run at the same time, and this is  
 essential for performance; modern computers have many cores and run many  
 programs at a time, typically for many users. However, concurrency is hard to  
 do robustly, especially when processes can act on the same data. Processes may  
 use old data; they can make inconsistent updates; the order of updates may  
 or may not matter; the system might deadlock; the data in different systems  
 might never converge to consistent values; and when it’s important to make  
 things happen in the right order, or even to know the exact time, this can be  
 trickier than you might think. These issues go up and down the entire stack.

Systems are becoming ever more concurrent for a number of reasons. First

is scale: Google may have started off with four machines but their ﬂeet passed  
 a million in 2011. Second is device complexity; a luxury car can now contain  
 dozens to hundreds of different processors. The same holds for your laptop

and your mobile phone. Deep within each CPU, instructions are executed in  
 parallel, and this complexity leads to the Spectre attacks we discussed in the  
 chapter on access control. On top of this, virtualization technologies such as Xen  
 are the platforms on which modern cloud services are built, and they may turn  
 a handful of real CPUs in a server into hundreds or even thousands of virtual  
 CPUs. Then there’s interaction complexity: going up to the application layer,  
 an everyday transaction such as booking a rental car may call other systems to  
 check your credit card, your credit reference agency score, your insurance claim  
 history and much else, while these systems in turn may depend on others.

Programming concurrent systems is hard, and the standard textbook exam-

ples come from the worlds of operating system internals and of performance  
 measurement. Computer scientists are taught Amdahl’s law: if the proportion  
 that can be parallelised is *p* and *s* is the speedup from the extra resources, the  
 overall speedup is (1*ffip*+*p/s*)*ffi*1. Thus if three-quarters of your program can be  
 you can get is four times; and if you throw eight cores at it, the practical speedup  
 is not quite three times1. But concurrency control in the real world is also a  
 security issue. Like access control, it is needed to prevent users interfering with  
 each other, whether accidentally or on purpose. And concurrency problems can  
 occur at many levels in a system, from the hardware right up to the business  
 logic. In what follows, I provide a number of concrete examples; they are by no  
 means exhaustive.

**7.2.1** **Using old data versus paying to propagate state**

I’ve already described two kinds of concurrency problem: replay attacks on  
 protocols, where an attacker manages to pass off out-of-date credentials; and  
 race conditions, where two programs can race to update some security state.  
 As an example, I mentioned the ‘mkdir’ vulnerability from Unix, in which a  
 privileged instruction that is executed in two phases could be attacked halfway  
 through by renaming the object on which it acts. Another example goes back to

|  |  |
| --- | --- |
| 1(1 *�* ~~3~~4 + 4*.*8 )*�*1 =(0*.*25 + 0*.*09375)*�*1=(0*.*34375)*�*1=2.909  **Security Engineering** 237 | Ross Anderson |

*7.2. CONCURRENCY*

the 1960s, where in one of the ﬁrst multiuser operating systems, IBM’s OS/360,  
 an attempt to open a ﬁle caused it to be read and its permissions checked; if  
 the user was authorised to access it, it was read again. The user could arrange  
 things so that the ﬁle was altered in between [1129].

These are examples of a *time-of-check-to-time-of-use* (TOCTTOU) attack.

We have systematic ways of ﬁnding such attacks in ﬁle systems [251], but attacks  
 still crop up both at lower levels, such as system calls in virtualised environ-  
 ments, and at higher levels such as business logic. Preventing them isn’t always  
 economical, as propagating changes in security state can be expensive.

A good case study is card fraud. Since credit and debit cards became popular

in the 1970s, the banking industry has had to manage lists of *hot* cards (whether  
 stolen or abused), and the problem got steadily worse in the 1980s as card  
 networks went international. It isn’t possible to keep a complete hot card list in  
 every merchant terminal, as we’d have to broadcast all loss reports instantly to  
 tens of millions of devices, and even if we tried to verify all transactions with the  
 bank that issued the card, we’d be unable to use cards in places with no network  
 (such as in remote villages and on airplanes) and we’d impose unacceptable costs  
 and delays elsewhere. Instead, there are multiple levels of stand-in processing,  
 exploiting the fact that most payments are local, or low-value, or both.

Merchant terminals are allowed to process transactions up to a certain limit

(the *ﬂoor limit*) offline; larger transactions need online veriﬁcation with the  
 merchant’s bank, which will know about all the local hot cards plus foreign cards  
 that are being actively abused; above another limit it might refer the transaction  
 to a network such as VISA with a reasonably up-to-date international list; while  
 the largest transactions need a reference to the card-issuing bank. In effect, the  
 only transactions that are checked immediately before use are those that are  
 local or large.

Experience then taught that a more centralised approach can work better

for bad terminals. About half the world’s ATM transactions use a service that  
 gets alerts from subscribing banks when someone tries to use a stolen card at an  
 ATM, or guesses the PIN wrong. FICO observed that criminals take a handful  
 of stolen cards to a cash machine and try them out one by one; they maintain a  
 list of the 40 ATMs worldwide that have been used most recently for attempted  
 fraud, and banks that subscribe to their service decline all transactions at those  
 machines – which become unusable by those banks’ cards for maybe half an  
 hour. Most thieves don’t understand this and just throw them away.

Until about 2010, payment card networks had the largest systems that man-

age the global propagation of security state, and their experience taught us that  
 revoking compromised credentials quickly and on a global scale is expensive.  
 The lesson was learned elsewhere too; the US Department of Defense, for ex-  
 ample, issued 16 million certiﬁcates to military personnel during 1999–2005, by  
 which time it had to download 10 million revoked certiﬁcates to all security  
 servers every day, and some systems took half an hour to do this when they  
 were ﬁred up [1299].

The costs of propagating security state can lead to centralisation. Big service

ﬁrms such as Google, Facebook and Microsoft have to maintain credentials  
 for billions of users anyway, so they offer logon as a service to other websites.

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 238 | Ross Anderson |

*7.2. CONCURRENCY*

Other ﬁrms, such as certiﬁcation authorities, also provide online credentials.  
 But although centralisation can cut costs, a compromise of the central service  
 can be disruptive. In 2011, for example, hackers operating from Iranian IP

addresses compromised the Dutch certiﬁcation authority Diginotar. On July  
 9th, they generated fake certiﬁcates and did middleperson attacks on the gmail  
 of Iranian activists. Diginotar noticed on the 19th that certiﬁcates had been  
 wrongly issued but merely called in its auditors. The hack became public on the  
 29th, and Google reacted by removing all Diginotar certiﬁcates from Chrome  
 on September 3rd, and getting Mozilla to do likewise. This led immediately to  
 the failure of the company, and Dutch public services were unavailable online  
 for many days as ministries scrambled to get certiﬁcates for their web services  
 from other suppliers [471].

**7.2.2** **Locking to prevent inconsistent updates**

When people work concurrently on a document, they may use a version control  
 system to ensure that only one person has write access at any one time to any  
 given part of it, or at least to warn of contention and ﬂag up any inconsistent  
 edits. *Locking* is one general way to manage contention for resources such as  
 ﬁlesystems and to make conﬂicting updates less likely. Another approach is

*callback*; a server may keep a list of all those clients which rely on it for security  
 state and notify them when the state changes.

Credit cards again provide an example of how this applies to security. If I

own a hotel and a customer presents a credit card on check-in, I ask the card  
 company for a *pre-authorisation*, which records that I will want to make a debit  
 in the near future; I might register a claim on ‘up to $500’. This is implemented  
 by separating the authorisation and settlement systems. Handling the failure  
 modes can be tricky. If the card is cancelled the following day, my bank can  
 call me and ask me to contact the police, or to get her to pay cash2. This is an  
 example of the *publish-register-notify* model of how to do robust authorisation  
 in distributed systems (of which there’s a more general description in [152]).

Callback mechanisms don’t provide a universal solution, though. The cre-

dential issuer might not want to run a callback service, and the customer might  
 object on privacy grounds to the issuer being told all her comings and goings.  
 Consider passports as another example. In many countries, government ID is  
 required for many transactions, but governments won’t provide any guarantee,  
 and most citizens would object if the government kept a record of every time  
 an ID document was presented. Indeed, one of the frequent objections to the  
 Indian government’s requirement that the Aadhar biometric ID system be used  
 in more and more transactions is that checking citizens’ ﬁngerprints or iris codes  
 at all signiﬁcant transactions creates an audit trail of all the places where they  
 have done business, which is available to officials and to anyone who cares to  
 bribe them.

2My bank might or might not have guaranteed me the money; it all depends on what sort

of contract I’ve got with it. There were also attacks for a while when crooks ﬁgured out how to  
 impersonate a store and cancel an authorisation so that a card could be used to make multiple  
 big purchases. And it might take a day or three for the card-issuing bank to propagate an  
 alarm to the merchant’s bank. A deep dive into all this would be a book chapter in itself!

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 239 | Ross Anderson |

*7.2. CONCURRENCY*

There is a general distinction between those credentials whose use gives rise

to some obligation on the issuer, such as credit cards, and the others, such  
 as passports. Among the differences is whether the credential’s use changes

important state, beyond possibly adding to a log ﬁle or other surveillance system.  
 This is linked with whether the order in which updates are made is important.

**7.2.3** **The order of updates**

If two transactions arrive at the government’s bank account – say a credit of  
 $500,000 and a debit of $400,000 – then the order in which they are applied may  
 not matter much. But if they’re arriving at my bank account, the order will  
 have a huge effect on the outcome! In fact, the problem of deciding the order in  
 which transactions are applied has no clean solution. It’s closely related to the  
 problem of how to parallelise a computation, and much of the art of building  
 efficient distributed systems lies in arranging matters so that processes are either  
 simply sequential or completely parallel.

The traditional bank algorithm was to batch the transactions overnight and

apply all the credits for each account before applying all the debits. Inputs from  
 devices such as ATMs and check sorters were ﬁrst batched up into journals  
 before the overnight reconciliation. Payments which bounce then have to be  
 reversed out – and in the case of ATM and debit transactions where the cash  
 has already gone, you can end up with customers borrowing money without  
 authorisation. In practice, chains of failed payments terminate. In recent years,  
 one country after another has introduced *real-time gross settlement* (RTGS)  
 systems in which transactions are booked in order of arrival. There are several  
 subtle downsides. First, at many institutions, the real-time system for retail  
 customers is an overlay on a platform that still works by overnight updates.  
 Second, the outcome can depend on the order of transactions, which can depend  
 on human, system and network vagaries, which can be an issue when many very  
 large payments are made between ﬁnancial institutions. Credit cards operate a  
 hybrid strategy, with credit limits run in real time while settlement is run just  
 as in an old-fashioned checking account.

In the late 2010s, the wave of interest in cryptocurrency has led some en-

trepreneurs to believe that a blockchain might solve the problems of inconsistent  
 update, simplifying applications such as supply-chain management. The energy  
 costs rule out a blockchain based on proof-of-work for most applications; but  
 might some other kind of append-only public ledger ﬁnd a killer app? We will  
 have to wait and see. Meanwhile, the cryptocurrency community makes exten-  
 sive use of off-chain mechanisms that are often very reminiscent of the checking-  
 account approach: disconnected applications propose tentative updates that are  
 later reconciled and applied to the main chain. Experience suggests that there  
 is no magic solution that works in the general case, short perhaps of having a  
 small number of very large banks that are very competent at technology. We’ll  
 discuss this further in the chapter on banking.

In other systems, the order in which transactions arrive is much less im-

portant. Passports are a good example. Passport issuers only worry about

their creation and expiration dates, not the order in which visas are stamped

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 240 | Ross Anderson |

*7.2. CONCURRENCY*

on them3.

**7.2.4** **Deadlock**

Another problem is deadlock, where two systems are each waiting for the other  
 to move ﬁrst. Edsger Dijkstra famously explained this problem, and its possible  
 solutions, via the *dining philosophers’ problem*. A number of philosophers are  
 seated round a table, with a chopstick between each of them; and a philosopher  
 can only eat when they can pick up the two chopsticks on either side. So if all  
 of them try to eat at once and each picks up the chopstick on their right, they  
 get stuck [560].

This can get really complex when you have multiple hierarchies of locks

distributed across systems, some of which fail (and where failures can mean  
 that the locks aren’t reliable) [151]. And deadlock is not just about technology;  
 the phrase ‘Catch-22’ has become popular to describe deadlocks in bureaucratic  
 processes 4. Where a process is manual, some fudge may be found to get round  
 the catch, but when everything becomes software, this option may no longer be  
 available.

In a well known business problem – the *battle of the forms* – one company

issues an order with its own contract terms attached, another company accepts  
 it subject to its own terms, and trading proceeds without any further agreement.  
 In the old days, the matter might only be resolved if something went wrong and  
 the companies ended up in court; even so, one company’s terms might specify  
 an American court while the other’s specify one in England. As trading has  
 become more electronic, the winner is often the company that can compel the  
 loser to trade using its website and thus accept its terms and conditions. Firms  
 increasingly try to make sure that things fail in their favour. The resulting

liability games can have rather negative outcomes for both security and safety;  
 we’ll discuss them further in the chapter on economics.

**7.2.5** **Non-convergent state**

When designing protocols that update the state of a distributed system, the  
 ‘motherhood and apple pie’ is ACID – that transactions should be *atomic,*  
 *consistent, isolated and durable*. A transaction is atomic if you ‘do it all or

not at all’ – which makes it easier to recover after a failure. It is consistent if  
 some invariant is preserved, such as that the books must still balance. This is  
 common in banking systems, and is achieved by insisting that the sum total of  
 credits and debits made by each transaction is zero (I’ll discuss this more in  
 the chapter on banking and bookkeeping). Transactions are isolated if they are  
 serialisable, and they are durable if once done they can’t be undone.

These properties can be too much, or not enough, or both. On the one

hand, each of them can fail or be attacked in numerous obscure ways; on the

3Many Arab countries won’t let you in with an Israeli stamp on your passport, but most

pure identiﬁcation systems are essentially stateless.

4Joseph Heller’s 1961 novel of that name described multiple instances of inconsistent and

crazy rules in the World War 2 military bureaucracy.

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 241 | Ross Anderson |

*7.2. CONCURRENCY*

other, it’s often sufficient to design the system to be *convergent*. This means  
 that, if the transaction volume were to tail off, then eventually there would  
 be consistent state throughout [1353]. Convergence is usually achieved using  
 semantic tricks such as timestamps and version numbers; this can often be  
 enough where transactions get appended to ﬁles rather than overwritten.

In real life, you also need ways to survive things that go wrong and are

not completely recoverable. The life of a security or audit manager can be a  
 constant battle against entropy: apparent deﬁcits (and surpluses) are always  
 turning up, and sometimes simply can’t be explained. For example, different  
 national systems have different ideas of which ﬁelds in bank transaction records  
 are mandatory or optional, so payment gateways often have to guess data in  
 order to make things work. Sometimes they guess wrong; and sometimes people  
 see and exploit vulnerabilities which aren’t understood until much later (if ever).  
 In the end, things may get fudged by adding a correction factor and setting a  
 target for keeping it below a certain annual threshold.

Durability is a subject of debate in transaction processing. The advent of

phishing and keylogging attacks has meant that some small proportion of bank  
 accounts will at any time be under the control of criminals; money gets moved  
 both from them and through them. When an account compromise is detected,  
 the bank moves to freeze it and perhaps to reverse payments that have recently  
 been made from it. The phishermen naturally try to move funds through in-  
 stitutions, or jurisdictions, that don’t do transaction reversal, or do it at best  
 slowly and grudgingly [75]. This sets up a tension between the recoverability  
 and thus the resilience of the payment system on the one hand and transaction  
 durability and ﬁnality on the other5.

**7.2.6** **Secure time**

The ﬁnal concurrency problem of special interest to the security engineer is the  
 provision of accurate time. As authentication protocols such as Kerberos can  
 be attacked by inducing clock error, it’s not enough to simply trust a random  
 external time source. One possibility is a *Cinderella attack*: if a security critical  
 program such as a ﬁrewall has a licence with a timelock, an attacker might wind  
 your clock forward “and cause your ﬁrewall to turn into a pumpkin”. Given the  
 spread of IoT devices that may be safety-critical and use time in ways that are  
 poorly understood, there is now some concern about possible large-scale service  
 denial attacks. Time is a lot harder than it looks: even if you have an atomic  
 clock, leap seconds cannot be predicted but need to be broadcast somehow;  
 some minutes have 61 and even 62 seconds; odd time effects can be a security  
 issue6; and much of the world is not using the Gregorian calendar.

5This problem goes back centuries, with a thicket of laws around whether someone acting

in good faith can acquire good title to stolen goods or stolen funds. The Bills of Exchange  
 Act 1882 gave good title to people who bought bills of exchange in good faith, even if they  
 were stolen. Something similar used to hold for stolen goods bought in an open market, but  
 that was eventually repealed. In the case of electronic payments, the banks acted as a cartel  
 to make payments ﬁnal more quickly, both via card network rules and by lobbying European  
 institutions over the Payment Services Directives. As for the case of bitcoin, it’s still in ﬂux;  
 see section 20.7.5.

6Some ATMs didn’t check customer balances for a few days after Y2K, leading to unau-

thorised overdrafts once the word got round

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 242 | Ross Anderson |

*7.3. FAULT TOLERANCE AND FAILURE RECOVERY*

Anyway, there are several possible approaches to the provision of secure time.

You can give every computer a radio clock, and indeed your smartphone has  
 GPS – but that can be jammed by a passing truck driver. You can abandon  
 absolute time and instead use *Lamport time*, in which all you care about is  
 whether event A happened before event B rather than what date it is [1122].  
 For robustness reasons, Google doesn’t use time in its internal certiﬁcates, but  
 uses ranges of serial numbers coupled to a revocation mechanism [23].

In many applications, you may end up using the *network time protocol*

(NTP). This has a moderate amount of protection, with clock voting and au-  
 thentication of time servers, and is dependable enough for many purposes. How-  
 ever, you still need to take care. For example, Netgear hardwired their home  
 routers to use an NTP server at the University of Wisconsin-Madison, which  
 was swamped with hundreds of thousands of packets a second; Netgear ended  
 up having to pay them $375,000 to maintain the time service for three years.  
 Shortly afterwards, D-Link repeated the same mistake [445]. Second, from 2016  
 there have been denial-of-service attacks using NTP servers as force multipli-  
 ers; millions of servers turned out to be abusable, so many ISPs and even IXPs  
 started blocking them. So if you’re planning to deploy lots of devices outside  
 your corporate network that will rely on NTP, you’d better think hard about  
 which servers you want to trust and pay attention to the latest guidance from  
 CERT [1797].

**7.3** **Fault Tolerance and Failure Recovery**

Failure recovery is often the most important aspect of security engineering, yet  
 it is one of the most neglected. For many years, most of the research papers  
 on computer security have dealt with conﬁdentiality, and most of the rest with  
 authenticity and integrity; availability has almost been ignored. Yet the actual  
 expenditures of a modern information business – whether a bank or a search  
 engine – are the other way round. Far more is spent on availability and recovery  
 mechanisms, such as multiple processing sites and redundant networks, than in  
 integrity mechanisms such as code review and internal audit, and this in turn is  
 way more than is spent on encryption. As you read through this book, you’ll see  
 that many other applications, from burglar alarms through electronic warfare to  
 protecting a company from DDoS attacks, are fundamentally about availability.  
 Fault tolerance and failure recovery are often the core of the security engineer’s  
 job.

Classical fault tolerance is usually based on redundancy, fortiﬁed using mech-

anisms such as logs and locking, and is greatly complicated when it must with-  
 stand malicious attacks on these mechanisms. Fault tolerance interacts with  
 security in a number of ways: the failure model, the nature of resilience, the  
 location of redundancy used to provide it, and defence against service denial  
 attacks. I’ll use the following deﬁnitions: a *fault* may cause an *error*, which  
 is an incorrect state; this may lead to a *failure*, which is a deviation from the  
 system’s speciﬁed behavior. The resilience which we build into a system to tol-  
 erate faults and recover from failures will have a number of components, such  
 as fault detection, error recovery and if necessary failure recovery. The meaning

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 243 | Ross Anderson |

*7.3. FAULT TOLERANCE AND FAILURE RECOVERY*

of *mean-time-before-failure* (MTBF) and *mean-time-to-repair* (MTTR) should  
 be obvious.

**7.3.1** **Failure models**

In order to decide what sort of resilience we need, we must know what sort of  
 attacks to expect. Much of this will come from an analysis of threats speciﬁc to  
 our system’s operating environment, but some general issues bear mentioning.

**7.3.1.1** **Byzantine failure**

First, the failures with which we are concerned may be normal or malicious,  
 and we often model the latter as *Byzantine*. Byzantine failures are inspired by  
 the idea that there are *n* generals defending Byzantium, *t* of whom have been  
 bribed by the attacking Turks to cause as much confusion as possible. The

generals can pass oral messages by courier, and the couriers are trustworthy, so  
 each general can exchange conﬁdential and authentic communications with each  
 other general (we could imagine them encrypting and computing a MAC on each  
 message). What is the maximum number *t* of traitors that can be tolerated?

The key observation is that if we have only three generals, say Anthony,

Basil and Charalampos, and Anthony is the traitor, then he can tell Basil “let’s  
 attack” and Charalampos “let’s retreat”. Basil can now say to Charalampos  
 “Anthony says let’s attack”, but this doesn’t let Charalampos conclude that  
 Anthony’s the traitor. It could just as easily have been Basil; Anthony could  
 have said “let’s retreat” to both of them, but Basil lied when he said “Anthony  
 says let’s attack”.

This beautiful insight is due to Leslie Lamport, Robert Shostak and Marshall

Pease, who proved that the problem has a solution if and only if *n ffi* 3*t*+1 [1124].  
 different things to two different colleagues. This illustrates the power of digital  
 signatures in particular and of end-to-end security mechanisms in general. There  
 is now a substantial literature on Byzantine fault tolerance – the detailed design  
 of systems able to withstand this kind of failure; see for example the algorithm  
 by Miguel Castro and Barbara Liskov [394].

Another lesson is that if a component which fails (or can be induced to fail

by an opponent) gives the wrong answer rather than just no answer, then it’s  
 much harder to use it to build a resilient system. It can be useful if components  
 that fail just stop, or if they can at least be quickly identiﬁed and blacklisted.

**7.3.1.2** **Interaction with fault tolerance**

So we can constrain the failure rate in a number of ways. The two most obvi-  
 ous are by using *redundancy* and *fail-stop processes*. The latter process error-  
 correction information along with data, and stop when an inconsistency is de-  
 tected; for example, bank transaction processing will typically stop if an out-of-  
 balance condition is detected after a processing task. The two may be combined;  
 the processors used in some safety-critical functions in cars and aircraft typically

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 244 | Ross Anderson |

*7.3. FAULT TOLERANCE AND FAILURE RECOVERY*

have two or more cores. There was pioneering work on a *fault-tolerant multipro-*  
 *cessor* (FTMP) in the 1970s, driven by the Space Shuttle project; this explored  
 which components should be redundant and the associated design trade-offs  
 around where the error detection takes places and how closely everything is  
 synchronised [920]. Such research ended up driving the design of fault-tolerant  
 processors used in various submarines and spacecraft, as well as architectures  
 used by Boeing and Airbus. The FTMP idea was also commercialised by Tan-  
 dem and then by Stratus, which sold machines for payment processing. The  
 Stratus had two disks, two buses and even two CPUs, each of which would stop  
 if it detected errors; the fail-stop CPUs were built by having two CPU chips on  
 the same card and comparing their outputs. If they disagreed the output went  
 open-circuit. A replacement card would arrive in the post; you’d take it down  
 to the machine room, notice that card 5 had a ﬂashing red light, pull it out  
 and replace it with the new one – all while the machine was processing dozens  
 of transactions per second. Nowadays, the data centres of large service ﬁrms  
 have much more elaborate protocols to ensure that if a machine fails, another  
 machine takes over; if a rack fails, another rack takes over; and even if a data  
 centre fails, its workload is quickly recovered on others. Google was a leader  
 in developing the relevant software stack, having discovered in the early 2000s  
 that it was much cheaper to build large-scale systems with commodity PCs and  
 smart software than to buy ever-larger servers from specialist vendors.

While redundancy can make a system more *resilient*, it has costs. First,

we have to deal with a more complex software stack and toolchain. Banks

eventually moved away from Stratus because they found it was less reliable  
 overall than traditional mainframes: although there was less downtime due to  
 hardware failure, this didn’t compensate for the extra software failure caused by  
 an unfamiliar development environment. Second, if I have multiple sites with  
 backup data, then conﬁdentiality could fail if any of them gets compromised7;  
 and if I have some data that I have a duty to destroy, then purging it from  
 multiple backup tapes can be a headache. The modern-day issue with developing  
 software in containers on top of redundant cloud services is not so much the  
 programming languages, or compromise via data centres; it’s that developers  
 are unfamiliar with the cloud service providers’ access control tools and all too  
 often leave sensitive data world-readable.

There are other traps for the unwary. In one case in which I was called as

an expert, my client was arrested while using a credit card in a store, accused of  
 having a forged card, and beaten up by the police. He was adamant that the card  
 was genuine. Much later, we got the card examined by VISA, who conﬁrmed  
 that it was indeed genuine. What happened, as well as we can reconstruct it,  
 was this. Credit cards have two types of redundancy on the magnetic strip – a  
 simple checksum obtained by combining together all the bytes on the track using  
 exclusive-or, and a cryptographic checksum which we’ll describe in detail later  
 in section 12.5.1. The former is there to detect errors, and the latter to detect  
 forgery. It appears that in this particular case, the merchant’s card reader was  
 out of alignment in such a way as to cause an even number of bit errors which  
 cancelled each other out by chance in the simple checksum, while causing the

7Or the communications between your data centres get tapped; we discussed in section 2.1

how GCHQ did that to Google.

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 245 | Ross Anderson |

*7.3. FAULT TOLERANCE AND FAILURE RECOVERY*

crypto checksum to fail. The result was a false alarm, and a major disruption  
 in my client’s life.

Redundancy is hard enough to deal with in mechanical systems. For ex-

ample, training pilots to handle multi-engine aircraft involves drilling them on  
 engine failure procedures, ﬁrst in the simulator and then in real aircraft with  
 an instructor. Novice pilots are in fact more likely to be killed by an engine  
 failure in a multi-engine plane than in a single; landing in the nearest ﬁeld is  
 less hazardous for them than coping with sudden asymmetric thrust. The same  
 goes for instrument failures; it doesn’t help to have three artiﬁcial horizons in  
 the cockpit if, under stress, you rely on the one that’s broken. Aircraft are much  
 simpler than many modern information systems – yet there are still air crashes  
 when pilots fail to manage the redundancy that’s supposed to keep them safe.  
 There are also complex failures, as when two Boeing 737 Max aircraft crashed  
 because of failures in a single sensor, when the plane had two but the software  
 failed to read them both, and the pilots hadn’t been trained how to diagnose  
 the problem and manage the consequences. All too often, system designers put  
 in multiple protection mechanisms and don’t think through the consequences  
 carefully enough. Many other safety failures are failures of usability, and the  
 same applies to security, as we discussed in Chapter 3; redundancy isn’t an  
 antidote to poor design.

**7.3.2** **What is resilience for?**

When introducing redundancy or other resilience mechanisms into a system,  
 we need to understand what they’re for and the incentives facing the various  
 actors. It therefore matters whether the resilience is local or crosses geographical  
 or organisational boundaries.

In the ﬁrst case, replication can be an internal feature of the server to make

it more trustworthy. I already mentioned 1980s systems such as Stratus and  
 Tandem; then we had replication of standard hardware at the component level,  
 such as *redundant arrays of inexpensive disks* (RAID). Since the late 1990s  
 there has been massive investment in developing rack-scale systems that let  
 multiple cheap PCs do the work of expensive servers, with mechanisms to ensure  
 a single server that fails will have its workload taken over rapidly by another,  
 and indeed a rack that fails can also be recovered on a hot spare. These are  
 now a standard component of cloud service architecture: any ﬁrm operating  
 hundreds of thousands of servers will have so many failures that recovery must  
 be largely automated.

But often things are much more complicated. A service may have to assume

that some of its clients are trying to cheat it and may also have to rely on a  
 number of services, none of which is completely accurate. When opening a bank  
 account, or issuing a passport, we might want to check against services from  
 voter rolls through credit reference agencies to a database of driver’s licences,  
 and the results may often be inconsistent. Trust decisions may involve complex  
 logic, not entirely unlike the systems used in electronic warfare to try to work  
 out which of your inputs are being jammed. (I’ll discuss these further in the  
 chapter on electronic and information warfare.)

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 246 | Ross Anderson |

*7.3. FAULT TOLERANCE AND FAILURE RECOVERY*

The direction of mistrust has an effect on protocol design. A server faced

with multiple untrustworthy clients and a client relying on multiple servers that  
 may be incompetent, unavailable or malicious will both wish to control the ﬂow  
 of messages in a protocol in order to contain the effects of service denial. It’s  
 hard to design systems for the real world in which everyone is unreliable and all  
 are mutually suspicious.

Sometimes the emphasis is on *security renewability*. The obvious example

here is bank cards: a bank can upgrade security from time to time by mailing out  
 newer versions of its cards, whether upgrading from mag strip to chip or from  
 cheap chips to more sophisticated ones; and it can recover from a compromise  
 by mailing out cards out of cycle to affected customers. Pay TV and mobile  
 phones are somewhat similar.

**7.3.3** **At what level is the redundancy?**

Systems may be made resilient against errors, attacks and equipment failures  
 at a number of levels. As with access control, these become progressively more  
 complex and less reliable as we go up to higher layers in the system.

Some computers have been built with redundancy at the hardware level, such

as Stratus systems and RAID discs I mentioned earlier. But simple replication  
 cannot provide a defense against malicious software, or against an intruder who  
 exploits faulty software.

At the next level up, there is *process group redundancy*. Here, we may run

multiple copies of a system on multiple servers in different locations and com-  
 pare their outputs. This can stop the kind of attack in which the opponent gets  
 physical access to a machine and subverts it, whether by mechanical destruction  
 or by inserting unauthorised software. It can’t defend against attacks by autho-  
 rised users or damage by bad authorised software, which could simply order the  
 deletion of a critical ﬁle.

The next level is *backup*, where we typically take a copy of the system (a

*checkpoint*) at regular intervals. The copies are usually kept on media that

can’t be overwritten such as write-protected tapes or discs with special software.  
 We may also keep *journals* of all the transactions applied between checkpoints.  
 Whatever the detail, backup and recovery mechanisms not only enable us to  
 recover from physical asset destruction, they also ensure that if we do get an  
 attack at the logical level, we have some hope of recovering. The classic example  
 in the 1980s would have been a time bomb that deletes the customer database  
 on a speciﬁc date; since the arrival of cryptocurrency, the fashion has been for  
 ransomware.

Businesses with critical service requirements, such as banks and retailers,

have had backup data centres for many years. The idea is that if the main  
 centre goes down, the service will *failover* to a second facility. Maintaining such  
 facilities absorbed most of a typical bank’s information security budget.

Backup is not the same as *fallback*. A fallback system is typically a less

capable system to which processing reverts when the main system is unavailable.  
 One example was the use of manual imprinting machines to capture credit card

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 247 | Ross Anderson |

*7.3. FAULT TOLERANCE AND FAILURE RECOVERY*

transactions from the card embossing when electronic terminals failed. Fallback  
 systems are an example of redundancy in the application layer – the highest  
 layer we can put it.

It is important to realise that these are different mechanisms, which do

different things. Redundant disks won’t protect against a malicious programmer  
 who deletes all your account ﬁles, and backups won’t stop him if rather than just  
 deleting ﬁles he writes code that slowly inserts more and more errors8. Neither  
 will give much protection against attacks on data conﬁdentiality. On the other  
 hand, the best encryption in the world won’t help you if your data processing  
 center burns down. Real-world recovery plans and mechanisms involve a mixture  
 of all of the above.

The remarks that I made earlier about the difficulty of redundancy, and

the absolute need to plan and train for it properly, apply in spades to system  
 backup. When I was working in banking in the 1980s, we reckoned that we  
 could probably get our backup system working within an hour or so of our main  
 processing centre being destroyed, but the tests were limited by the fact that we  
 didn’t want to risk processing during business hours: we would recover the main  
 production systems on our backup data centre one Saturday a year. By the early  
 1990s, Tesco, a UK supermarket, had gotten as far as live drills: they’d pull the  
 plug on the main processing centre once a year without warning the operators,  
 to make sure the backup came up within 40 seconds. By 2011, Netﬂix had

developed ‘chaos monkeys’ – systems that would randomly knock out a machine,  
 or a rack, or even a whole data centre, to test resilience constantly. By 2019,  
 large service ﬁrms have gotten to such a scale that they don’t need this. If you  
 have three million machines across thirty data centres, then you’ll lose machines  
 constantly, racks frequently, and whole data centres often enough that you have  
 to engineer things to keep going. So nowadays, you can simply pay money and a  
 cloud service provider will worry about a lot of the detail for you. But you need  
 to really understand what sort of failures Amazon or Google or Microsoft can  
 handle for you and what you have to deal with yourself. The standard service  
 level agreements of the major providers allow them to interrupt your service  
 for quite a few hours per month, and if you use a smaller cloud service (even a  
 government cloud), it will have capacity limits about which you have to think  
 carefully.

It’s worth trying to work out which services you depend on that are outside

your direct supply chain. For example, Britain suffered a fuel tanker drivers’  
 strike in 2001, and some hospitals had to close because of staff shortages, which  
 was supposed to not happen. The government had allocated petrol rations to  
 doctors and nurses, but not to schoolteachers. So the schools closed, and the  
 nurses had to stay home to look after their kids, and this closed hospitals too.  
 This helped the strikers defeat Prime Minister Tony Blair: he abandoned his  
 signature environmental policy of steadily increasing fuel duty. As we become  
 increasingly dependent on each other, contingency planning gets ever harder.

8Nowadays the really serious ransomware operators will hack your system, add ﬁle encryp-

tion surreptitiously and wait before they pounce – so they hold hostage not just your current  
 data but several weeks’ backups too

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 248 | Ross Anderson |

*7.3. FAULT TOLERANCE AND FAILURE RECOVERY*

**7.3.4** **Service-denial attacks**

One of the reasons we want security services to be fault-tolerant is to make  
 service-denial attacks less attractive, less effective, or both. Such attacks are  
 often used as part of a larger plan. For example, one might take down a security  
 server to force other servers to use cached copies of credentials, or swamp a web  
 server to take it temporarily offline and then get another machine to serve the  
 pages that victims try to download.

A powerful defense against service denial is to prevent the opponent from

mounting a selective attack. If principals are anonymous – say there are several  
 equivalent services behind a load balancer, and the opponent has no idea which  
 one to attack – then he may be ineffective. I’ll discuss this further in the context  
 of burglar alarms and electronic warfare.

Where this isn’t possible, and the opponent knows where to attack, then

there are some types of service-denial attacks that can be stopped by redundancy  
 and resilience mechanisms and others that can’t. For example, the TCP/IP  
 protocol has few effective mechanisms for hosts to protect themselves against  
 network ﬂooding, which comes in a wide variety of ﬂavours. Defense against this  
 kind of attack tends to involve moving your site to a beeﬁer hosting service with  
 specialist packet-washing hardware – or tracing and arresting the perpetrator.

Distributed denial-of-service (DDoS) attacks came to public notice when

they were used to bring down Panix, a New York ISP, for several days in 1996.  
 During the late 1990s they were occasionally used by script kiddies to take down  
 chat servers. In 2001 I mentioned them in passing in the ﬁrst edition of this book.  
 Over the following three years, extortionists started using them; they’d assemble  
 a *botnet*, a network of compromised PCs, which would ﬂood a target webserver  
 with packet traffic until its owner paid them to desist. Typical targets were  
 online bookmakers, and amounts of $10,000 – $50,000 were typically demanded  
 to leave them alone, and the typical bookie paid up the ﬁrst time this happened.  
 When the attacks persisted, the ﬁrst solution was replication: operators moved  
 their websites to hosting services such as Akamai whose servers are so numerous  
 (and so close to customers) that they can shrug off anything the average botnet  
 could throw at them. In the end, the blackmail problem was solved when the  
 bookmakers met and agreed not to pay any more blackmail money, and the  
 Ukrainian police were prodded into arresting the gang responsible.

By 2018, we had come full circle, and about ﬁfty bad people were operating

DDoS-as-a-service, mostly for gamers who wanted to take down their opponents’  
 teamspeak servers. The services were sold online as ‘booters’ that would boot  
 your opponents out of the game; a few dollars would get a ﬂood of perhaps  
 100Gbit/sec. Service operators also called them, more euphemistically, ‘stres-  
 sors’ – with the line that you could use them to test the robustness of your own  
 website. This didn’t fool anyone, and just before Christmas 2018 the FBI took  
 down ﬁfteen of these sites, arresting a number of their operators and causing  
 the volumes of DDoS traffic to drop noticeably for several months [1445].

Finally, where a more vulnerable fallback system exists, a common technique

is to use a service-denial attack to force victims into fallback mode. The classic  
 example is in payment cards. Smartcards are generally harder to forge than

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 249 | Ross Anderson |

*7.4. NAMING*

magnetic strip cards, but perhaps 1% of them fail every year, thanks to static  
 electricity and worn contacts. Also, some tourists still use magnetic strip cards.  
 So most card payment systems still have a fallback mode that uses the magnetic  
 strip. A simple attack is to use a false terminal, or a bug inserted into the cable to  
 a genuine terminal, to capture card details and then write them to the magnetic  
 strip of a card with a dead chip.

**7.4** **Naming**

Naming is a minor if troublesome aspect of ordinary distributed systems, but  
 it becomes surprisingly hard in security engineering. During the dotcom boom  
 in the 1990s, when SSL was invented and we started building public-key cer-  
 tiﬁcation authorities, we hit the problem of what names to put on certiﬁcates.  
 A certiﬁcate that says simply “the person named Ross Anderson is allowed to  
 administer machine X” is little use. I used to be the only Ross Anderson I knew  
 of; but as soon as the ﬁrst search engines came along, I found dozens of us. I  
 am also known by different names to dozens of different systems. Names exist  
 in contexts, and naming the principals in secure systems is becoming ever more  
 important and difficult.

Conceptually, namespaces can be hierarchical or ﬂat. You can identify me

as ‘The Ross Anderson who teaches computer science at Cambridge, England’  
 or as ‘The Ross Anderson who’s rossjanderson@gmail.com’ or even as ‘the Ross  
 Anderson with such-and-such a passport number’. But these are not the same  
 kind of thing, and linking them causes all sorts of problems.

In general, using more names increases complexity. A public-key certiﬁcate

that simply says “this is the key to administer machine X” is a bearer token,  
 just like a metal door key; whoever controls the private key for that certiﬁcate  
 is the admin, just as if the root password were in an envelope in a bank vault.  
 But once my name is involved, and I have to present some kind of passport or  
 ID card to prove who I am, the system acquires a further dependency. If my  
 passport is compromised the consequences could be far-reaching, and I really  
 don’t want to give the government an incentive to issue a false passport in my  
 name to one of its agents.

After 9/11, governments started to force businesses to demand government-

issue photo ID in places where this was not previously thought necessary. In  
 the UK, for example, you can no longer board a domestic ﬂight using just the  
 credit card with which you bought the ticket; you have to produce a passport  
 or driving license – which you also need to order a bank transfer in a branch for  
 more than £1000, to rent an apartment, to hire a lawyer or even to get a job.  
 Such measures are not only inconvenient but introduce new failure modes into  
 all sorts of systems.

There is a second reason that the world is moving towards larger, ﬂatter name

spaces: the growing dominance of the large service ﬁrms in online authentication.  
 Your name is increasingly a global one; it’s your Gmail or Hotmail address,  
 your Twitter handle, or your Facebook account. These ﬁrms have not merely  
 beneﬁted from the technical externalities, which we discussed in the chapter on

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 250 | Ross Anderson |

*7.4. NAMING*

authentication, and business externalities, which we’ll discuss in the chapter on  
 economics, they have sort-of solved some of the problems of naming. But we  
 can’t be complacent as many other problems remain. So it’s useful to canter  
 through what a generation of computer science researchers have learned about  
 naming in distributed systems.

**7.4.1** **The Needham naming principles**

During the last quarter of the twentieth century, engineers building distributed  
 systems ran up against many naming problems. The basic algorithm used to  
 bind names to addresses is known as *rendezvous*: the principal exporting a name  
 advertises it somewhere, and the principal seeking to import and use it searches  
 for it. Obvious examples include phone books and ﬁle system directories.

People building distributed systems soon realised that naming gets complex

quickly, and the lessons are set out in a classic article by Needham [1424]. Here  
 are his ten principles.

1. *The function of names is to facilitate sharing*. This continues to hold:

my bank account number exists in order to share the information that  
 I deposited money last week with the teller from whom I am trying to  
 withdraw money this week. In general, names are needed when the data  
 to be shared is changeable. If I only ever wished to withdraw exactly

the same sum as I’d deposited, a bearer deposit certiﬁcate would be ﬁne.  
 Conversely, names need not be shared – or linked – where data will not  
 be; there is no need to link my bank account number to my telephone  
 number unless I am going to pay my phone bill from the account.

2. *The naming information may not all be in one place, and so resolving*

*names brings all the general problems of a distributed system*. This holds  
 with a vengeance. A link between a bank account and a phone number  
 assumes both of them will remain stable. So each system relies on the  
 other, and an attack on one can affect the other. Many banks use two-  
 channel authorisation to combat phishing – if you order a payment online,  
 you get a text message on your mobile phone saying ‘if you want to pay $X  
 to account Y, please enter the following four-digit code into your browser’.  
 The standard attack is for the crook to claim to be you to the phone  
 company and report the loss of your phone. So they give him a new SIM  
 that works for your phone number, and he makes off with your money.  
 The phone company could stop that, but it doesn’t care too much about  
 authentication, as all it stands to lose is some airtime, whose marginal  
 cost is zero. And the latest attack is to use Android malware to steal  
 authentication codes. Google could stop that by locking down the Android  
 platform as tightly as Apple – but it lacks the incentive to do so.

3. *It is bad to assume that only so many names will be needed*. The shortage

of IP addresses, which motivated the development of IP version 6 (IPv6), is  
 well enough discussed. What is less well known is that the most expensive  
 upgrade the credit card industry ever had to make was the move from  
 thirteen-digit credit card numbers to sixteen. Issuers originally assumed

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 251 | Ross Anderson |

*7.4. NAMING*

that thirteen digits would be enough, but the system ended up with tens  
 of thousands of banks – many with dozens of products – so a six-digit  
 bank identiﬁcation number was needed. Some issuers have millions of

customers, so a nine-digit account number is the norm. And there’s also  
 a *check digit* to detect errors.

4. *Global names buy you less than you think.* For example, the 128-bit ad-

dress in IPv6 can in theory enable every object in the universe to have  
 a unique name. However, for us to do business, a local name at my end  
 must be resolved into this unique name and back into a local name at your  
 end. Invoking a unique name in the middle may not buy us anything; it  
 may even get in the way if the unique naming service takes time, costs  
 money, or occasionally fails (as it surely will). In fact, the name service  
 itself will usually have to be a distributed system, of the same scale (and  
 security level) as the system we’re trying to protect. So we can expect  
 no silver bullets from this quarter. Adding an extra name, or adopting  
 a more complicated one, has the potential to add extra costs and failure  
 modes.

5. *Names imply commitments, so keep the scheme ﬂexible enough to cope with*

*organisational changes*. This sound principle was ignored in the design  
 of the UK government’s key management system for secure email [115].  
 There, principals’ private keys are generated from their email addresses.  
 So the frequent reorganisations meant that the security infrastructure had  
 to be rebuilt each time – and that more money had to be spent solving  
 secondary problems such as how people access old material.

6. *Names may double as access tickets, or capabilities*. We have already seen

a number of examples of this in Chapters 2 and 3. In general, it’s a bad idea  
 to assume that today’s name won’t be tomorrow’s password or capability  
 – remember the Utrecht fraud we discussed in section 4.5. Norway, for  
 example, used to consider the citizen’s ID number to be public, but it  
 ended up being used as a sort of password in so many applications that  
 they had to relent and make it private. There are similar issues around  
 the US Social Security Number (SSN). So the Department of Defense  
 created a surrogate number called the EDIPI, which was supposed to be  
 not sensitive; but, sure enough, people started using it as an authenticator  
 instead of as an identiﬁer.

I’ve given a number of examples of how things go wrong when a name  
 starts being used as a password. But sometimes the roles of name and  
 password are ambiguous. In order to get entry to a car park I used to use  
 at the university, I had to speak my surname and parking badge number  
 into a microphone at the barrier. So if I say, “Anderson, 123”, which

of these is the password? In fact it was “Anderson”, as anyone can walk  
 through the car park and note down valid badge numbers from the parking  
 permits on the car windscreens.

7. *Things are made much simpler if an incorrect name is obvious*. In stan-

dard distributed systems, this enables us to take a liberal attitude to  
 caching. In payment systems, credit card numbers used to be accepted  
 while the terminal was offline so long as the credit card number appears

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 252 | Ross Anderson |

*7.4. NAMING*

valid (i.e., the last digit is a proper check digit of the ﬁrst ﬁfteen) and it is  
 not on the hot card list. The certiﬁcates on modern chip cards provide a  
 higher-quality implementation of the same basic concept; authentication  
 mechanisms such as crypto and security printing can give the added bene-  
 ﬁt of making names resilient to spooﬁng. As an example of what can still  
 go wrong, the Irish police created over 50 dockets for Mr ‘Prawo Jazdy’,  
 wanted for failing to pay over ﬁfty traffic tickets – until they realised that  
 this is Polish for ‘Driving licence’ [192].

8. *Consistency is hard, and is often fudged. If directories are replicated, then*

*you may ﬁnd yourself unable to read, or to write, depending on whether*  
 *too many or too few directories are available*. Naming consistency causes  
 problems for business in a number of ways, of which perhaps the most no-  
 torious is the bar code system. Although this is simple enough in theory  
 – with a unique numerical code for each product – in practice different  
 manufacturers, distributors and retailers attach quite different descrip-  
 tions to the bar codes in their databases. Thus a search for products by  
 ‘Kellogg’s’ will throw up quite different results depending on whether or  
 not an apostrophe is inserted, and this can cause confusion in the supply  
 chain. Proposals to ﬁx this problem can be surprisingly complicated [914].  
 There are also the issues of convergence discussed above; data might not  
 be consistent across a system, even in theory. There are also the problems  
 of timeliness, such as whether a product has been recalled.

9. *Don’t get too smart. Phone numbers are much more robust than computer*

*addresses*. Early secure messaging systems – from PGP to government  
 systems – tried to link keys to email addresses, but these change when  
 people’s jobs do. More modern systems such as Signal and WhatsApp use  
 mobile phone numbers instead. In the same way, early attempts to replace  
 bank account numbers and credit card numbers with public-key certiﬁcates  
 in protocols like SET failed, though in some mobile payment systems, such  
 as Kenya’s M-Pesa, they’ve been replaced by phone numbers. (I’ll discuss  
 further speciﬁc problems of public key infrastructures in section 21.6.)

10. *Some names are bound early, others not; and in general it is a bad thing*

*to bind early if you can avoid it*. A prudent programmer will normally  
 avoid coding absolute addresses or ﬁlenames as that would make it hard  
 to upgrade or replace a machine. It’s usually better to leave this to a  
 conﬁguration ﬁle or an external service such as DNS. Yet secure systems  
 often want stable and accountable names as any third-party service used  
 for last-minute resolution could be a point of attack. Designers therefore  
 need to pay attention to where the naming information goes, how devices  
 get personalised with it, and how they get upgraded – including the names  
 of services on which the security may depend, such as the NTP service  
 discussed in section 7.2.6 above.

**7.4.2** **What else goes wrong**

The Needham principles were crafted for the world of the early 1990s in which  
 naming systems could be imposed at the system owner’s convenience. Once we

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 253 | Ross Anderson |

*7.4. NAMING*

moved to the reality of modern web-based (and interlinked) service industries,  
 operating at global scale, we found that there is more to add.

By the early 2000s, we had learned that no naming system can be globally

unique, decentralised and human-meaningful. In fact, it’s a classic trilemma:  
 you can only have two of those attributes (Zooko’s triangle) [37]. In the past,  
 engineers went for naming systems that were unique and meaningful, like URLs,  
 or unique and decentralised, as with public keys in PGP or the self-signed cer-  
 tiﬁcates that function as app names in Android. Human names are meaningful  
 and local but don’t scale to the Internet. I mentioned above that as soon as  
 the ﬁrst search engines came along, I could instantly ﬁnd dozens of other people  
 called Ross Anderson, but it’s even worse than that; half a dozen worked in ﬁelds  
 I’ve also worked in, such as software engineering and electricity distribution.

The innovation from sites like Facebook is to show on a really large scale that

names don’t have to be unique. We can use social context to build systems that  
 are both decentralised and meaningful – which is just what our brains evolved  
 to cope with. Every Ross Anderson has a different set of friends and you can  
 tell us apart that way.

How can we make sense of all this, and stop it being used to trip people up?

It is sometimes helpful to analyse the properties of names in detail.

**7.4.2.1** **Naming and identity**

First, the principals in security protocols are usually known by many different  
 kinds of name – a bank account number, a company registration number, a  
 personal name plus a date of birth or a postal address, a telephone number, a  
 passport number, a health service patient number, or a userid on a computer  
 system.

A common mistake is to confuse naming with identity. *Identity* is when

two different names (or instances of the same name) correspond to the same  
 principal (this is known to computer scientists as an *indirect name* or *symbolic*  
 *link*). One classic example comes from the registration of title to real estate.  
 Someone who wishes to sell a house often uses a different name than they did  
 at the time it was purchased: they might have changed their name on marriage,  
 or on gender transition, or started using their middle name instead. A land-  
 registration system must cope with a lot of identity issues like this.

There are two types of identity failure leading to compromise: where I’m

happy to impersonate anybody, and where I want to impersonate a speciﬁc in-  
 dividual. The former case includes setting up accounts to launder cybercrime  
 proceeds, while an example of the latter is SIM replacement (I want to clone  
 a CEO’s phone so I can loot a company bank account). If banks (or phone  
 companies) just ask people for two proofs of address, such as utility bills, that’s  
 easy. Demanding government-issue photo ID may require us to analyse state-  
 ments such as “The Aaron Bell who owns bank account number 12345678 is the  
 Aaron James Bell with passport number 98765432 and date of birth 3/4/56”.  
 This may be seen as a symbolic link between two separate systems – the bank’s  
 and the passport office’s. Note that the latter part of this ‘identity’ encapsulates  
 a further statement, which might be something like “The US passport office’s

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 254 | Ross Anderson |

*7.4. NAMING*

ﬁle number 98765432 corresponds to the entry in the New York birth register  
 for 3/4/56 of one Aaron James Bell.” If Aaron is commonly known as Jim, it  
 gets messier still.

In general, names may involve several steps of recursion, which gives attack-

ers a choice of targets. For example, a lot of passport fraud is *pre-issue fraud*:  
 the bad guys apply for passports in the names of genuine citizens who haven’t  
 applied for a passport already and for whom copies of birth certiﬁcates are easy  
 to obtain. Postmortem applications are also common. Linden Labs, the op-  
 erators of Second Life, introduced a scheme whereby you prove you’re over 18  
 by providing the driver’s license number or social security number of someone  
 who is. Now a web search quickly pulls up such data for many people, such as  
 the rapper Tupac Amaru Shakur; and yes, Linden Labs did accept Mr Shakur’s  
 license number – even through the license had expired and he’s dead.

There can also be institutional failure. For example, the United Arab Emi-

rates started taking iris scans of all visitors after women who had been deported  
 to Pakistan for prostitution offences would turn up a few weeks later with a  
 genuine Pakistani passport in a different name and accompanied by a different  
 ‘husband’. Similar problems led many countries to issue biometric visas so they  
 don’t have to depend on passport issuers in countries they don’t want to have  
 to trust.

In addition to corruption, a pervasive failure is the loss of original records. In

countries where registers of births, marriages and deaths are kept locally and on  
 paper, some are lost, and smart impersonators exploit these. You might think  
 that digitisation is ﬁxing this problem, but the long-term preservation of digital  
 records is a hard problem even for rich countries; document formats change,  
 software and hardware become obsolete, and you either have to emulate old  
 machines or translate old data, neither of which is ideal. Various states have run  
 pilot projects on electronic documents that must be kept forever, such as civil  
 registration, but we still lack credible standards. Sensible developed countries  
 still keep paper originals as the long-term document of record. In less developed  
 countries, you may have to steer between the Scylla of ﬂaky government IT and  
 the Charybdis of natural disasters9.

**7.4.2.2** **Cultural assumptions**

The assumptions that underlie names change from one country to another. In  
 the English-speaking world, people may generally use as many names as they  
 please; a name is simply what you are known by. But some countries forbid the  
 use of aliases, and others require them to be registered. The civil registration  
 of births, marriages, civil partnerships, gender transitions and deaths is an ex-  
 tremely complex one, often politicised, tied up with religion in many countries  
 and with the issue of ID documents as well. And incompatible rules between  
 countries cause real problems for migrants, for tourists and indeed for companies  
 with overseas customers.

In earlier editions of this book, I gave as an example that writers who change

9while listening to the siren song of development consultants saying ‘put it on the

blockchain!

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 255 | Ross Anderson |

*7.4. NAMING*

their legal name on marriage often keep publishing using their former name. So  
 my lab colleague, the late Professor Karen Sp¨arck Jones, got a letter from the  
 university every year asking why she hadn’t published anything (she was down  
 on the payroll as Karen Needham). The publication-tracking system just could  
 not cope with everything the personnel system knew. And as software gets in  
 everything and systems get linked up, conﬂicts can have unexpected remote  
 effects. For example, Karen was also a trustee of the British Library and was  
 not impressed when it started to issue its own admission tickets using the name  
 on the holder’s home university library card. Such issues caused even more

friction when the university introduced an ID card system keyed to payroll  
 names to give uniﬁed access to buildings, libraries and canteens. These issues  
 with multiple names are now mainstream; it’s not just professors, musicians  
 and novelists who use more than one name. Trans people who want to stop  
 ﬁrms using names from a previous gender; women who want to stop using  
 a married name when they separate or divorce, and who perhaps need to if  
 they’re ﬂeeing an abusive partner; people who’ve assumed new names following  
 religious conversion – there’s no end of sources of conﬂict. If you’re building a  
 system that you hope will scale up globally, you’ll eventually have to deal with  
 them all.

Human naming conventions also vary by culture. Chinese may have both

English and Chinese given names if they’re from Hong Kong, with the English  
 one coming before and the Chinese one coming after the family name. Many  
 people in South India, Indonesia and Mongolia have only a single name – a  
 mononym. The Indian convention is to add two initials – for your place of birth  
 and your father’s name. So ‘BK Rajan’ may mean Rajan, son of Kumar, from  
 Bangalore. A common tactic among South Indian migrants to the USA is to  
 use the patronymic (here, Kumar) as a surname; but when western computer  
 systems misinterpret Rajan as a surname, confusion can arise. Russians are  
 known by a forename, a patronymic and a surname. Icelanders have no surname;  
 their given name is followed by a patronymic if they are male and a matronymic  
 if they are female. In the old days, when ‘Maria Trosttad´ottir’ arrived at US  
 immigration and the officer learned that ‘Trosttad´ottir’ isn’t a surname or even  
 a patronymic, their standard practice was to compel her to adopt as a surname  
 a patronymic (say, ‘Carlsson’ if her father was called Carl). Many Indians in the  
 USA have had similar problems, all of which cause unnecessary offence. And  
 then there are cultures where your name changes after you have children.

Another cultural divide is often thought to be that between the English-

speaking countries, where identity cards were unacceptable on privacy grounds10,  
 and the countries conquered by Napoleon or by the Soviets, where identity cards  
 are the norm. What’s less well known is that the British Empire happily imposed  
 ID on many of its subject populations, so the real divide is perhaps whether a  
 country was ever conquered.

The local history of ID conditions all sorts of assumptions. I know Germans

who have refused to believe that a country could function at all without a proper  
 system of population registration and ID cards yet admit they are asked for  
 their ID card only rarely (for example, to open a bank account or get married).  
 Their card number can’t be used as a name because it is a document number

10unless they’re called drivers’ licences or health service cards!

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 256 | Ross Anderson |

*7.4. NAMING*

and changes every time a new card is issued. The Icelandic ID card number,  
 however, is static; it’s just the citizen’s date of birth plus two further digits.  
 What’s more, the law requires that bank account numbers contain the account  
 holder’s ID number. These are perhaps the extremes of private and public ID  
 numbering.

Finally, in many less developed countries, the act of registering citizens and

issuing them with ID is not just inefficient but political [88]. The ruling tribe  
 may seek to disenfranchise the others by making it hard to register births in  
 their territory or by making it inconvenient to get an ID card. Sometimes cards  
 are reissued in the run-up to an election in order to refresh or reinforce the  
 discrimination. Cards can be tied to business permits and welfare payments;  
 delays can be used to extract bribes. Some countries (such as Brazil) have

separate registration systems at the state and federal level, while others (such  
 as Malawi) have left most of their population unregistered. There are many  
 excluded groups, such as refugee children born outside the country of their par-  
 ents’ nationality, and groups made stateless for religious or ideological reasons.  
 Target 16.9 of the United Nations’ Sustainable Development Goals is to ‘provide  
 legal identity for all, including birth registration’; and a number of companies  
 sell ID systems and voting systems ﬁnanced by development aid. These interact  
 with governments in all sorts of complex ways, and there’s a whole research  
 community that studies this [88]. Oh, and if you think this is a third-world  
 problem, there are several US states using onerous registration procedures to  
 make it harder for Black people to vote; and in the Windrush scandal, it emerged  
 that the UK government had deported a number of foreign-born UK residents  
 who were automatically entitled to citizenship as they had not maintained a  
 good enough paper trail of their citizenship to satisfy increasingly xenophobic  
 ministers.

In short, the hidden assumptions about the relationship between govern-

ments and people’s names vary in ways that constrain system design and cause  
 unexpected failures when assumptions are carried across borders. The engineer  
 must always be alert to the fact that a service-oriented ID is one thing and a  
 legal identity or certiﬁcate of citizenship is another. Governments are forever  
 trying to entangle the two, but this leads to all sorts of pain.

**7.4.2.3** **Semantic content of names**

Changing from one type of name to another can be hazardous. A bank got sued  
 after they moved from storing customer data by account number to storing it by  
 name and address. They wrote a program to link up all the accounts operated  
 by each of their customers, in the hope that it would help them target junk mail  
 more accurately. The effect on one customer was serious: the bank statement  
 for the account he kept for his mistress got sent to his wife, who divorced him.

The semantics of names can change over time. In many transport systems,

tickets and toll tags can be bought for cash, which defuses privacy concerns, but  
 it’s more convenient to link them to bank accounts, and these links accumulate  
 over time. The card that UK pensioners use to get free bus travel also started out  
 anonymous, but in practice the bus companies try to link up the card numbers  
 to other passenger identiﬁers. In fact, I once got a hardware store loyalty card

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 257 | Ross Anderson |

*7.4. NAMING*

with a random account number (and no credit checks). I was offered the chance  
 to change this into a bank card after the store was taken over by a supermarket  
 and the supermarket started a bank.

**7.4.2.4** **Uniqueness of names**

Human names evolved when we lived in small communities. We started off with  
 just forenames, but by the late Middle Ages the growth of travel led governments  
 to bully people into adopting surnames. That process took a century or so and  
 was linked with the introduction of paper into Europe as a lower-cost and more  
 tamper-resistant replacement for parchment; paper enabled the badges, seals  
 and other bearer tokens, which people had previously used for road tolls and  
 the like, to be replaced with letters that mentioned their names.

The mass movement of people, business and administration to the Internet

has been too fast for social adaptation. There are now way more people (and  
 systems) online than we’re used to dealing with. So how can we make human-  
 memorable names unique? As we discussed above, Facebook tells one John

Smith from another the way humans do, by clustering each one with his set of  
 friends and adding a photo.

Perhaps the other extreme is cryptographic names. Names are hashes either

of public keys or of other stable attributes of the object being named. All sorts of  
 mechanisms have been proposed to map real-world names, addresses and even  
 document content indelibly and eternally on to the bitstring outputs of hash  
 functions (see, for example, [845]). You can even use hashes of biometrics or the  
 surface microstructure of objects, coupled with a suitable error-correction code.  
 The world of cryptocurrency and blockchains makes much use of hash-based  
 identiﬁers. Such mechanisms can make it impossible to reuse names; as expired  
 domain names are often bought by bad people and exploited, this is sometimes  
 important.

This isn’t entirely new, as it has long been common in transaction process-

ing to just give everything and everyone a number. This can lead to failures,  
 though, if you don’t put enough uniqueness in the right place. For example, a  
 UK bank assigned unique sequence numbers to transactions by printing them  
 on the stationery used to capture the deal. Once, when they wanted to send  
 £20m overseas, the operator typed in £10m by mistake. A second payment  
 of £10m was ordered – but this acquired the same transaction sequence num-  
 ber from the paperwork. So two payments were sent to SWIFT with the same  
 date, payee, amount and sequence number – and the second was discarded as a  
 duplicate [309].

**7.4.2.5** **Stability of names and addresses**

Many names include some kind of address, yet addresses change. While we still  
 had a phone book in Cambridge, about a quarter of the addresses changed every  
 year; with work email, the turnover is probably higher. When we tried in the  
 late 1990s to develop a directory of people who use encrypted email, together  
 with their keys, we found that the main cause of changed entries was changes of

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 258 | Ross Anderson |

*7.4. NAMING*

email address [103]. (Some people had assumed it would be the loss or theft of  
 keys; the contribution from this source was precisely zero.) Things are perhaps  
 more stable now. Most people try to keep their personal mobile phone numbers,  
 so they tend to be long-lived, and the same goes increasingly for personal email  
 addresses. The big service providers like Google and Microsoft generally don’t  
 issue the same email address twice, but other ﬁrms still do.

Distributed systems pioneers considered it a bad thing to put addresses

in names [1353]. But hierarchical naming systems can involve multiple lay-

ers of abstraction with some of the address information at each layer forming  
 part of the name at the layer above. Also, whether a namespace is better

ﬂat depends on the application. Often people end up with different names

at the departmental and organisational level (such as rja14@cam.ac.uk and  
 ross.anderson@cl.cam.ac.uk in my own case). So a clean demarcation be-  
 tween names and addresses is not always possible.

Authorisations have many (but not all) of the properties of addresses. Kent’s

Law tells designers that if a credential contains a list of what it may be used for,  
 then the more things there are on this list the shorter its period of usefulness. A  
 similar problem besets systems where names are composite. For example, some  
 online businesses recognize me by the combination of email address and credit  
 card number. This is clearly bad practice. Quite apart from the fact that I have  
 several email addresses, I have several credit cards.

There are good reasons to use pseudonyms. Until Facebook came along,

people considered it sensible for children and young people to use online names  
 that weren’t easily linkable to their real names and addresses. When you go for  
 your ﬁrst job on leaving college aged 22, or for a CEO’s job at 45, you don’t  
 want a search to turn up all your teenage rants. Many people also change email  
 addresses from time to time to escape spam; I used to give a different email  
 address to every website where I shop. On the other hand, some police and  
 other agencies would prefer people not to use pseudonyms, which takes us into  
 the whole question of traceability online – which I’ll discuss in Part II.

**7.4.2.6** **Restrictions on the use of names**

The interaction between naming and society brings us to a further problem:  
 some names may be used only in restricted circumstances. This may be laid  
 down by law, as with the US social security number and its equivalents in some  
 other countries. Sometimes it is a matter of marketing: a signiﬁcant minority  
 of customers avoid websites that demand too much information.

Restricted naming systems interact in unexpected ways. For example, it’s

fairly common for hospitals to use a patient number as an index to medical  
 record databases, as this may allow researchers to use pseudonymous records  
 for some purposes. This causes problems when a merger of health maintenance  
 organisations, or a policy change, forces the hospital to introduce uniform names.  
 There have long been tussles in Britain’s health service, for example, about  
 which pseudonyms can be used for which purposes.

Finally, when we come to law and policy, the deﬁnition of a name throws

up new and unexpected gotchas. For example, regulations that allow police to

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 259 | Ross Anderson |

*7.4. NAMING*

collect communications data – that is, a record of who called whom and when  
 – are usually much more lax than the regulations governing phone tapping; in  
 many countries, police can get communications data just by asking the phone  
 company. This led to tussles over the status of URLs, which contain data such  
 as the parameters passed to search engines. Clearly some policemen would

like a list of everyone who hit a URL like http://www.google.com/search?q=  
 cannabis+cultivation; just as clearly, many people would consider such large-  
 scale trawling to be an unacceptable invasion of privacy. The resolution in UK  
 law was to deﬁne traffic data as that which was sufficient to identify the machine  
 being communicated with, or in lay language ‘Everything up to the ﬁrst slash.’  
 I discuss this in much more detail later, in the chapter ‘Surveillance or Privacy?’

**7.4.3** **Types of name**

Not only is naming complex at all levels – from the technical up through the  
 organisational to the political – but some of the really wicked issues go across  
 levels. I noted in the introduction that names can refer not just to persons (and  
 machines acting on their behalf), but also to organisations, roles (‘the officer of  
 the watch’), groups, and compound constructions: *principal in role* – Alice as  
 manager; *delegation* – Alice for Bob; *conjunction* – Alice and Bob. Conjunction  
 often expresses implicit access rules: ‘Alice acting as branch manager plus Bob  
 as a member of the group of branch accountants’.

That’s only the beginning. Names also apply to services (such as NFS,

or a public-key infrastructure) and channels (which might mean wires, ports  
 or crypto keys). The same name might refer to different roles: ‘Alice as a

computer game player’ ought to have less privilege than ‘Alice the system ad-  
 ministrator’. The usual abstraction used in the security literature is to treat  
 them as different principals. So there’s no easy mapping between names and  
 principals, especially when people bring their own devices to work or take work  
 devices home, and therefore may have multiple conﬂicting names or roles on the  
 same platform. Many organisations are starting to distinguish carefully between  
 ‘Alice in person’, ‘Alice as a program running on Alice’s home laptop’ and ‘a  
 program running on Alice’s behalf on the corporate cloud’, and we discussed  
 some of the possible mechanisms in the chapter on access control.

Functional tensions are often easier to analyse if you work out how they’re

driven by the underlying business processes. Businesses mainly want to get paid,  
 while governments want to identify people uniquely. In effect, business wants  
 your credit card number while government wants your passport number. An  
 analysis based on incentives can sometimes indicate whether a naming system  
 might be better open or closed, local or global, stateful or stateless – and whether  
 the people who maintain it are the same people who will pay the costs of failure  
 (economics is one of the key issues for dependability,and is the subject of the  
 next chapter).

Finally, although I’ve illustrated many of the problems of naming with re-

spect to people – as that makes the problems more immediate and compelling  
 – many of the same problems pop up in various ways for cryptographic keys,  
 unique product codes, document IDs, ﬁle names, URLs and much more. When  
 we dive into the internals of a modern corporate network we may ﬁnd DNS

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 260 | Ross Anderson |

*7.5. SUMMARY*

Round Robin to multiple machines, each on its own IP addresses, behind a  
 single name; or Anycast to multiple machines, each on the same IP address,  
 behind a single name; or Cisco’s HSRP protocol, where the IP address and the  
 Ethernet MAC address move from one router to another router. (I’ll discuss  
 more technical aspects of network security in Part 2.) Anyway, as systems scale,  
 it becomes less realistic to rely on names that are simple, interchangeable and  
 immutable. You need to scope naming carefully, understand who controls the  
 names on which you rely, work out how slippery they are, and design your  
 system to be dependable despite their limitations.

**7.5** **Summary**

Many secure distributed systems have incurred large costs, or developed seri-  
 ous vulnerabilities, because their designers ignored the basics of how to build  
 (and how not to build) distributed systems. Most of these basics have been in  
 computer science textbooks for a generation.

Many security breaches are concurrency failures of one kind or another;

systems use old data, make updates inconsistently or in the wrong order, or  
 assume that data are consistent when they aren’t or even can’t be. Using time  
 to order transactions may help, but knowing the right time is harder than it  
 seems.

Fault tolerance and failure recovery are critical. Providing the ability to

recover from security failures, as well as from random physical and software  
 failures, is the main purpose of the protection budget for many organisations. At  
 a more technical level, there are signiﬁcant interactions between protection and  
 resilience mechanisms. Byzantine failure – where defective processes conspire  
 rather than failing randomly – is an issue, and it interacts with our choice of  
 cryptographic tools.

There are many different ﬂavors of redundancy, and we have to use the

right combination. We need to protect not just against failures and attempted  
 manipulation, but also against deliberate attempts to deny service that may be  
 part of larger attack plans.

Many problems also arise from trying to make a name do too much, or

making assumptions about it which don’t hold outside of one particular system,  
 culture or jurisdiction. For example, it should be possible to revoke a user’s  
 access to a system by cancelling their user name without getting sued on account  
 of other functions being revoked. The simplest solution is often to assign each  
 principal a unique identiﬁer used for no other purpose, such as a bank account  
 number or a system logon name. But many problems arise when merging two  
 systems that use naming schemes that are incompatible. Sometimes this can  
 even happen by accident.

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 261 | Ross Anderson |

*7.5. SUMMARY*

**Research problems**

I’ve touched on many technical issues in this chapter, from secure time protocols  
 to the complexities of naming. But perhaps the most important research prob-  
 lem is to work out how to design systems that are resilient in the face of malice,  
 that degrade gracefully, and whose security can be recovered simply once the  
 attack is past. All sorts of remedies have been pushed in the past, from get-  
 ting governments to issue everyone with ID to putting it all on the blockchain.  
 However these magic bullets don’t seem to kill any of the goblins.

It’s always a good idea for engineers to study failures; we learn more from

the one bridge that falls down than from the thousand that don’t. We now have  
 a growing number of failed ID systems, such as the UK government’s Verify  
 scheme – an attempt to create a federated logon system for public service that  
 was abandoned in 2019 [1392]. There is a research community that studies

failures of ID systems in less developed countries [88]. And then there’s the  
 failure of blockchains to live up to their initial promise, which I’ll discuss in  
 Part 2 of this book.

Perhaps we need to study more carefully the conditions under which we

can recover neatly from corrupt security state. Malware and phishing attacks  
 mean that at any given time a small (but nonzero) proportion of customer bank  
 accounts are under criminal control. Yet the banking system carries on. The  
 proportion of infected laptops, and phones, varies quite widely by country, and  
 the effects might be worth more careful study.

Classical computer science theory saw convergence in distributed systems as

an essentially technical problem, whose solution depended on technical proper-  
 ties (at one level, atomicity, consistency, isolation and durability; at another,  
 digital signatures, dual control and audit). Perhaps we need a higher-level view  
 in which we ask how we obtain sufficient agreement about the state of the world  
 and incorporate not just technical resilience mechanisms and protection tech-  
 nologies, but also the mechanisms whereby people who have been victims of  
 fraud obtain redress. Purely technical mechanisms that try to obviate the need  
 for robust redress may actually make things worse.

**Further reading**

If the material in this chapter is unfamiliar to you, you may be coming to the sub-  
 ject from a maths/crypto background or chips/engineering or even law/policy.  
 Computer science students get many lectures on distributed systems; to catch  
 up, I’d suggest Saltzer and Kaashoek [1640]. Other books we’ve recommended  
 to our students over the years include Tanenbaum and van Steen [1860] and Mul-  
 lender [1353]. A 2003 report from the US National Research Council, *‘Who Goes*  
 *There? Authentication Through the Lens of Privacy’*, discusses the tradeoffs be-  
 tween authentication and privacy and how they tend to scale poorly [1039].  
 Finally, there’s a recent discussion of naming by Pat Helland [880].

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 262 | Ross Anderson |