

# Chapter 2

## Who is the Opponent?

Going all the way back to early time-sharing systems we systems  
people regarded the users, and any code they wrote, as the mortal  
enemies of us and each other. We were like  
the police force in a violent slum.  
– ROGER NEEDHAM

False face must hide  
what the false heart doth know  
– MACBETH

### 2.1 Introduction

Ideologues may deal with the world as they would wish it to be, but engineers deal with the world as it is. If you’re going to defend systems against attack, you first need to know who your enemies are.

In the early days of computing, we mostly didn’t have real enemies; while banks and the military had to protect their systems, most other people didn’t really bother. The first computer systems were isolated, serving a single company or university. Students might try to hack the system to get more resources and sysadmins would try to stop them, but it was mostly a game. When dial-up connections started to appear, pranksters occasionally guessed passwords and left joke messages, as they’d done at university. The early Internet was a friendly place, inhabited by academics, engineers at tech companies, and a few hobbyists. We knew that malware was possible but almost nobody took it seriously until the late 1980s when PC viruses appeared, followed by the Internet worm in 1988. (Even that was a student experiment that escaped from the lab; I tell the story in section 21.3.2.)

Things changed once everyone started to get online. The mid-1990s saw the first spam, the late 1990s brought the first distributed denial-of-service attack, and the explosion of mail-order business in the dotcom boom introduced credit card fraud. To begin with, online fraud was a cottage industry; the same person

## **2.1. INTRODUCTION**

---

would steal credit card numbers and use them to buy goods which he'd then sell, or make up forged cards to use in a store. Things changed in the mid-2000s with the emergence of underground markets. These let the bad guys specialise – one gang could write malware, another could harvest bank credentials, and yet others could devise ways of cashing out. This enabled them to get good at their jobs, to scale up and to globalise, just as manufacturing did in the late eighteenth century. The 2000s also saw the world's governments putting in the effort to 'Master the Internet' (as the NSA put it) – working out how to collect data at scale and index it, just as Google does, to make it available to analysts. It also saw the emergence of social networks, so that everyone could have a home online – not just geeks with the skills to create their own handcrafted web pages. And of course, once everyone is online, that includes not just the spooks and the crooks but also the jerks, creeps, racists and bullies.

Over the past decade, this threat landscape has stabilised. We also know quite a lot about it. Thanks to Ed Snowden and other whistleblowers, we know a lot about the capabilities and methods of Western intelligence services; we've also learned a lot about China, Russia and other nation-state threat actors. We know a lot about cybercrime; online crime now makes up about half of all crime, by volume and by value. There's a substantial criminal infrastructure based on malware and botnets with which we are constantly struggling; there's also a large ecosystem of scams. Many traditional crimes have gone online, and a typical firm has to worry not just about external fraudsters but also about dishonest insiders. Some firms have to worry about hostile governments, some about other firms, and some about activists. Many people have to deal with online hostility, from kids suffering cyber-bullying at school through harassment of elected politicians to people who are stalked by former partners. And our politics may become more polarised because of the dynamics of online extremism.

One of the first things the security engineer needs to do when tackling a new problem is to identify the likely opponents. Although you can design some specific system components (such as cryptography) to resist all reasonable adversaries, the same is much less true for a complex real-world system. You can't protect it against all possible threats and still expect it to do useful work at a reasonable cost. So what sort of capabilities will the adversaries have, and what motivation? How certain are you of this assessment, and how might it change over the system's lifetime? In this chapter I will classify online and electronic threats depending on motive. First, I'll discuss surveillance, intrusion and manipulation done by governments for reasons of state, ranging from cyber-intelligence to cyber-conflict operations. Second, I'll deal with criminals whose motive is mainly money. Third will be researchers who find vulnerabilities for fun or for money, or who report them out of social conscience – compelling firms to patch their software and clean up their operations. Finally, I'll discuss bad actors whose reasons are personal and who mainly commit crimes against the person, from cyber-bullies to stalkers.

The big service firms, such as Microsoft, Google and Facebook, have to worry about all four classes of threat. Most firms and most private individuals will only be concerned with some of them. But it's important for a security engineer to understand the big picture so you can help clients work out what their own threat model should be, and what sort of attacks they should plan to forestall.

## 2.2 The spooks

Governments have a range of tools for both passive surveillance of networks and active attacks on computer systems. Hundreds of firms sell equipment for wire-tapping, for radio intercept, and for using various vulnerabilities to take over computers, phones and other digital devices. However, there are significant differences among governments in scale, objectives and capabilities. We'll discuss four representative categories – the USA and its allies, China, Russia and the Arab world – from the viewpoint of potential opponents. Even if the spooks aren't in your threat model today, the tools they use will quite often end up in the hands of the crooks too, sooner or later.

### 2.2.1 The Five Eyes

Just as everyone in a certain age range remembers where they were when John Lennon was shot, everyone who's been in our trade since 2013 remembers where they were when they learned of the Snowden revelations on Friday 7th June of that year.

#### 2.2.1.1 Prism

I was in a hotel in Palo Alto, California, reading the Guardian online before a scheduled visit to Google where I'd been as a scientific visitor in 2011, helping develop contactless payments for Android phones. The headline was 'NSA Prism program taps in to user data of Apple, Google and others'; the article, written by Glenn Greenwald and Ewen MacAskill, describes a system called Prism that collects the gmail and other data of users who are not US citizens or permanent residents, and is carried out under an order from the FISA court [817]. After breakfast I drove to the Googleplex, and found that my former colleagues were just as perplexed as I was. They knew nothing about Prism. Neither did the gmail team. How could such a wiretap have been built? Had an order been served on Eric Schmidt, and if so how could he have implemented it without the mail and security teams knowing? As the day went on, people stopped talking.

It turned out that Prism was an internal NSA codename for an access channel that had been provided to the FBI to conduct warranted wiretaps. US law permits US citizens to be wiretapped provided an agency convinces a court to issue a warrant, based on 'probable cause' that they were up to no good; but foreigners could be wiretapped freely. So for a foreign target like me, all an NSA intelligence analyst had to do was click on a tab saying he believed I was a non-US person. The inquiry would be routed automatically via the FBI infrastructure and pipe my Gmail to their workstation. According to the article, this program had started at Microsoft in 2007; Yahoo had fought it in court, but lost, joining in late 2008; Google and Facebook had been added in 2009 and Apple finally in 2012. A system that people thought was providing targeted, warranted wiretaps to law enforcement was providing access at scale for foreign intelligence purposes, and according to a slide deck leaked to the Guardian it

## 2.2. THE SPOOKS

---

was ‘the SIGAD<sup>1</sup> most used in NSA reporting’.

The following day we learned that the source of the story was Edward Snowden, an NSA system administrator who’d decided to blow the whistle. The story was that he’d smuggled over 50,000 classified documents out of a facility in Hawaii on a memory stick and met Guardian journalists in Hong Kong [818]. He tried to fly to Latin America on June 21st to claim asylum, but after the US government cancelled his passport he got stuck in Moscow and eventually got asylum in Russia instead. A consortium of newspapers coordinated a series of stories describing the signals intelligence capabilities of the ‘Five Eyes’ countries – the USA, the UK, Canada, Australia and New Zealand – as well as how these capabilities were not just used but also abused.

The first story based on the leaked documents had actually appeared two days before the Prism story; it was about how the FISA court had ordered Verizon to hand over all call data records (CDRs) to the NSA in February that year [814]. This hadn’t got much attention from security professionals as we knew the agencies did that anyway. But it certainly got the attention of lawyers and politicians, as it broke during the Privacy Law Scholars’ Conference and showed that US Director of National Intelligence James Clapper had lied to Congress when he’d testified that the NSA collects Americans’ domestic communications ‘only inadvertently’. And what was to follow changed everything.

### 2.2.1.2 Tempora

On June 21st, the press ran stories about Tempora, a program to collect intelligence from international fibre optic cables [1199]. This wasn’t a complete surprise; the journalist Duncan Campbell had described a system called Echelon in 1988 which tapped the Intelsat satellite network, keeping voice calls on tape while making metadata available for searching so that analysts could select traffic to or from phone numbers of interest [373, 374] (I’ll give more historical background in section 26.2.6). Snowden gave us an update on the technology. In Cornwall alone, 200 transatlantic fibres were tapped and 46 could be collected at any one time. As each of these carried 10Gb/s, the total data volume could be as high as 21Pb a day, so the incoming data feeds undergo *massive volume reduction*, discarding video, news and the like. Material was then selected using *selectors* – not just phone numbers but more general search terms such as IP addresses – and stored for 30 days in case it turns out to be of interest.

The Tempora program, like Echelon before it, has heavy UK involvement. Britain has physical access to about a quarter of the Internet’s backbone, as modern cables tend to go where phone cables used to, and they were often laid between the same end stations as nineteenth-century telegraph cables. So one of the UK’s major intelligence assets turns out to be the legacy of the communications infrastructure it built to control its nineteenth-century empire. And the asset is indeed significant: by 2012, 300 analysts from GCHQ, and 250 from the NSA, were sifting through the data, using 40,000 and 31,000 selectors respectively to sift 600m ‘telephone events’ each day.

---

<sup>1</sup>SIGINT (Signals Intelligence) Activity Designator

## 2.2. THE SPOOKS

### 2.2.1.3 Muscular

One of the applications running on top of Tempora was Muscular. Revealed on October 30th, this collected data as it flowed between the data centres of large service firms such as Yahoo and Google [2016]. Your mail may have been encrypted using SSL en route to the service's front end, but it then flowed in the clear between each company's data centres. After an NSA PowerPoint slide on 'Google Cloud Exploitation' was published in the Washington Post – see figure 2.1 – the companies scrambled to encrypt everything on their networks. Executives and engineers at cloud service firms took the smiley as a personal affront. It reminded people in the industry that even if you comply with warrants, the spooks will also hack you if they can. It made people outside the industry stop and think: Google had accrued so much access to all our lives via search, mail, maps, calendars and other services that unrestricted intelligence-service access to its records (and to Facebook's and Microsoft's too) was a major privacy breach.

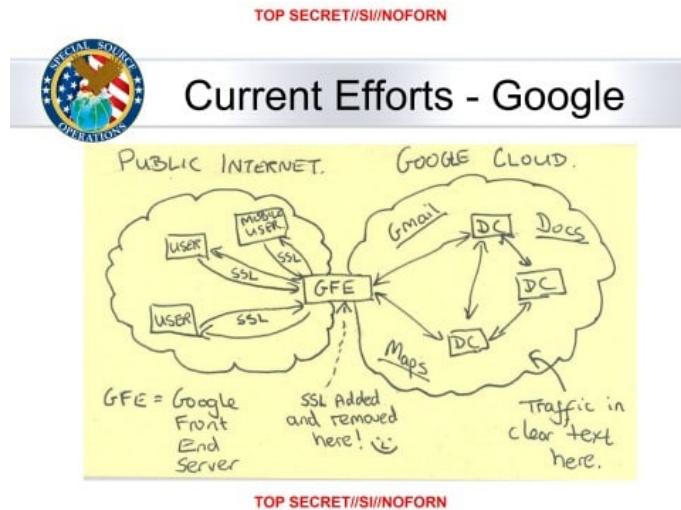


Figure 2.1: Muscular – the slide

Two years later, at a meeting at Princeton which Snowden attended in the form of a telepresence robot, he pointed out that a lot of Internet communications that appear to be encrypted aren't really, as modern websites use content delivery networks (CDNs) such as Akamai and Cloudflare; while the web traffic is encrypted from the user's laptop or phone to the CDN's point of presence at their ISP, it isn't encrypted on the backhaul unless they pay extra – which most of them don't [86]. So the customer thinks the link is encrypted, and it's protected from casual snooping – but not from nation states or firms who can read backbone traffic.

### 2.2.1.4 Special collection

The NSA and CIA jointly operate the Special Collection Service (SCS) whose most visible activity may be the plastic panels near the roofs of US and allied

embassies worldwide; these hide antennas for hoovering up cellular communication (a program known as ‘Stateroom’). Beyond this, SCS implants collection equipment in foreign telcos, Internet exchanges and government facilities. This can involve classical spy tradecraft, from placing bugs that monitor speech or electronic communications, through recruiting moles in target organisations, to the covert deployment of antennas in target countries to tap internal microwave links. Such techniques are not restricted to state targets: Mexican drug cartel leader ‘El Chapo’ Guzman was caught after US agents suborned his system administrator.

Close-access operations include Tempest monitoring: the collection of information leaked by the electromagnetic emissions from computer monitors and other equipment, described in 19.3.2. The Snowden leaks disclose the collection of computer screen data and other electromagnetic emanations from a number of countries’ embassies and UN missions including those of India, Japan, Slovakia and the EU.<sup>2</sup>.

### 2.2.1.5 Bullrun and Edgehill

Special collection increasingly involves supply-chain tampering. SCS routinely intercepts equipment such as routers being exported from the USA, adds surveillance implants, repackages them with factory seals and sends them onward to customers. And an extreme form of supply-chain tampering was when the NSA covertly bought Crypto AG, a Swiss firm that was the main supplier of cryptographic equipment to non-aligned countries during the Cold War; I tell the story in more detail later in section 26.2.7.1.

Bullrun is the NSA codename, and Edgehill the GCHQ one, for ‘crypto enabling’, a \$100m-a-year program of tampering with supplies and suppliers at all levels of the stack. This starts off with attempts to direct, or misdirect, academic research<sup>3</sup>; it continued with placing trusted people on standards committees, and using NIST’s influence to get weak standards adopted. One spectacular incident was the Dual EC\_DRBG debacle, where NIST standardised a random number generator based on elliptic curves that turned out to contain an NSA backdoor. Most of the actual damage, though, was done by restrictions on cryptographic key length, dovetailed with diplomatic pressure on allies to enforce export controls, so that firms needing export licenses could have their arms twisted to use an ‘appropriate’ standard, and was entangled with the Crypto Wars (which I discuss in section 26.2.7). The result was that many of the systems in use today were compelled to use weak cryptography, leading to vulnerabilities in everything from hotel and car door locks to VPNs. In addition to that, supply-chain attacks introduce covert vulnerabilities into widely-used software; many nation states play this game, along with some private actors [890]. We’ll see vulnerabilities that result from surveillance and cryptography policies in one

<sup>2</sup>If the NSA needs to use high-tech collection against you as they can’t get a software implant into your computer, that may be a compliment!

<sup>3</sup>In the 1990s, when I bid to run a research program in coding theory, cryptography and computer security at the Isaac Newton Institute at Cambridge University, a senior official from GCHQ offered the institute a £50,000 donation not to go ahead, saying “There’s nothing interesting happening in cryptography, and Her Majesty’s Government would like this state of affairs to continue”. He was shown the door.

chapter after another, and return in Part 3 of the book to discuss the policy history in more detail.

### 2.2.1.6 Xkeyscore

With such a vast collection of data, you need good tools to search it. The Five Eyes search computer data using Xkeyscore, a distributed database that enables an analyst to search collected data remotely and assemble the results. Exposed on July 31 2013, NSA documents describe it as its “widest-reaching” system for developing intelligence; it enables an analyst to search emails, SMSes, chats, address book entries and browsing histories [815]. Examples in a 2008 training deck include “my target speaks German but is in Pakistan. How can I find him?” “Show me all the encrypted Word documents from Iran” and “Show me all PGP usage in Iran”. By searching for anomalous behaviour, the analyst can find suspects and identify strong selectors (such as email addresses, phone numbers or IP addresses) for more conventional collection.

Xkeyscore is a federated system, where one query scans all sites. Its components buffer information at collection points – in 2008, 700 servers at 150 sites. Some appear to be hacked systems overseas from which the NSA malware can exfiltrate data matching a submitted query. The only judicial approval required is a prompt for the analyst to enter a reason why they believe that one of the parties to the conversation is not resident in the USA. The volumes are such that traffic data are kept for 30 days but content for only 3–5 days. Tasked items are extracted and sent on to whoever requested them, and there’s a notification system (Trafficthief) for tipping off analysts when their targets do anything of interest. Extraction is based either on fingerprints or plugins – the latter allow analysts to respond quickly with detectors for new challenges like steganography and homebrew encryption.

Xkeyscore can also be used for target discovery: one of the training queries is “Show me all the exploitable machines in country X” (machine fingerprints are compiled by a crawler called Mugshot). For example, it came out in 2015 that GCHQ and the NSA hacked the world’s leading provider of SIM cards, the Franco-Dutch company Gemalto, to compromise the keys needed to intercept (and if need be spoof) the traffic from hundreds of millions of mobile phones [1658]. The hack used Xkeyscore to identify the firm’s sysadmins, who were then phished; agents were also able to compromise billing servers to suppress SMS billing and authentication servers to steal keys; another technique was to harvest keys in transit from Gemalto to mobile service providers. According to an interview with Snowden in 2014, Xkeyscore also lets an analyst build a fingerprint of any target’s online activity so that they can be followed automatically round the world. The successes of this system are claimed to include the capture of over 300 terrorists; in one case, Al-Qaida’s Sheikh Atiyatallah blew his cover by googling himself, his various aliases, an associate and the name of his book [1658].

There’s a collection of decks on Xkeyscore with a survey by Morgan Marquis-Boire, Glenn Greenwald and Micah Lee [1230]; a careful reading of the decks

can be a good starting point for exploring the Snowden hoard<sup>4</sup>.

### 2.2.1.7 Longhaul

Bulk key theft and supply-chain tampering are not the only ways to defeat cryptography. The Xkeyscore training deck gives an example: “Show me all the VPN startups in country X, and give me the data so I can decrypt and discover the users”. VPNs appear to be easily defeated; a decryption service called Longhaul ingests ciphertext and returns plaintext. The detailed description of cryptanalytic techniques is held as *Extremely Compartmented Information* (ECI) and is not found in the Snowden papers, but some of them talk of recent breakthroughs in cryptanalysis. What might these be?

The leaks do show diligent collection of the protocol messages used to set up VPN encryption, so some cryptographers suggested in 2015 that some variant of the “Logjam attack” is feasible for a nation-state attacker against the 1024-bit prime used by most VPNs and many TLS connections with Diffie-Hellman key exchange [26]. Others pointed to the involvement of NSA cryptographers in the relevant standard, and a protocol flaw discovered later; yet others pointed out that even with advances in number theory or protocol exploits, the NSA has enough money to simply break 1024-bit Diffie-Hellman by brute force, and this would be easily justified if many people used the same small number of prime moduli – which they do [853]. I’ll discuss cryptanalysis in more detail in Chapter 5.

### 2.2.1.8 Quantum

There is a long history of attacks on protocols, which can be spoofed, replayed and manipulated in various ways. (We’ll discuss this topic in detail in Chapter 4.) The best-documented NSA attack on Internet traffic goes under the codename of Quantum and involves the dynamic exploitation of one of the communication end-points. Thus, to tap an encrypted SSL/TLS session to a webmail provider, the Quantum system fires a ‘shot’ that exploits the browser. There are various flavours; in ‘Quantuminsert’, an injected packet redirects the browser to a ‘Foxacid’ attack server. Other variants attack software updates and the advertising networks whose code runs in mobile phone apps [1995].

### 2.2.1.9 CNE

Computer and Network Exploitation (CNE) is the generic NSA term for hacking, and it can be used for more than just key theft or TLS session hijacking; it can be used to acquire access to traffic too. Operation Socialist was the GCHQ codename for a hack of Belgium’s main telco Belgacom<sup>5</sup>, in 2010–11. GCHQ attackers used Xkeyscore to identify three key Belgacom technical staff, then used Quantuminsert to take over their PCs when they visited sites like LinkedIn. The attackers then used their sysadmin privileges to install malware

---

<sup>4</sup>There’s also a search engine for the collection at <https://www.edwardsnowden.com>.

<sup>5</sup>It is now called Proximus.

## 2.2. THE SPOOKS

---

on dozens of servers, including authentication servers to leverage further access, billing servers so they could cover their tracks, and the company’s core Cisco routers [734]. This gave them access to large quantities of mobile roaming traffic, as Belgacom provides service to many foreign providers when their subscribers roam in Europe. The idea that one NATO and EU member state would conduct a cyber-attack on the critical infrastructure of another took many by surprise. The attack also gave GCHQ access to the phone system in the European Commission and other European institutions. Given that these institutions make many of the laws for the UK and other member states, this was almost as if a US state governor had got his state troopers to hack AT&T so he could wiretap Congress and the White House.

Belgacom engineers started to suspect something was wrong in 2012, and realised they’d been hacked in the spring of 2013; an anti-virus company found sophisticated malware masquerading as Windows files. The story went public in September 2013, and the German news magazine Der Spiegel published Snowden documents showing that GCHQ was responsible. After the Belgian prosecutor reported in February 2018, we learned that the attack must have been authorised by then UK Foreign Secretary William Hague, but there was not enough evidence to prosecute anyone; the investigation had been hampered in all sorts of ways both technical and political; the software started deleting itself within minutes of discovery, and institutions such as Europol (whose head was British) refused to help. The Belgian minister responsible for telecomms, Alexander de Croo, even suggested that Belgium’s own intelligence service might have informally given the operation a green light [735]. Europol later adopted a policy that it will help investigate hacks of ‘suspected criminal origin’; it has nothing to say about hacks by governments.

A GCHQ slide deck on CNE explains that it’s used to support conventional sigint both by redirecting traffic and by “enabling” (breaking) cryptography; that it must always be “UK deniable”; and that it can also be used for “effects”, such as degrading communications or “changing users’ passwords on extremist website” [735]. Other papers show that the agencies frequently target admins of phone companies and ISPs in the Middle East, Africa and indeed worldwide – compromising a key technician is “generally the entry ticket to the network” [1139]. As one phone company executive explained, “The MNOs were clueless at the time about network security. Most networks were open to their suppliers for remote maintenance with an ID and password and the techie in China or India had no clue that their PC had been hacked”.

The hacking tools and methods used by the NSA and its allies are now fairly well understood; some are shared with law enforcement. The Snowden papers reveal an internal store where analysts can get a variety of tools; a series of leaks in 2016–7 by the Shadow Brokers (thought to be Russian military intelligence, the GRU) disclosed a number of actual NSA malware samples, used by hackers at the NSA’s Tailored Access Operations team to launch attacks [238]. (Some of these tools were repurposed by the Russians to launch the NotPetya worm and by the North Koreans in WannaCry, as I’ll discuss later.) The best documentation of all is probably about a separate store of goodies used by the CIA, disclosed in some detail to WikiLeaks in the ‘Vault 7’ leaks in 2017. These include manuals for tools that can be used to install a remote access Trojan

on your machine, with components to geolocate it and to exfiltrate files (including SSH credentials), audio and video; a tool to jump air gaps by infecting thumb drives; a tool for infecting wifi routers so they'll do man-in-the-middle attacks; and even a tool for watermarking documents so a whistleblower who leaks them could be tracked. Many of the tools are available not just for Windows but also for OSX and Android; some infect firmware, making them hard to remove. There are tools for hacking TVs and IoT devices too, and tools to hamper forensic investigations. The Vault 7 documents are useful reading if you're curious about the specifications and manuals for modern government malware [2019]. As an example of the law-enforcement use of such tools, in June 2020 it emerged that the French police in Lille had since 2018 installed malware on thousands of Android phones running EncroChat, an encrypted messaging system favoured by criminals, leading to the arrest of 800 criminal suspects in France, the Netherlands, the UK and elsewhere, as well as the arrest of several police officers for corruption and the seizure of several tons of drugs [1332].

### 2.2.1.10 The analyst's viewpoint

The intelligence analyst thus has a big bag of tools. If they're trying to find the key people in an organisation – whether the policymakers advising on a critical decision, or the lawyers involved in laundering an oligarch's profits – they can use the traffic data in Xkeyscore to map contact networks. There are various neat tools to help, such as 'Cotraveler' which flags up mobile phones that have traveled together. We have some insight into this process from our own research into cybercrime, where we scrape tens of millions of messages from underground forums and analyse them to understand crime types new and old. One might describe the process as 'adaptive message mining'. Just as you use adaptive text mining when you do a web search, and constantly refine your search terms based on samples of what you find, with message mining you also have metadata – so you can follow threads, trace actors across forums, do clustering analysis and use various other tricks to 'find more messages like this one'. The ability to switch back and forth between the detailed view you get from reading individual messages, and the statistical view you get from analysing bulk collections, is extremely powerful.

Once the analyst moves from the hunting phase to the gathering phase, they can use Prism to look at the targets' accounts at Facebook, Google and Microsoft, while Xkeyscore will let them see what websites they visit. Traffic data analysis gives still more: despite the growing use of encryption, the communications to and from a home reveal what app or device is used when and for how long<sup>6</sup>. The agencies are pushing for access to end-to-end messaging systems such as WhatsApp; in countries like the UK, Australia and China, legislators have already authorised this, though it's not at all clear which US companies might comply (I'll discuss policy in Chapter 26).

Given a high-value target, there's a big bag of tools the analyst can install on their laptop or cellphone directly. They can locate it physically, turn it into a room bug and even use it as a remote camera. They can download

---

<sup>6</sup>See for example Hill and Mattu who wiretapped a modern smart home to measure this [900].

## 2.2. THE SPOOKS

---

the target’s address book and contact history and feed that into Xkeyscore to search recursively for their direct and indirect contacts. Meanwhile the analyst can bug messaging apps, beating the end-to-end encryption by collecting the call contents once they’ve been decrypted. They can set up an alarm to notify them whenever the target sends or receives messages of interest, or changes location. The coverage is pretty complete. And when it’s time for the kill, the target’s phone can be used to guide a bomb or a missile. Little wonder Ed Snowden insisted that journalists interviewing him put their phones in the fridge!

Finally, the analyst has also a proxy through which they can access the Internet surreptitiously – typically a machine on a botnet. It might even be the PC in your home office.

### 2.2.1.11 Offensive operations

The Director NSA also heads the US Cyber Command, which since 2009 has been one of ten unified commands of the United States Department of Defense. It is responsible for offensive cyber operations, of which the one that made a real difference was Stuxnet. This was a worm designed to damage Iran’s uranium enrichment centrifuges by speeding them up and slowing them down in patterns designed to cause mechanical damage, and was developed jointly by the USA and Israel [325, 826]. It was technically sophisticated, using four zero-day exploits and two stolen code-signing certificates to spread promiscuously through Windows PCs, until it found Siemens programmable logic controllers of the type used at Iran’s Natanz enrichment plant – where it would then install a rootkit that would issue the destructive commands, while the PC assured the operators that everything was fine. It was apparently introduced using USB drives to bridge the air gap to the Iranian systems, and came to light in 2010 after copies had somehow spread to central Asia and Indonesia. Two other varieties of malware (Flame and Duqu) were then discovered using similar tricks and common code, performing surveillance at a number of companies in the Middle East and South Asia; more recent code-analysis tools have traced a lineage of malware that goes back to 2002 (Flowershop) and continued to operate until 2016 (with the Equation Group tools) [2068].

Stuxnet acted as a wake-up call for other governments, which rushed to acquire ‘cyber-weapons’ and develop offensive cyber *doctrine* – a set of principles for what cyber warriors might do, developed with some thought given to rationale, strategy, tactics and legality. Oh, and the price of zero-day vulnerabilities rose sharply.

### 2.2.1.12 Attack scaling

Computer scientists know the importance of how algorithms scale, and exactly the same holds for attacks. Tapping a single mobile phone is hard. You have to drive around behind the suspect with radio and cryptanalysis gear in your car, risk being spotted, and hope that you manage to catch the suspect’s signal as they roam from one cell to another. Or you can drive behind them with a false

## 2.2. THE SPOOKS

---

base station<sup>7</sup> and hope his phone will roam to it as the signal is louder than the genuine one; but then you risk electronic detection too. Both are highly skilled work and low-yield: you lose the signal maybe a quarter of the time. So if you want to wiretap someone in central Paris often enough, why not just wiretap everyone? Put antennas on your embassy roof, collect it all, write the decrypted calls and text messages into a database, and reconstruct the sessions electronically. If you want to hack everyone in France, hack the telco, perhaps by subverting the equipment it uses. At each stage the capital cost goes up but the marginal cost of each tap goes down. The Five Eyes strategy is essentially to collect everything in the world; it might cost billions to establish and maintain the infrastructure, but once it's there you have everything.

The same applies to offensive cyber operations, which are rather like sabotage. In wartime, you can send commandos to blow up an enemy radar station; but if you do it more than once or twice, your lads will start to run into a lot of sentries. So we scale kinetic attacks differently: by building hundreds of bomber aircraft, or artillery pieces, or (nowadays) thousands of drones. So how do you scale a cyber attack to take down not just one power station, but the opponent's whole power grid? The Five Eyes approach is this. Just as Google keeps a copy of the Internet on a few thousand servers, with all the content and links indexed, US Cyber Command keeps a copy of the Internet that indexes what version of software all the machines in the world are using – the Mugshot system mentioned above – so a Five Eyes cyber warrior can instantly see which targets can be taken over by which exploits.

A key question for competitor states, therefore, is not just to what extent they can create some electronic spaces that are generally off-limits to the Five Eyes. It's the extent to which they can scale up their own intelligence and offensive capabilities rather than having to rely on America. The number of scans and probes that we see online indicates that the NSA are not alone in trying to build cyber weapons that scale. Not all of them might be nation states; some might simply be arms vendors or mercenaries. This raises a host of policy problems to which we'll return in Part 3. For now we'll continue to look at capabilities.

### 2.2.2 China

China is now the leading competitor to the USA, being second not just in terms of GDP but as a technology powerhouse. The Chinese lack the NSA's network of alliances and access to global infrastructure (although they're working hard at that). Within China itself, however, they demand unrestricted access to local data. Some US service firms used to operate there, but trouble followed. After Yahoo's systems were used to trap the dissident Wang Xiaoning in 2002, Alibaba took over Yahoo's China operation in 2005; but there was still a row when Wang's wife sued Yahoo in US courts in 2007, and showed that Yahoo had misled Congress over the matter [1760]. In 2008, it emerged that the version of Skype available in China had been modified so that messages were scanned for sensitive keywords and, if they were found, the user's texts were uploaded to a

---

<sup>7</sup>These devices are known in the USA as a Stingray and in Europe as an IMSI-catcher; they conduct a man-in-the-middle attack of the kind we'll discuss in detail in section 22.2.1.

## 2.2. THE SPOOKS

---

server in China [1959]. In December 2009, Google discovered a Chinese attack on its corporate infrastructure, which became known as Operation Aurora; Chinese agents had hacked into the Google systems used to do wiretaps for the FBI (see Prism above) in order to discover which of their own agents in the USA were under surveillance. Google had already suffered criticism for operating a censored version of their search engine for Chinese users, and a few months later, they pulled out of China. By this time, Facebook, Twitter and YouTube had already been blocked. A Chinese strategy was emerging of total domestic control, augmented by ever-more aggressive collection overseas.

From about 2002, there had been a series of hacking attacks on US and UK defence agencies and contractors, codenamed ‘Titan Rain’ and ascribed to the Chinese armed forces. According to a 2004 study by the US Foreign Military Studies Office (FMSO), Chinese military doctrine sees the country in a state of war with the West; we are continuing the Cold War by attacking China, trying to overthrow its communist regime by exporting subversive ideas to it over the Internet [1881]. Chinese leaders see US service firms, news websites and anonymity tools such as Tor (which the State Department funds so that Chinese and other people can defeat censorship) as being of one fabric with the US surveillance satellites and aircraft that observe their military defences. Yahoo and Google were thus seen as fair game, just like Lockheed Martin and BAe.

Our own group’s first contact with the Chinese came in 2008. We were asked for help by the Dalai Lama, who had realised that the Chinese had hacked his office systems in the run-up to the Beijing Olympics that year. One of my research students, Shishir Nagaraja, happened to be in Delhi waiting for his UK visa to be renewed, so he volunteered to go up to the Tibetan HQ in Dharamsala and run some forensics. He found that about 35 of the 50 PCs in the office of the Tibetan government in exile had been hacked; information was being siphoned off to China, to IP addresses located near the three organs of Chinese state security charged with different aspects of Tibetan affairs. The attackers appear to have got in by sending one of the monks an email that seemed to come from a colleague; when he clicked on the attached PDF, it had a JavaScript buffer overflow that used a vulnerability in Adobe Reader to take over his machine. This technique is called *phishing*, as it works by offering a lure that someone bites on; when it’s aimed at a specific individual (as in this case) it’s called *spear phishing*. They then compromised the Tibetans’ mail server, so that whenever one person in the office sent a .pdf file to another, it would arrive with an embedded attack. The mail server itself was in California.

This is pretty sobering, when you stop to think about it. You get an email from a colleague sitting ten feet away, you ask him if he just sent it – and when he says yes, you click on the attachment. And your machine is suddenly infected by a server that you rent ten thousand miles away in a friendly country. We wrote this up in a tech report on the ‘Snooping Dragon’ [1374]. After it came out, we had to deal for a while with attacks on our equipment, and heckling at conference talks by Chinese people who claimed we had no evidence to attribute the attacks to their government. Colleagues at the Open Net Initiative in Toronto followed through, and eventually found from analysis of the hacking tools’ dashboard that the same espionage network had targeted 1,295 computers in 103 countries [1223]

## 2.2. THE SPOOKS

---

– ranging from the Indian embassy in Washington through Associated Press in New York to the ministries of foreign affairs in Thailand, Iran and Laos.

There followed a series of further reports of Chinese state hacking, from a complex dispute with Rio Tinto in 2009 over the price of iron ore and a hack of the Melbourne International Film festival in the same year when it showed a film about a Uighur leader [1898]. In 2011, the Chinese hacked the CIA’s covert communications system, after the Iranians had traced it, and executed about 30 agents – though that did not become publicly known till later [578]. The first flashbulb moment was a leaked Pentagon report in 2013 that Chinese hackers had stolen some of the secrets of the F35 joint strike fighter, as well as a series of other weapon systems [1379]. Meanwhile China and Hong Kong were amounting for over 80% of all counterfeit goods seized at US ports. The Obama administration vowed to make investigations and prosecutions in the theft of trade secrets a top priority, and the following year five members of the People’s Liberation Army were indicted in absentia.

The White House felt compelled to act once more after the June 2015 news that the Chinese had hacked the Office of Personnel Management (OPM), getting access to highly personal data on 22 million current and former federal employees, ranging from fingerprints to sensitive information from security clearance interviews. Staff applying for Top Secret clearances are ordered to divulge all information that could be used to blackmail them, from teenage drug use to closeted gay relationships. All sexual partners in the past five years have to be declared for a normal Top Secret clearance; for a Strap clearance (to deal with signals intelligence material) the candidate even has to report any foreigners they meet regularly at their church. So this leak affected more than just 22 million people. Officially, this invasive data collection is to mitigate the risk that intelligence agency staff can be blackmailed. (Cynics supposed it was also so that whistleblowers could be discredited.) Whatever the motives, putting all such information in one place was beyond stupid; it was a real ‘database of ruin’. For the Chinese to get all the compromising information on every American with a sensitive government job was jaw-dropping. (Britain screwed up too; in 2008, a navy officer lost a laptop containing the personal data of 600,000 people who had joined the Royal Navy, or tried to [1072].) At a summit in September that year, Presidents Obama and Xi agreed to refrain from computer-enabled theft of intellectual property for commercial gain<sup>8</sup>. Nothing was said in public though about military secrets – or the sex lives of federal agents.

The Chinese attacks of the 2000s used smart people plus simple tools; the attacks on the Tibetans used Russian crimeware as the remote access Trojans. The state also co-opted groups of ‘patriotic hackers’, or perhaps used them for deniability; some analysts noted waves of naïve attacks on western firms that were correlated with Chinese university terms, and wondered whether students had been tasked to hack as coursework. The UK police and security service warned UK firms in 2007. By 2009, multiple Chinese probes had been reported on US electricity firms, and by 2010, Chinese spear-phishing attacks had been

---

<sup>8</sup>The Chinese have kept their promise; according to US firms doing business in China, IP is now sixth on the list of concerns, down from second in 2014 [704]. In any case, the phrase ‘IP theft’ was always a simplification, used to conflate the theft of classified information defence contractors with the larger issue of compelled technology transfer by other firms who wanted access to Chinese markets and the side-issue of counterfeiting.

## 2.2. THE SPOOKS

---

reported on government targets in the USA, Poland and Belgium [1304]. As with the Tibetan attacks, these typically used crude tools and had such poor operational security that it was fairly clear where they came from.

By 2020 the attacks had become more sophisticated, with a series of advanced persistent threats (APTs) tracked by threat intelligence firms. A campaign to hack the phones of Uighurs involved multiple zero-day attacks, even on iPhones, that were delivered via compromised Uighur websites [393]; this targeted not only Uighurs in China but the diaspora too. China also conducts industrial and commercial espionage, and Western agencies claim they exploit managed service providers<sup>9</sup>. Another approach was attacking software supply chains; a Chinese group variously called Wicked Panda or Barium compromised software updates from computer maker Asus, a PC cleanup tool and a Korean remote management tool, as well as three popular computer games, getting its malware installed on millions of machines; rather than launching banking trojans or ransomware, it was then used for spying [810]. Just as in GCHQ’s Operation Socialist, such indirect strategies give a way to scale attacks in territory where you’re not the sovereign. And China was also playing the Socialist game: it came out in 2019 that someone had hacked at least ten western mobile phone companies over the previous seven years and exfiltrated call data records – and that the perpetrators appeared to be the APT10 gang, linked to the Chinese military [2017].

Since 2018 there has been a political row over whether Chinese firms should be permitted to sell routers and 5g network hardware in NATO countries, with the Trump administration blacklisting Huawei in May 2019. There had been a previous spat over another Chinese firm, ZTE; in 2018 GCHQ warned that ZTE equipment “would present risk to UK national security that could not be mitigated effectively or practicably” [1475]<sup>10</sup>. President Trump banned ZTE for breaking sanctions on North Korea and Iran, but relented and allowed its equipment back in the USA subject to security controls<sup>11</sup>.

The security controls route had been tried with Huawei, which set up a centre in Oxfordshire in 2010 where GCHQ could study its software as a condition of the company’s being allowed to sell in the UK. While the analysts did not find any backdoors, their 2019 report surfaced some scathing criticisms of Huawei’s software engineering practices [931]. Huawei had copied a lot of code, couldn’t patch what they didn’t understand, and no progress was being made in tackling many problems despite years of promises. There was an unmanageable number of versions of OpenSSL, including versions that had known vulnerabilities and that were not supported: 70 full copies of 4 different OpenSSL versions, and 304 partial copies of 14 versions. Not only could the Chinese hack the Huawei systems; so could anybody. Their equipment had been excluded for

<sup>9</sup>This became public in 2019 with the claim that they had hacked Wipro and used this to compromise their customers [1093]; but it later emerged that Wipro had been hacked by a crime gang operating for profit.

<sup>10</sup>The only router vendor to have actually been caught with a malicious backdoor in its code is the US company Juniper, which not only used the NSA’s Dual-EC backdoor to make VPN traffic exploitable, but did it in such a clumsy way that others could exploit it too – and at least one other party did so [413].

<sup>11</sup>This was done as a favour to President Xi, according to former National Security Adviser John Bolton, who declared himself ‘appalled’ that the president would interfere in a criminal prosecution [156].

## 2.2. THE SPOOKS

---

some years from UK backbone routers and from systems used for wiretapping. The UK demanded “sustained evidence of improvement across multiple versions and multiple product ranges” before it will put any more trust in it. A number of countries, including Australia and New Zealand, then banned Huawei equipment outright, and in 2019 Canada arrested Huawei’s CFO (who is also its founder’s daughter) following a US request to extradite her for conspiring to defraud global banks about Huawei’s relationship with a company operating in Iran. China retaliated by arresting two Canadians, one a diplomat on leave, on spurious espionage charges, and by sentencing two others to death on drugs charges. The USA hit back with a ban on US suppliers selling chips, software or support to Huawei. The UK banned the purchase of their telecomms equipment from the end of 2020 and said it would remove it from UK networks by 2027. Meanwhile, China is helping many less developed countries modernise their networks, and this access may help them rival the Five Eyes’ scope in due course. Trade policy, industrial policy and cyber-defence strategy have become intertwined in a new Cold War.

Strategically, the question may not be just whether China could use Huawei routers to wiretap other countries at scale, so much as whether they could use it in time of tension to launch DDoS attacks that would break the Internet by subverting BGP routing. I discuss this in more detail in the section 21.2.1. For years, China’s doctrine of ‘Peaceful Rise’ meant avoiding conflict with other major powers until they’re strong enough. The overall posture is one of largely defensive information warfare, combining pervasive surveillance at home, a walled-garden domestic Internet that is better defended against cyber-attack than anyone else’s, plus considerable and growing capabilities, which are mainly used for diligent intelligence-gathering in support of national strategic interests. They are starting to bully other countries in various ways that sometimes involve online operations. In 2016, during a dispute with Vietnam over some islands in the South China Sea, they hacked the airport systems in Hanoi and Ho Chi Minh City, displaying insulting messages and forcing manual check-in for passengers [1195]. In 2020, the EU has denounced China for spreading disruptive fake news about the coronavirus pandemic [1577], and Australia has denounced cyber-attacks that have happened since it called for an international inquiry into the pandemic’s origins [935]. These information operations displayed a first-class overt and covert disinformation capability and followed previous more limited campaigns in Hong Kong and Taiwan [564]. Diplomatic commentators note that China’s trade policy, although aggressive, is no different from Japan’s in the 1970s and not as aggressive as America’s; that the new Cold War is just as misguided and just as likely to be wasteful and dangerous as the last one; that China still upholds the international order more than it disrupts it; and that it upholds it more consistently than the USA has done since WWII [704]. China’s external propaganda aim is to present itself as a positive socio-economic role model for the world, as it competes for access and influence and emerges as a peer competitor to the USA and Europe.

### 2.2.3 Russia

Russia, like China, lacks America's platform advantage and compensates with hacking teams that use spear-phishing and malware. Unlike China, it takes the low road, acting frequently as a spoiler, trying to disrupt the international order, and sometimes benefiting directly via a rise in the price of oil, its main export. The historian Timothy Snyder describes Putin's rise to power and his embrace of oligarchs, orthodox Christianity, homophobia and the fascist ideologue Ivan Ilyin, especially since rigged elections in 2012. This leaves the Russian state in need of perpetual struggle against external enemies who threaten the purity of the Russian people [1798]. Its strategic posture online is different from China's in four ways. First, it's a major centre for cybercrime; underground markets first emerged in Russia and the Ukraine in 2003–5, as we'll discuss in the following section on cybercrime. Second, although Russia is trying to become more closed like China, its domestic Internet is relatively open and intertwined with the West's, including major service firms such as VK and Yandex [605]. Third, Russia's strategy of re-establishing itself as a regional power has been pursued much more aggressively than China's, with direct military interference in neighbours such as Georgia and the Ukraine. These interventions have involved a mixed strategy of cyber-attacks plus 'little green men' – troops without Russian insignia on their uniforms – with a political strategy of denial. Fourth, Russia was humiliated by the USA and Europe when the USSR collapsed in 1989, and still feels encircled. Since about 2005 its goal has been to undermine the USA and the EU, and to promote authoritarianism and nationalism as an alternative to the rules-based international order. This has been pursued more forcefully since 2013; Snyder tells the history [1798]. With Brexit, with the emergence of authoritarian governments in Hungary, Turkey and Poland, and with authoritarians in coalition governments in Italy, Slovakia and Austria, this strategy appears to be winning.

Russian cyber-attacks came to prominence in 2007, after Estonia moved a much-hated Soviet-era statue in Tallinn to a less prominent site, and the Russians felt insulted. DDoS attacks on government offices, banks and media companies forced Estonia to rate-limit its external Internet access for a few weeks [692]. Russia refused to extradite the perpetrators, most of whom were Russian, though one ethnic-Russian Estonian teenager was fined. Sceptics said that the attacks seemed the work of amateurs and worked because the Estonians hadn't hardened their systems the way US service providers do. Estonia nonetheless appealed to NATO for help, and one outcome was the Tallinn Manual, which sets out the law of cyber conflict [1664]. I'll discuss this in more detail in the chapter on electronic and information warfare, in section 23.8. The following year, after the outbreak of a brief war between Russia and Georgia, Russian hackers set up a website with a list of targets in Georgia for Russian patriots to attack [1990].

Estonia and Georgia were little more than warm-ups for the Ukraine. Following demonstrations in Maidan Square in Kiev against pro-Russian President Yanukovich, and an intervention in February 2014 by Russian mercenaries who shot about a hundred demonstrators, Yanukovich fled. The Russians invaded Ukraine on February 24th, annexing Crimea and setting up two puppet states in the Donbass area of eastern Ukraine. Their tactics combined Russian spe-

## 2.2. THE SPOOKS

---

cial forces in plain uniforms, a welter of propaganda claims of an insurgency by Russian-speaking Ukrainians or of Russia helping defend the population against Ukrainian fascists or of defending Russian purity against homosexuals and Jews; all of this coordinated with a variety of cyber-attacks. For example, in May the Russians hacked the website of the Ukrainian election commission and rigged it to display a message that a nationalist who'd received less than 1% of the vote had won; this was spotted and blocked, but Russian media announced the bogus result anyway [1798].

The following year, as the conflict dragged on, Russia took down 30 electricity substations on three different distribution systems within half an hour of each other, leaving 230,000 people without electricity for several hours. They involved multiple different attack vectors that had been implanted over a period of months, and since they followed a Ukrainian attack on power distribution in Crimea – and switched equipment off when they could have destroyed it instead – seemed to have been intended as a warning [2067]. This attack was still tiny compared with the other effects of the conflict, which included the shooting down of a Malaysian Airlines airliner with the loss of all on board; but it was the first cyber-attack to disrupt mains electricity. Finally on June 27 2017 came the NotPetya attack – by far the most damaging cyber-attack to date [813].

The NotPetya worm was initially distributed using the update service for MeDoc, the accounting software used by the great majority of Ukrainian businesses. It then spread laterally in organisations across Windows file-shares using the EternalBlue vulnerability, an NSA exploit with an interesting history. From March 2016, a Chinese gang started using it against targets in Vietnam, Hong Kong and the Philippines, perhaps as a result of finding and reverse engineering it (it's said that you don't launch a cyberweapon; you share it). It was leaked by a gang called the 'Shadow Brokers' in April 2017, along with other NSA software that the Chinese didn't deploy, and then used by the Russians in June. The NotPetya worm used EternalBlue together with the Mimikatz tool that recovers passwords from Windows memory. The worm's payload pretended to be ransomware; it encrypted the infected computer's hard disk and demanded a ransom of \$300 in bitcoin. But there was no mechanism to decrypt the files of computer owners who paid the ransom, so it was really a destructive service-denial worm. The only way to deal with it was to re-install the operating system and restore files from backup.

The NotPetya attack took down banks, telcos and even the radiation monitoring systems at the former Chernobyl nuclear plant. What's more, it spread from the Ukraine to international firms who had offices there. The world's largest container shipping company, Maersk, had to replace most of its computers and compensate customers for late shipments, at a cost of \$300m; FedEx also lost \$300m, and Mondelez \$100m. Mondelez' insurers refused to pay out on the ground that it was an 'Act of War', as the governments of the Ukraine, the USA and the UK all attributed NotPetya to Russian military intelligence, the GRU [1232].

2016 was marked by the Brexit referendum in the UK and the election of President Trump in the USA, in both of which there was substantial Russian interference. In the former, the main intervention was financial support for the leave campaigns, which were later found to have broken the law by spending too

much [1265]; this was backed by intensive campaigning on social media [363]. In the latter, Russian interference was denounced by President Obama during the campaign, leading to renewed economic sanctions, and by the US intelligence community afterwards. An inquiry by former FBI director Robert Mueller found that Russia interfered very widely via the disinformation and social media campaigns run by its Internet Research Agency ‘troll farm’, and by the GRU which hacked the emails of the Democratic national and campaign committees, most notably those of the Clinton campaign chair John Podesta. Some Trump associates went to jail for various offences.

As I’ll discuss in section 26.4.2, it’s hard to assess the effects of such interventions. On the one hand, a report to the US Senate’s Committee on Foreign Relations sets out a story of a persistent Russian policy, since Putin came to power, to undermine the influence of democratic states and the rules-based international order, promoting authoritarian governments of both left and right, and causing trouble where it can. It notes that European countries use broad defensive measures including bipartisan agreements on electoral conduct and raising media literacy among voters; it recommends that these be adopted in the USA as well [385]. On the other hand, Yochai Benkler cautions Democrats against believing that Trump’s election was all Russia’s fault; the roots of popular disaffection with the political elite are much older and deeper [227]. Russia’s information war with the West predates Putin; it continues the old USSR’s strategy of weakening the West by fomenting conflict via a variety of national liberation movements and terrorist groups (I discuss the information-warfare aspects in section 23.8.3). Timothy Snyder places this all in the context of modern Russian history and politics [1798]; his analysis also outlines the playbook for disruptive information warfare against a democracy. It’s not just about hacking substations, but about hacking voters’ minds; about undermining trust in institutions and even in facts, exploiting social media and recasting politics as showbusiness. Putin is a judo player; judo’s about using an opponent’s strength and momentum to trip them up.

### 2.2.4 The rest

The rest of the world’s governments have quite a range of cyber capabilities, but common themes, including the nature and source of their tools. Middle Eastern governments were badly shaken by the Arab Spring uprisings, and some even turned off the Internet for a while, such as Libya in April–July 2010, when rebels were using Google maps to generate target files for US, UK and French warplanes. Since then, Arab states have developed strategies that combine spyware and hacking against high-profile targets, through troll farms pumping out abusive comments in public fora, with physical coercion.

The operations of the United Arab Emirates were described in 2019 by a whistleblower, Lori Stroud [247]. An NSA analyst – and Ed Snowden’s former boss – she was headhunted by a Maryland contractor in 2014 to work in Dubai as a mercenary, but left after the UAE’s operations started to target Americans. The UAE’s main technique was spear-phishing with Windows malware, but their most effective tool, called Karma, enabled them to hack the iPhones of foreign statesmen and local dissidents. They also targeted foreigners critical of

## 2.2. THE SPOOKS

---

the regime. In one case they social-engineered a UK grad student into installing spyware on his PC on the pretext that it would make his communications hard to trace. The intelligence team consisted of several dozen people, both mercenaries and Emiratis, in a large villa in Dubai. The use of iPhone malware by the UAE government was documented by independent observers [1219].

In 2018, the government of Saudi Arabia murdered the Washington Post journalist Jamal Khashoggi in its consulate in Istanbul. The Post campaigned to expose Saudi crown prince Mohammed bin Salman as the man who gave the order, and in January 2019 the National Enquirer published a special edition containing texts showing that the Post's owner Jeff Bezos was having an affair. Bezos pre-empted the Enquirer by announcing that he and his wife were divorcing, and hired an investigator to find the source of the leak. The Enquirer had attempted to blackmail Bezos over some photos it had also obtained; it wanted both him and the investigator to declare that the paper hadn't relied upon 'any form of electronic eavesdropping or hacking in their news-gathering process'. Bezos went public instead. According to the investigator, his iPhone had been hacked by the Saudi Arabian government [199]; the malicious WhatsApp message that did the damage was sent from the phone of the Crown Prince himself [1053]. The US Justice Department later charged two former Twitter employees with spying, by disclosing to the Saudis personal account information of people who criticised their government [1500].

An even more unpleasant example is Syria, where the industrialisation of brutality is a third approach to scaling information collection. Malware attacks on dissidents were reported from 2012, and initially used a variety of spear-phishing lures. As the civil war got underway, police who were arresting suspects would threaten female family members with rape on the spot unless the suspect disclosed his passwords for mail and social media. They would then spear-phish all his contacts while he was being taken away in the van to the torture chamber. This victim-based approach to attack scaling resulted in the compromise of many machines not just in Syria but in America and Europe. The campaigns became steadily more sophisticated as the war evolved, with false-flag attacks, yet retained a brutal edge with some tools displaying beheading videos [737].

Thanks to John Scott-Railton and colleagues at Toronto, we have many further documented examples of online surveillance, computer malware and phone exploits being used to target dissidents; many in Middle Eastern and African countries but also in Mexico and indeed in Hungary [1219]. The real issue here is the ecosystem of companies, mostly in the USA, Europe and Israel, that supply hacking tools to unsavoury states. These tools range from phone malware, though mass-surveillance tools you use on your own network against your own dissidents, to tools that enable you to track and eavesdrop on phones overseas by abusing the signaling system [488]. These tools are used by dictators to track and monitor their enemies in the USA and Europe.

NGOs have made attempts to push back on this cyber arms trade. In one case NGOs argued that the Syrian government's ability to purchase mass-surveillance equipment from the German subsidiary of a UK company should be subject to export control, but the UK authorities were unwilling to block it. GCHQ was determined that if there were going to be bulk surveillance devices on President Assad's network, they should be British devices rather than

## 2.2. THE SPOOKS

---

Ukrainian ones. (I describe this in more detail later in section 26.2.9.) So the ethical issues around conventional arms sales persist in the age of cyber; indeed they can be worse because these tools are used against Americans, Brits and others who are sitting at home but who are unlucky enough to be on the contact list of someone an unpleasant government doesn't like. In the old days, selling weapons to a far-off dictator didn't put your own residents in harm's way; but cyber weapons can have global effects.

Having been isolated for years by sanctions, Iran has developed an indigenous cyber capability, drawing on local hacker forums. Like Syria, its main focus is on intelligence operations, particularly against dissident Iranians, both at home and overseas. It has also been the target of US and other attacks of which the best known was Stuxnet, after which it traced the CIA's covert communications network and rounded up a number of agents [578]. It has launched both espionage operations and attacks of its own overseas. An example of the former was its hack of the Diginotar CA in the Netherlands which enabled it to monitor dissidents' Gmail; while its Shamoon malware damaged thousands of PCs at Aramco, Saudi Arabia's national oil company. The history of Iranian cyber capabilities is told by Collin Anderson and Karim Sadjadpour [49]. Most recently, it attacked Israeli water treatment plants in April 2020; Israel responded the following month with an attack on the Iranian port of Bandar Abbas [229].

Finally, it's worth mentioning North Korea. In 2014, after Sony Pictures started working on a comedy about a plot to assassinate the North Korean leader, a hacker group trashed much of Sony's infrastructure, released embarrassing emails that caused its top film executive Amy Pascal to resign, and leaked some unreleased films. This was followed by threats of terrorist attacks on movie theatres if the comedy were put on general release. The company put the film on limited release, but when President Obama criticised them for giving in to North Korean blackmail, they put it on full release instead.

In 2017, North Korea again came to attention after their WannaCry worm infected over 200,000 computers worldwide, encrypting data and demanding a bitcoin ransom – though like NotPetya it didn't have a means of selective decryption, so was really just a destructive worm. It used the NSA EternalBlue vulnerability, like NotPetya, but was stopped when a malware researcher discovered a kill switch. In the meantime it had disrupted production at carmakers Nissan and Renault and at the Taiwanese chip foundry TSMC, and also caused several hospitals in Britain's National Health Service to close their accident and emergency units. In 2018, the US Department of Justice unsealed an indictment of a North Korean government hacker for both incidents, and also for a series of electronic bank robberies, including of \$81m from the Bank of Bangladesh [1653]. In 2019, North Korean agents were further blamed, in a leaked United Nations report, for the theft of over \$1bn from cryptocurrency exchanges [346].

### 2.2.5 Attribution

It's often said that cyber is different, because attribution is hard. As a general proposition this is untrue; anonymity online is much harder than you think.

### 2.3. CROOKS

---

Even smart people make mistakes in operational security that give them away, and threat intelligence companies have compiled a lot of data that enable them to attribute even false flag operations with reasonable probability in many cases [180]. Yet sometimes it may be true, and people still point to the Climategate affair. Several weeks before the 2009 Copenhagen summit on climate change, someone published over a thousand emails, mostly sent to or from four climate scientists at the University of East Anglia, England. Climate sceptics seized on some of them, which discussed how to best present evidence of global warming, as evidence of a global conspiracy. Official inquiries later established that the emails had been quoted out of context, but the damage had been done. People wonder whether the perpetrator could have been the Russians or the Saudis or even an energy company. However one of the more convincing analyses suggests that it was an internal leak, or even an accident; only one archive file was leaked, and its filename (`FOIA2009.zip`) suggests it may have been prepared for a freedom-of-information disclosure in any case. The really interesting thing here may be how the emails were talked up into a conspiracy theory.

Another possible state action was the Equifax hack. The initial story was that on 8th March 2017, Apache warned of a vulnerability in Apache Struts and issued a patch; two days later, a gang started looking for vulnerable systems; on May 13th, they found that Equifax's dispute portal had not been patched, and got in. The later story, in litigation, was that Equifax had used the default username and password 'admin' for the portal [358]. Either way, the breach had been preventable; the intruders found a plaintext password file giving access to 51 internal database systems, and spent 76 days helping themselves to the personal information of at least 145.5 million Americans before the intrusion was reported on July 29th and access blocked the following day. Executives sold stock before they notified the public on September 7th; Congress was outraged, and the CEO Rick Smith was fired. So far, so ordinary. But no criminal use has been made of any of the stolen information, which led analysts at the time to suspect that the perpetrator was a nation-state actor seeking personal data on Americans at scale [1444]; in due course, four members of the Chinese military were indicted for it [552].

In any case, the worlds of intelligence and crime have long been entangled, and in the cyber age they seem to be getting more so. We turn to cybercrime next.

## 2.3 Crooks

Cybercrime is now about half of all crime, both by volume and by value, at least in developed countries. Whether it is slightly more or less than half depends on definitions (do you include tax fraud now that tax returns are filed online?) and on the questions you ask (do you count harassment and cyber-bullying?) – but even with narrow definitions, it's still almost half. Yet the world's law-enforcement agencies typically spend less than one percent of their budgets on fighting it. Until recently, police forces in most jurisdictions did their best to ignore it; in the USA, it was dismissed as 'identity theft' and counted separately, while in the UK victims were told to complain to their bank instead of the police

### 2.3. CROOKS

---

from 2005–15. The result was that as crime went online, like everything else, the online component wasn’t counted and crime appeared to fall. Eventually, though, the truth emerged in those countries that have started to ask about fraud in regular victimisation surveys<sup>12</sup>.

Colleagues and I run the Cambridge Cybercrime Centre where we collect and curate data for other researchers to use, ranging from spam and phish through malware and botnet command-and-control traffic to collections of posts to underground crime forums. This section draws on a survey we did in 2019 of the costs of cybercrime and how they’ve been changing over time [91].

Computer fraud has been around since the 1960s, a notable early case being the Equity Funding insurance company which from 1964–72 created more than 60,000 bogus policies which it sold to reinsurers, creating a special computer system to keep track of them all. Electronic frauds against payment systems have been around since the 1980s, and spam arrived when the Internet was opened to all in the 1990s. Yet early scams were mostly a cottage industry, where individuals or small groups collected credit card numbers, then forged cards to use in shops, or used card numbers to get mail-order goods. Modern cybercrime can probably be dated to 2003–5 when underground markets emerged that enabled crooks to specialise and get good at their jobs, just as happened in the real economy with the Industrial Revolution.

To make sense of cybercrime, it’s convenient to consider the shared infrastructure first, and then the main types of cybercrime that are conducted for profit. There is a significant overlap with the crimes committed by states that we considered in the last section, and those committed by individuals against other individuals that we’ll consider in the next one; but the actors’ motives are a useful primary filter.

#### 2.3.1 Criminal infrastructure

Since about 2005, the emergence of underground markets has led to people specialising as providers of criminal infrastructure, most notably botnet herders, malware writers, spam senders and cashout operators. I will discuss the technology in much greater detail in section 21.3; in this section my focus is on the actors and the ecosystem in which they operate. Although this ecosystem consists of perhaps a few thousand people with revenues in the tens to low hundreds of millions, they impose costs of many billions on the industry and on society. Now that cybercrime has been industrialised, the majority of ‘jobs’ are now in boring roles such as customer support and system administration, including all the tedious setup work involved in evading law enforcement takedowns [453]. The ‘firms’ they work for specialise; the entrepreneurs and technical specialists can make real money. (What’s more, the cybercrime industry has been booming during the coronavirus pandemic.)

---

<sup>12</sup>The USA, the UK, Australia, Belgium and France

## 2.3. CROOKS

---

### 2.3.1.1 Botnet herders

The first botnets – networks of compromised computers – may have been seen in 1996 with an attack on the ISP Panix in New York, using compromised Unix machines in hospitals to conduct a SYN flood attack [368]. The next use was spam, and by 2000 the Earthlink spammer sent over a million phishing emails; its author was sued by Earthlink. Once cyber-criminals started to get organised, there was a significant scale-up. We started to see professionally built and maintained botnets that could be rented out by bad guys, whether spammers, phishermen or others; by 2007 the Cutwail botnet was sending over 50 million spams a minute from over a million infected machines [1832]. Bots would initially contact a command-and-control server for instructions; these would be taken down, or taken over by threat intelligence companies for use as sinkholes to monitor infected machines, and to feed lists of them to ISPs and corporates.

The spammers' first response was peer-to-peer botnets. In 2007 Storm suddenly grew to account for 8% of all Windows malware; it infected machines mostly by malware in email attachments and had them use the eDonkey peer-to-peer network to find other infected machines. It was used not just for spam but for DDoS, for pump-and-dump stock scams and for harvesting bank credentials. Defenders got lots of peers to join this network to harvest lists of bot addresses, so the bots could be cleaned up, and by late 2008 Storm had been cut to a tenth of the size. It was followed by Kelihos, a similar botnet that also stole bitcoins; its creator, a Russian national, was arrested while on holiday in Spain in 2017 and extradited to the USA where he pled guilty in 2018 [661].

The next criminal innovation arrived with the Conficker botnet: the domain generation algorithm (DGA). Conficker was a worm that spread by exploiting a Windows network service vulnerability; it generated 250 domain names every day, and infected machines would try them all out in the hope that the botmaster had managed to rent one of them. Defenders started out by simply buying up the domains, but a later variant generated 50,000 domains a day and an industry working group made agreements with registrars that these domains would simply be put beyond use. By 2009 Conficker had grown so large, with maybe ten million machines, that it was felt to pose a threat to the largest websites and perhaps even to nation states. As with Storm, its use of randomisation proved to be a two-edged sword; defenders could sit on a subset of the domains and harvest feeds of infected machines. By 2015 the number of infected machines had fallen to under a million.

Regardless of whether something can be done to take out the command-and-control system, whether by arresting the botmaster or by technical tricks, the universal fix for botnet infections is to clean up infected machines. But this raises many issues of scale and incentives. While AV companies make tools available, and Microsoft supplies patches, many people don't use them. So long as your infected PC is merely sending occasional spam but works well enough otherwise, why should you go to the trouble of doing anything? But bandwidth costs ISPs money, so the next step was that some ISPs, particularly the cable companies like Comcast, would identify infected machines and confine their users to a ‘walled garden’ until they promised to clean up. By 2019 that has

### 2.3. CROOKS

---

become less common as people now have all sorts of devices on their wifi, many of which have no user interface; communicating with human users has become harder.

In 2020, we find many botnets with a few tens of thousands of machines that are too small for most defenders to care about, plus some large ones that tend to be multilayer – typically with peer-to-peer mechanisms at the bottom that enable the footsoldier bots to communicate with a few control nodes, which in turn use a domain generation algorithm to find the botmaster. Fragmenting the footsoldiers into a number of small botnets makes it hard for defenders to infiltrate all of them, while the control nodes may be located in places that are hard for defenders to get at. The big money for such botnets in 2020 appears to be in clickfraud.

The latest innovation – since October 2016 – is Mirai, a family of botnets that exploit IoT devices. The first Mirai worm infected CCTV cameras that had been manufactured by Xiaomi and that had a known factory default password that couldn't be changed. Mirai botnets scan the Internet's IPv4 address space for other vulnerable devices which typically get infected within minutes of being powered up. The first attack was on DynDNS and took down Twitter for six hours on the US eastern seaboard. Since then there have been over a thousand variants, which researchers study to determine what's changed and to work out what countermeasures might be used.

At any one time, there may be half a dozen large botnet herders. The Mirai operators, for example, seem to be two or three groups that might have involved a few dozen people.

#### 2.3.1.2 Malware devs

In addition to the several hundred software engineers who write malware for the world's intelligence agencies and their contractors, there may be hundreds of people writing malware for the criminal market; nobody really knows (though we can monitor traffic on hacker forums to guess the order of magnitude).

Within this community there are specialists. Some concentrate on turning vulnerabilities into exploits, a nontrivial task for modern operating systems that use stack canaries, ASLR and other techniques we'll discuss later in section 6.4.1. Others specialise in the remote access Trojans that the exploits install; others build the peer-to-peer and DGA software for resilient command-and-control communications; yet others design specialised payloads for bank fraud. The highest-value operations seem to be platforms that are maintained with constant upgrades to cope with the latest countermeasures from the anti-virus companies. Within each specialist market segment there are typically a handful of operators, so that when we arrest one of them it makes a difference for a while. Some of the providers are based in jurisdictions that don't extradite their nationals, like Russia, and Russian crimeware is used not just by Russian state actors but by others too.

As Android has taken over from Windows as the most frequently used operating system we've seen a rise in Android malware. In China and in countries with a lot of second-hand and older phones, this may be software that uses an

### 2.3. CROOKS

---

unpatched vulnerability to root an Android phone; the USA and Europe have lots of unpatched phones (as many OEMs stop offering patches once a phone is no longer on sale) but it's often just apps that do bad things, such as stealing SMSes used to authenticate banking transactions.

#### 2.3.1.3 Spam senders

Spamming arrived on a small scale when the Internet opened to the public in the mid-1990s, and by 2000 we saw the Earthlink spammer making millions from sending phishing lures. By 2010 spam was costing the world's ISPs and tech companies about \$1bn a year in countermeasures, but it earned its operators perhaps one percent of that. The main beneficiaries may have been webmail services such as Yahoo, Hotmail and Gmail, which can operate better spam filters because of scale; during the 2010s, hundreds of millions of people switched to using their services.

Spam is now a highly specialised business, as getting past modern spam filters requires a whole toolbox of constantly-changing tricks. If you want to use spam to install ransomware, you're better off paying an existing service than trying to learn it all from scratch. Some spam involves industrial-scale email compromise, which can be expensive for the victim; some \$350m was knocked off the \$4.8bn price at which Yahoo was sold to Verizon after a bulk compromise [771].

#### 2.3.1.4 Bulk account compromise

Some botnets are constantly trying to break into email and other online accounts by trying to guess passwords and password recovery questions. A large email service provider might be recovering several tens of thousands of accounts every day. There are peaks, typically when hackers compromise millions of email addresses and passwords at one website and then try them out at all the others. In 2019, this *credential stuffing* still accounts for the largest number of attempted account compromises by volume [1882]. Compromised accounts are sold on to people who exploit them in various ways. Primary email accounts often have recovery information for other accounts, including bank accounts if the attacker is lucky. They can also be used for scams such as the stranded traveler, where the victim emails all their friends saying they've been robbed in some foreign city and asking for urgent financial help to pay the hotel bill. If all else fails, compromised email accounts can be used to send spam.

A variant on the theme is the pay-per-install service, which implants malware on phones or PCs to order and at scale. This can involve a range of phishing lures in a variety of contexts, from free porn sites that ask you to install a special viewer, to sports paraphernalia offers and news about topical events. It can also use more technical means such as drive-by downloads. Such services are often offered by botnets which need them to maintain their own numbers; they might charge third party customers \$10-15 per thousand machines infected in the USA and Europe, and perhaps \$3 for Asia.

## 2.3. CROOKS

---

### 2.3.1.5 Targeted attackers

We've seen the emergence of hack-for-hire operators who will try to compromise a specific target account for a fee, of typically \$750 [1882]. They will investigate the target, make multiple spear-phishing attempts, try password recovery procedures, and see if they can break in through related accounts. This continues a tradition of private eyes who traditionally helped in divorce cases and also stalked celebrities on behalf of red-top newspapers – though with even fewer ethical constraints now that services can be purchased anonymously online. John Scott-Railton and colleagues exposed the workings of Dark Basin, a hack-for-hire company that had targeted critics of ExxonMobil, and also net neutrality advocates, and traced it to a company in India [1692].

In recent years, targeted attacks have also been used at scale against small business owners and the finance staff of larger firms in order to carry out various kinds of payment fraud, as I'll discuss below in 2.3.2.

### 2.3.1.6 Cashout gangs

Back in the twentieth century, people who stole credit card numbers would have to go to the trouble of shopping for goods and then selling them to get money out. Nowadays there are specialists who buy compromised bank credentials on underground markets and exploit them. The prices reveal where the real value lies in the criminal chain; a combination of credit card number and expiry date sells for under a dollar, and to get into the single dollars you need a CVV, the cardholder's name and address, and more.

Cashout techniques change every few years, as paths are discovered through the world's money-laundering controls, and the regulations get tweaked to block them. Some cashout firms organise armies of *mules* to whom they transfer some of the risk. Back in the mid-2000s, mules could be drug users who would go to stores and buy goods with stolen credit cards; then there was a period when unwitting mules were recruited by ads promising large earnings to 'agents' to represent foreign companies but who were used to remit stolen funds through their personal bank accounts. The laundrymen next used Russian banks in Latvia, to which Russian mules would turn up to withdraw cash. Then Liberty Reserve, an unlicensed digital currency based in Costa Rica, was all the rage until it was closed down and its founder arrested in 2013. Bitcoin took over for a while but its popularity with the cybercrime community tailed off as its price became more volatile, as the US Department of the Treasury started arm-twisting bitcoin exchanges into identifying their customers.

As with spam, cashout is a constantly evolving attack-defence game. We monitor it and analyse the trends using CrimeBB, a database we've assembled of tens of millions of posts in underground hacker forums where cybercriminals buy and sell services including cashout [1499]. It also appears to favour gangs who can scale up, until they get big enough to attract serious law-enforcement attention: in 2020, one Sergey Medvedev pleaded guilty to inflicting more than \$568 million in actual losses over the period 2010–15 [1928].

### 2.3. CROOKS

---

#### 2.3.1.7 Ransomware

One reason for the decline in cryptocurrency may have been the growth of ransomware, and as the gangs involved in this switched to payment methods that are easier for victims to use. By 2016–17, 42% of ransomware encountered by US victims demanded prepaid vouchers such as Amazon gift cards; 14% demanded wire transfers and only 12% demanded cryptocurrency; a lot of the low-end ransomware aimed at consumers is now really scareware as it doesn't actually encrypt files at all [1742]. Since 2017, we've seen ransomware-as-a-service platforms; the operators who use these platforms are often amateurs and can't decrypt even if you're willing to pay.

Meanwhile a number of more professional gangs penetrate systems, install ransomware, wait until several days or weeks of backup data have been encrypted and demand substantial sums of bitcoin. This has grown rapidly over 2019–20, with the most high-profile ransomware victims in the USA being public-sector bodies; several hundred local government bodies and a handful of hospitals have suffered service failures [358]. During the pandemic, more hospitals have been targeted; the medical school at UCSF paid over \$1m [1480]. It's an international phenomenon, though, and many private-sector firms fall victim too. Ransomware operators have also been threatening large-scale leaks of personal data to bully victims into paying.

#### 2.3.2 Attacks on banking and payment systems

Attacks on card payment systems started with lost and stolen cards, with forgery at scale arriving in the 1980s; the dotcom boom ramped things up further in the 1990s as many businesses started selling online with little idea of how to detect fraud; and it was card fraud that spawned underground markets in the mid-2000s as criminals sought ways to buy and sell stolen card numbers as well as related equipment and services.

Another significant component is pre-issue fraud, known in the USA as '*identity theft*' [670], where criminals obtain credit cards, loans and other assets in your name and leave you to sort out the mess. I write '*identity theft*' in parentheses as it's really just the old-fashioned offence of impersonation. Back in the twentieth century, if someone went to a bank, pretended to be me, borrowed money from them and vanished, then that was the bank's problem, not mine. In the early twenty-first, banks took to claiming that it's your identity that's been stolen rather than their money [1727]. There is less of that liability dumping now, but the FBI still records much cybercrime as '*identity theft*' which helps keep it out of the mainstream US crime statistics.

The card fraud ecosystem is now fairly stable. Surveys in 2011 and 2019 show that while card fraud doubled over the decade, the loss fell slightly as a percentage of transaction value [90, 91]; the system has been getting more efficient as it grows. Many card numbers are harvested in hacking attacks on retailers, which can be very expensive for them once they've paid to notify affected customers and reimburse banks for reissued cards. As with the criminal infrastructure, the total costs may be easily two orders of magnitude greater than anything the criminals actually get away with.

### **2.3. CROOKS**

---

Attacks on online banking ramped up in 2005 with the arrival of large-scale phishing attacks; emails that seemed to come from banks drove customers to imitation bank websites that stole their passwords. The banks responded with techniques such as two-factor authentication, or the low-cost substitute of asking for only a few letters of the password at a time; the crooks' response, from about 2009, has been credential-stealing malware. Zeus and later Trojans lurk on a PC until the user logs on to a bank whose website they recognise; they then make payments to mule accounts and hide their activity from the user – the so-called 'man-in-the-browser attack'. (Some Trojans even connect in real time to a human operator.) The crooks behind the Zeus and later the Dridex banking malware were named and indicted by US investigators in December 2019, and accused of stealing some \$100m, but they remain at liberty in Russia [795]. Other gangs have been broken up and people arrested for such scams, which continue to net in the hundreds of millions to low billions a year worldwide.

Firms also have to pay attention to business email compromise, where a crook compromises a business email account and tells a customer that their bank account number has changed; or where the crook impersonates the CEO and orders a financial controller to make a payment; and social engineering attacks by people pretending to be from your bank who talk you into releasing a code to authorise a payment. Most targeted attacks on company payment systems can in theory be prevented by the control procedures that most large firms already have, and so the typical target is a badly-run large firm, or a medium-sized firm with enough money to be worth stealing but not enough control to lock everything down.

I'll discuss the technicalities of such frauds in Chapter 12, along with a growing number of crimes that directly affect only banks, their regulators and their retail customers. I'll also discuss cryptocurrencies, which facilitate cybercrimes from ransomware to stock frauds, in Chapter 20.

#### **2.3.3 Sectoral cybercrime ecosystems**

A number of sectors other than banking have their own established cybercrime scenes. One example is travel fraud. There's a whole ecosystem of people who sell fraudulently obtained air tickets, which are sometimes simply bought with stolen credit card numbers, sometimes obtained directly by manipulating or hacking the systems of travel agents or airlines, sometimes booked by corrupt staff at these firms, and sometimes scammed from the public directly by stealing their air miles. The resulting cut-price tickets are sold directly using spam or through various affiliate marketing scams. Some of the passengers who use them to fly know they're dubious, while others are dupes – which makes it hard to deal with the problem just by arresting people at the boarding gate. (The scammers also supply tickets at the last minute, so that the alarms are usually too late.) For an account and analysis of travel fraud, see Hutchings [936]. An increasing number of other business sectors are acquiring their own dark side, and I will touch on some of them in later chapters.

### 2.3.4 Internal attacks

Fraud by insiders has been an issue since businesses started hiring people. Employees cheat the firm, partners cheat each other, and firms cheat their shareholders. The main defence is bookkeeping. The invention of double-entry bookkeeping, of which our earliest records are from the Cairo of a thousand years ago, enabled businesses to scale up beyond the family that owned them. This whole ecosystem is evolving as technology does, and its design is driven by the Big Four accounting firms who make demands on their audit clients that in turn drive the development of accounting software and the supporting security mechanisms. I discuss all this at length in Chapter 12. There are also inside attacks involving whistleblowing, which I discuss below.

### 2.3.5 CEO crimes

Companies attack each other, and their customers too. From the 1990s, printer vendors have used cryptography to lock their customers in to using proprietary ink cartridges, as I describe in section 24.6, while companies selling refills have been breaking the crypto. Games console makers have been playing exactly the same game with aftermarket vendors. The use of cryptography for accessory control is now pervasive, being found even on water filter cartridges in fridges [1071]. Many customers find this annoying and try to circumvent the controls. The US courts decided in the Lexmark v SCC case that this was fine: the printer vendor Lexmark sued SCC, a company that sold clones of its security chips to independent ink vendors, but lost. So the incumbent can now hire the best cryptographers they can find to lock their products, while the challenger can hire the best cryptanalysts they can find to unlock them – and customers can hack them any way they can. Here, the conflict is legal and open. As with state actors, corporates sometimes assemble teams with multiple PhDs, millions of dollars in funding, and capital assets such as electron microscopes<sup>13</sup>. We discuss this in greater detail later in section 24.6.

Not all corporate attacks are conducted as openly. Perhaps the best-known covert hack was by Volkswagen on the EU and US emissions testing schemes; diesel engines sold in cars were programmed to run cleanly if they detected the standard emission test conditions, and efficiently otherwise. For this, the CEO of VW was fired and indicted in the USA (to which Germany won't extradite him), while the CEO of Audi was fired and jailed in Germany [1084]. VW has set aside €25bn to cover criminal and civil fines and compensation. Other carmakers were cheating too; Daimler was fined €860m in Europe in 2019 [1466], and in 2020 reached a US settlement consisting of a fine of \$1.5bn from four government agencies plus a class action of \$700m [1856]. Settlements for other manufacturers and other countries are in the pipeline.

Sometimes products are designed to break whole classes of protection system, an example being the overlay SIM cards described later in Chapter 12. These are SIM cards with two sides and only 160 microns thick, which you stick on top of the SIM card in your phone to provide a second root of trust; they were

---

<sup>13</sup>Full disclosure: both our hardware lab and our NGO activities have on occasion received funding from such actors.

### 2.3. CROOKS

---

designed to enable people in China to defeat the high roaming charges of the early 2010s. The overlay SIM essentially does a man-in-the-middle attack on the real SIM, and can be programmed in Javacard. A side-effect is that such SIMs make it really easy to do some types of bank fraud.

So when putting together the threat model for your system, stop and think what capable motivated opponents you might have among your competitors, or among firms competing with suppliers on which products you depend. The obvious attacks include industrial espionage, but nowadays it's much more complex than that.

#### 2.3.6 Whistleblowers

Intelligence agencies, and secretive firms, can get obsessive about ‘the insider threat’. But in 2018, Barclays Bank’s CEO was fined £642,000 and ordered to repay £500,000 of his bonus for attempting to trace a whistleblower in the bank [698]. So let’s turn it round and look at it from the other perspective – that of the whistleblower. Many are trying to do the right thing, often at a fairly mundane level such as reporting a manager who’s getting bribes from suppliers or who is sexually harassing staff. In regulated industries such as banking they may have a legal duty to report wrongdoing and legal immunity against claims of breach of confidence by their employer. Even then, they often lose because of the power imbalance; they get fired and the problem goes on. Many security engineers think the right countermeasure to leakers is technical, such as data loss prevention systems, but robust mechanisms for staff to report wrongdoing are usually more important. Some organisations, such as banks, police forces and online services, have mechanisms for reporting crimes by staff but no effective process for raising ethical concerns about management decisions<sup>14</sup>.

But even basic whistleblowing mechanisms are often an afterthought; they typically lead the complainant to HR rather than to the board’s audit committee. External mechanisms may be little better. One big service firm ran a “Whistle-blowing hotline” for its clients in 2019; but the web page code has trackers from LinkedIn, Facebook and Google, who could thus identify unhappy staff members, and also JavaScript from CDNs, littered with cookies and referers from yet more IT companies. No technically savvy leaker would use such a service. At the top end of the ecosystem, some newspapers offer ways for whistleblowers to make contact using encrypted email. But the mechanisms tend to be clunky and the web pages that promote them do not always educate potential leakers about either the surveillance risks, or the operational security measures that might counter them. I discuss the usability and support issues around whistleblowing in more detail in Chapter 25.

This is mostly a policy problem rather than a technical one. It’s difficult to design a technical mechanism whereby honest staff can blow the whistle on abuses that have become ingrained in an organisation’s culture, such as pervasive sexual harassment or financial misconduct. In most cases, it’s immediately clear who the whistleblower is, so the critical factor is whether the whistleblower will

---

<sup>14</sup>Google staff ended up going on strike in 2018 about the handling of sexual harassment scandals.

get external support. For example, will they ever get another job? This isn't just a matter of formal legal protection but also of culture. For example, the rape conviction of Harvey Weinstein empowered many women to protest about sexual harassment and discrimination; hopefully the Black Lives Matter protests will similarly empower people of colour [31].

An example where anonymity did help, though, was the UK parliamentary expenses scandal of 2008–9. During a long court case about whether the public could get access to the expense claims of members of parliament, someone went to the PC where the records were kept, copied them to a DVD and sold the lot to the Daily Telegraph. The paper published the juicy bits in instalments all through May and June, when MPs gave up and published the lot on Parliament's website. Half-a-dozen ministers resigned; seven MPs and peers went to prison; dozens of MPs stood down or lost their seats at the following election; and there was both mirth and outrage at some of the things charged to the taxpayer. The whistleblower may have technically committed a crime, but their action was clearly in the public interest; now all parliamentary expenses are public, as they should have been all along. If a nation's lawmakers have their hands in the till, what else will clean up the system?

Even in the case of Ed Snowden, there should have been a robust way for him to report unlawful conduct by the NSA to the appropriate arm of government, probably a Congressional committee. But he knew that a previous whistleblower, Bill Binney, had been arrested and harassed after trying to do that. In hindsight, that aggressive approach was unwise, as President Obama's NSA review group eventually conceded. At the less exalted level of a commercial firm, if one of your staff is stealing your money, and another wants to tell you about it, you'd better make that work.

## 2.4 Geeks

Our third category of attacker are the people like me – researchers who investigate vulnerabilities and report them so they can be fixed. Academics look for new attacks out of curiosity, and get rewarded with professional acclaim – which can lead to promotion for professors and jobs for the students who help us. Researchers working for security companies also look for newsworthy exploits; publicity at conferences such as Black Hat can win new customers. Hobby hackers break into stuff as a challenge, just as people climb mountains or play chess; hacktivists do it to annoy companies they consider to be wicked. Whether on the right side of the law or not, we tend to be curious introverts who need to feel in control, but accept challenges and look for the 'rush'. Our reward is often fame – whether via academic publications, by winning customers for a security consulting business, by winning medals from academic societies or government agencies, or even on social media. Sometimes we break stuff out of irritation, so we can circumvent something that stops us fixing something we own; and sometimes there's an element of altruism. For example, people have come to us in the past complaining that their bank cards had been stolen and used to buy stuff, and the banks wouldn't give them a refund, saying their PIN must have been used, when it hadn't. We looked into some of these cases and discovered

the No-PIN and preplay attacks on chip and PIN systems, which I'll describe in the chapter on banking (the bad guys had actually discovered these attacks, but we replicated them and got justice for some of the victims).

Security researchers who discovered and reported vulnerabilities to a software vendor or system operator used to risk legal threats, as companies sometimes thought this would be cheaper than fixing things. So some researchers took to disclosing bugs anonymously on mailing lists; but this meant that the bad guys could use them at once. By the early 2000s, the IT industry had evolved practices of responsible disclosure whereby researchers disclose the bug to the maintainer some months in advance of disclosure. Many firms operate bug-bounty programs that offer rewards for vulnerabilities; as a result, independent researchers can now make serious money selling vulnerabilities, and more than one assiduous researcher has now earned over \$1m doing this. Since the Stuxnet worm, governments have raced to stockpile vulnerabilities, and we now see some firms that buy vulnerabilities from researchers in order to weaponise them, and sell them to cyber-arms suppliers. Once they're used, they spread, are eventually reverse-engineered and patched. I'll discuss this ecosystem in more detail in the chapters on economics and assurance.

Some more traditional sectors still haven't adopted responsible disclosure. Volkswagen sued researchers in the universities of Birmingham and Nijmegen who reverse-engineered some online car theft tools and documented how poor their remote key entry system was. The company lost, making fools of themselves and publicising the insecurity of their vehicles (I'll discuss the technical details in section 4.3.1 and the policy in section 27.5.7.2). Eventually, as software permeates everything, software industry ways of working will become more widespread too. In the meantime, we can expect turbulence. Firms that cover up problems that harm their customers will have to reckon with the possibility that either an internal whistleblower, or an external security researcher, will figure out what's going on, and when that happens there will often be an established responsible disclosure process to invoke. This will impose costs on firms that fail to align their business models with it.

## 2.5 The Swamp

Our fourth category is abuse, by which we usually mean offences against the person rather than against property. These range from cyber-bullying at schools all the way to state-sponsored Facebook advertising campaigns that get people to swamp legislators with death threats. I'll deal first with offences that scale, including political harassment and child sex abuse material, and then with offences that don't, ranging from school bullying to intimate partner abuse.

### 2.5.1 Hacktivism and hate campaigns

Propaganda and protest evolved as technology did. Ancient societies had to make do with epic poetry; cities enabled people to communicate with hundreds of others directly, by making speeches in the forum; and the invention of writing enabled a further scale-up. The spread of printing in the sixteenth century led

to wars of religion in the seventeenth, daily newspapers in the eighteenth and mass-market newspapers in the nineteenth. Activists learned to compete for attention in the mass media, and honed their skills as radio and then TV came along.

Activism in the Internet age started off with using online media to mobilise people to do conventional lobbying, such as writing to legislators; organisations such as Indymedia and Avaaz developed expertise at this during the 2000s. In 2011, activists such as Wael Ghonim used social media to trigger the Arab Spring, which we discuss in more detail in section 26.4.1. Since then, governments have started to crack down, and activism has spread into online hate campaigns and radicalisation. Many hate campaigns are covertly funded by governments or opposition parties, but by no means all: single-issue campaign groups are also players. If you can motivate hundreds of people to send angry emails or tweets, then a company or individual on the receiving end can have a real problem. Denial-of-service attacks can interrupt operations while doxxing can do real brand damage as well as causing distress to executives and staff.

Activists vary in their goals, in their organisational coherence and in the extent to which they'll break the law. There's a whole spectrum, from the completely law-abiding NGOs who get their supporters to email legislators to the slightly edgy, who may manipulate news by getting bots to click on news stories, to game the media analytics and make editors pay more attention to their issue. Then there are whistleblowers who go to respectable newspapers, political partisans who harass people behind the mild anonymity of Twitter accounts, hackers who break into target firms and vandalise their websites or even doxx them. The Climategate scandal, described in 2.2.5 above, may be an example of doxing by a hacktivist. At the top end, there are the hard-core types who end up in jail for terrorist offences.

During the 1990s, I happily used email and usenet to mobilise people against surveillance bills going through the UK parliament, as I'll describe later in section 26.2.7. I found myself on the receiving end of hacktivism in 2003 when the Animal Liberation Front targeted my university because of plans to build a monkey house, for primates to be used in research. The online component consisted of thousands of emails sent to staff members with distressing images of monkeys with wires in their brains; this was an early example of 'brigading', where hundreds of people gang up on one target online. We dealt with that online attack easily enough by getting their email accounts closed down. But they persisted with physical demonstrations and media harassment; our Vice-Chancellor decided to cut her losses, and the monkey house went to Oxford instead. Some of the leaders were later jailed for terrorism offences after they assaulted staff at a local pharmaceutical testing company and placed bombs under the cars of medical researchers [21].

Online shaming has become popular as a means of protest. It can be quite spontaneous, with a flash mob of vigilantes forming when an incident goes viral. An early example happened in 2005 when a young lady in Seoul failed to clean up after her dog defecated in a subway carriage. Another passenger photographed the incident and put it online; within days the 'dog poo girl' had been hounded into hiding, abandoning her university course [418]. There have been many other cases since.

The power of platforms such as Twitter became evident in Gamergate, a storm sparked by abusive comments about a female game developer made publicly by a former boyfriend in August 2014, and cascading into a torrent of misogynistic criticism of women in the gaming industry and of feminists who had criticised the industry's male-dominated culture. A number of people were doxxed, SWATted, or hounded from their homes [1932]. The harassment was coordinated on anonymous message boards such as 4Chan and the attackers would gang up on a particular target – who then also got criticised by mainstream conservative journalists [1130]. The movement appeared leaderless and evolved constantly, with one continuing theme being a rant against ‘social justice warriors’. It appears to have contributed to the development of the alt-right movement which influenced the 2016 election two years later.

A growing appreciation of the power of angry online mobs is leading politicians to stir them up, at all levels from local politicians trying to undermine their rivals to nation states trying to swing rival states’ elections. Angry mobs are an unpleasant enough feature of modern politics in developed countries; in less developed countries things get even worse, with real lynchings in countries such as India (where the ruling BJP party has been building a troll army since at least 2011 to harass political opponents and civil-society critics [1637]). Companies are targeted less frequently, but it does happen. Meanwhile the social-media companies are under pressure to censor online content, and as it’s hard for an AI program to tell the difference between a joke, abuse, a conspiracy theory and information warfare by a foreign government, they end up having to hire more and more moderators. I will return to the law and policy aspects of this in 26.4 below.

### 2.5.2 Child sex abuse material

When the Internet came to governments’ attention in the 1990s and they wondered how to get a handle on it, the first thing to be regulated was images of child sex abuse (CSA), in the Budapest Convention in 2001. We have little data on the real prevalence of CSA material as the legal restrictions make it hard for anyone outside law enforcement to do any research. In many countries, the approach to CSA material has less focus on actual harm reduction than it deserves. Indeed, many laws around online sexual offences are badly designed, and seem to be driven more by exploiting outrage than by minimising the number of victims and the harm they suffer. CSA may be a case study on how not to do online regulation because of forensic failures, takedown failures, weaponisation and the law-norm gap.

The most notorious forensic failure was Britain’s Operation Ore, which I describe in more detail in 26.5.3. Briefly, several thousand men were arrested on suspicion of CSA offences after their credit card numbers were found on an abuse website, and perhaps half of them turned out to be victims of credit card fraud. Hundreds of innocent men had their lives ruined. Yet nothing was done for the child victims in Brazil and Indonesia, and the authorities are still nowhere near efficient at taking down websites that host CSA material. In most countries, CSA takedown is a monopoly of either the police, or a regulated body that operates under public-sector rules (NCMEC in the USA and the IWF in

the UK), and takes from days to weeks; things would go much more quickly if governments were to use the private-sector contractors that banks use to deal with phishing sites [938]. The public-sector monopoly stems from laws in many countries that make the possession of CSA material a strict-liability offence. This not only makes it hard to deal with such material using the usual abuse channels, but also allows it to be weaponised: protesters can send it to targets and then report them to the police. It also makes it difficult for parents and teachers to deal sensibly with incidents that arise with teens using dating apps or having remote relationships. The whole thing is a mess, caused by legislators wanting to talk tough without understanding the technology. (CSA material is now a significant annoyance for some legislators' staff, and also makes journalists at some newspapers reluctant to make their email addresses public.)

There is an emerging law-norm gap with the growth in popularity of sexting among teenagers. Like it or not, sending intimate photographs to partners (real and intended) became normal behaviour for teens in many countries when smartphones arrived in 2008. This was a mere seven years after the Budapest convention, whose signatories may have failed to imagine that sexual images of under-18s could be anything other than abuse. Thanks to the convention, possessing an intimate photo of anyone under 18 can now result in a prison sentence in any of the 63 countries that have ratified it. Teens laugh at lectures from schoolteachers to not take or share such photos, but the end result is real harm. Kids may be tricked or pressured into sharing photos of themselves, and even if the initial sharing is consensual, the recipient can later use it for blackmail or just pass it round for a laugh. Recipients – even if innocent – are also committing criminal offences by simply having the photos on their phones, so kids can set up other kids and denounce them. This leads to general issues of bullying and more specific issues of intimate partner abuse.

### 2.5.3 School and workplace bullying

Online harassment and bullying are a fact of life in modern societies, not just in schools but in workplaces too, as people jostle for rank, mates and resources. From the media stories of teens who kill themselves following online abuse, you might think that cyber-bullying now accounts for most of the problem – at least at school – but the figures show that it's less than half. An annual UK survey discloses that about a quarter of children and young people are constantly bullied (13% verbal, 5% cyber and 3% physical) while about half are bullied sometimes (24%, 8% and 9% respectively) [565]. The only national survey of all ages of which I'm aware is the French national victimisation survey, which since 2007 has collected data not just on physical crimes such as burglary and online crimes such as fraud, but on harassment too [1458]. This is based on face-to-face interviews with 16,000 households and the 2017 survey reported two million cases of threatening behaviour, 7% were made on social networks and a further 9% by phone. But have social media made this worse? Research suggests that the effects of social media use on adolescent well-being are nuanced, small at best, and contingent on analytic methods [1473].

Yet there is talk in the media of a rise in teen suicide which some commentators link to social media use. Thankfully, the OECD mortality statistics show

that this is also untrue: suicides among 15–19 year olds have declined slightly from about 8 to about 7 cases per 100,000 over the period 1990–2015 [1477].

#### 2.5.4 Intimate relationship abuse

Just as I ended the last section by discussing whistleblowers – the insider threat to companies – I’ll end this section with intimate relationship abuse, the insider threat to families and individuals. Gamergate may have been a flashbulb example, but protection from former intimate partners and other family members is a real problem that exists at scale – with about half of all marriages ending in divorce, and not all breakups being amicable. Intimate partner abuse has been suffered by 27% of women and 11% of men. Stalking is not of course limited to former partners. Celebrities in particular can be stalked by people they’ve never met – with occasional tragic outcomes, as in the case of John Lennon. But former partners account for most of it, and law enforcement in most countries have historically been reluctant to do anything effective about them. Technology has made the victims’ plight worse.

One subproblem is the publication of non-consensual intimate imagery (NCII), once called ‘revenge porn’ – until California Attorney General Kamala Harris objected that this is cyber-exploitation and a crime. Her message got through to the big service firms who since 2015 have been taking down such material on demand from the victims [1690]. This followed an earlier report in 2012 where Harris documented the increasing use of smartphones, online marketplaces and social media in forcing vulnerable people into unregulated work including prostitution – raising broader questions about how technology can be used to connect with, and assist, crime victims [866].

The problems faced by a woman leaving an abusive and controlling husband are among the hardest in the universe of information security. All the usual advice is the wrong way round: your opponent knows not just your passwords but has such deep contextual knowledge that he can answer all your password recovery questions. There are typically three phases: a physical control phase where the abuser has access to your device and may install malware, or even destroy devices; a high-risk escape phase as you try to find a new home, a job and so on; and a life-apart phase when you might want to shield location, email address and phone numbers to escape harassment, and may have lifelong concerns. It takes seven escape attempts on average to get to life apart, and disconnecting from online services can cause other abuse to escalate. After escape, you may have to restrict childrens’ online activities and sever mutual relationships; letting your child post anything can leak the school location and lead to the abuser turning up. You may have to change career as it can be impossible to work as a self-employed professional if you can no longer advertise.

To support such users, responsible designers should think hard about usability during times of high stress and high risk; they should allow users to have multiple accounts; they should design things so that someone reviewing your history should not be able to tell you deleted anything; they should push two-factor authentication, unusual activity notifications, and incognito mode. They should also think about how a survivor can capture evidence for use in divorce and custody cases and possibly in criminal prosecution, while minimising the

## 2.6. SUMMARY

---

trauma [1248]. But that's not what we find in real life. Many banks don't really want to know about disputes or financial exploitation within families. A big problem in some countries is stalkerware – apps designed to monitor partners, ex-partners, children or employees. A report from Citizen Lab spells out the poor information security practices of these apps, how they are marketed explicitly to abusive men, and how they break the law in Europe and Canada; as for the USA and Australia, over half of abusers tracked women using stalkerware [1495]. And then there's the Absher app, which enables men in Saudi Arabia to control their women in ways unacceptable in developed countries; its availability in app stores has led to protests against Apple and Google elsewhere in the world, but as of 2020 it's still there.

Intimate abuse is hard for designers and others to deal with as it's entangled with normal human caregiving between partners, between friends and colleagues, between parents and young children, and later between children and elderly parents. Many relationships are largely beneficent but with some abusive aspects, and participants often don't agree on which aspects. The best analysis I know, by Karen Levy and Bruce Schneier, discusses the combination of multiple motivations, copresence which leads to technical vulnerabilities, and power dynamics leading to relational vulnerabilities [1154]. Technology facilitates multiple privacy invasions in relationships, ranging from the casual to serious abuse; designers need to be aware that households are not units, devices are not personal, and the purchaser of a device is not the only user. I expect that concerns about intimate abuse will expand in the next few years to concerns about victims of abuse by friends, teachers and parents, and will be made ever more complex by new forms of home and school automation.

## 2.6 Summary

The systems you build or operate can be attacked by a wide range of opponents. It's important to work out who might attack you and how, and it's also important to be able to figure out how you were attacked and by whom. Your systems can also be used to attack others, and if you don't think about this in advance you may find yourself in serious legal or political trouble.

In this chapter I've grouped adversaries under four themes: the spooks, the crooks, the hackers and the swamp. Not all threat actors are bad: many hackers report bugs responsibly and many whistleblowers are public-spirited. ('Our' spooks are of course considered good while 'theirs' are bad; moral valence depends on the public and private interests in play.) Intelligence and law enforcement agencies may use a mix of traffic data analysis and content sampling when hunting, and targeted collection for gathering; collection methods range from legal coercion via malware to deception. Both spooks and crooks use malware to establish botnets as infrastructure. Crooks typically use opportunistic collection for mass attacks, while for targeted work, spear-phishing is the weapon of choice; intelligence agencies may have fancier tools but use the same basic methods. There are also cybercrime ecosystems attached to specific business sectors; basically, crime will evolve where it can scale. As for the swamp, the weapon of choice is the angry mob, wielded nowadays by states, activist groups

and even individual orators. There are many ways in which abuse can scale, and when designing a system you need to work out how crimes against it, or abuse using it, might scale. It's not enough to think about usability; you need to think about abusability too.

Personal abuse matters too. Every police officer knows that the person who assaults you or murders you isn't usually a stranger, but someone you know – maybe another boy in your school class, or your stepfather. This has been ignored by the security research community, perhaps because we're mostly clever white or Asian boys from stable families in good neighbourhoods.

If you're defending a company of any size, you'll see enough machines on your network getting infected, and you need to know whether they're just zombies on a botnet or part of a targeted attack. So it's not enough to rely on patching and antivirus. You need to watch your network and keep good enough logs that when an infected machine is spotted you can tell whether it's a kid building a botnet or a targeted attacker who responds to loss of a viewpoint with a scramble to develop another one. You need to make plans to respond to incidents, so you know who to call for forensics – and so your CEO isn't left gasping like a landed fish in front of the TV cameras. You need to think systematically about your essential controls: backup to recover from ransomware, payment procedures to block business email compromise, and so on. If you're advising a large company they should have much of this already, and if it's a small company you need to help them figure out how to do enough of it.

The rest of this book will fill in the details.

## 2.7 Research problems

Until recently, research on cybercrime wasn't really scientific. Someone would get some data – often under NDA from an anti-virus company – work out some statistics, write up their thesis, and then go get a job. The data were never available to anyone else who wanted to check their results or try a new type of analysis. Since 2015 we've been trying to fix that by setting up the Cambridge Cybercrime Centre, where we collect masses of data on spam, phish, botnets and malware as a shared resource for researchers. We're delighted for other academics to use it. If you want to do research on cybercrime, call us.

We also need something similar for espionage and cyber warfare. People trying to implant malware into control systems and other operational technology are quite likely to be either state actors, or cyber-arms vendors who sell to states. The criticisms made by President Eisenhower of the 'military-industrial complex' apply here in spades. Yet not one of the legacy think-tanks seems interested in tracking what's going on. As a result, nations are more likely to make strategic miscalculations, which could lead not just to cyber-conflict but the real kinetic variety, too.

As for research into cyber abuse, there is now some research, but the technologists, the psychologists, the criminologists and the political scientists aren't talking to each other enough. There are many issues, from the welfare and rights of children and young people to our ability to hold fair and free elections. We

need to engage more technologists with public-policy issues and educate more policy people about the realities of technology. We also need to get more women involved, and people from poor communities in both developed and less developed countries, so we have a less narrow perspective on what the real problems are.

## 2.8 Further Reading

There's an enormous literature on the topics discussed in this chapter but it's rather fragmented. A starting point for the Snowden revelations might be Glen Greenwald's book '*No Place to Hide*' [816]; for an account of Russian strategy and tactics, see the 2018 report to the US Senate's Committee on Foreign Relations [385]; and for a great introduction to the history of propaganda see Tim Wu's '*The Attention Merchants*' [2050]. For surveys of cybercrime, see our 2012 paper "Measuring the Cost of Cybercrime" [90] and our 2019 follow-up "Measuring the Changing Cost of Cybercrime" [91]. Criminologists such as Bill Chambliss have studied state-organised crime, from piracy and slavery in previous centuries through the more recent smuggling of drugs and weapons by intelligence agencies to torture and assassination; this gives the broader context within which to assess unlawful surveillance. The story of Gamergate is told in Zoë Quinn's '*Crash Override*' [1567]. Finally, the tale of Marcus Hutchings, the malware expert who stopped WannaCry, is at [811].

## Chapter 3

# Psychology and Usability

Humans are incapable of securely storing high-quality cryptographic keys, and they have unacceptable speed and accuracy when performing cryptographic operations. (They are also large, expensive to maintain, difficult to manage, and they pollute the environment. It is astonishing that these devices continue to be manufactured and deployed. But they are sufficiently pervasive that we must design our protocols around their limitations.)  
– KAUFMANN, PERLMAN AND SPECINER [1025]

Only amateurs attack machines; professionals target people.  
– BRUCE SCHNEIER

Metternich told lies all the time, and never deceived any one;  
Talleyrand never told a lie and deceived the whole world.  
– THOMAS MACAULAY

### 3.1 Introduction

Many real attacks exploit psychology at least as much as technology. We saw in the last chapter how some online crimes involve the manipulation of angry mobs, while both property crimes and espionage make heavy use of *phishing*, in which victims are lured by an email to log on to a website that appears genuine but that's actually designed to steal their passwords or get them to install malware.

Online frauds like phishing are often easier to do, and harder to stop, than similar real-world frauds because many online protection mechanisms are neither as easy to use nor as difficult to forge as their real-world equivalents. It's much easier for crooks to create a bogus bank website that passes casual inspection than to build an actual bogus bank branch in a shopping street.

We've evolved social and psychological tools over millions of years to help us deal with deception in face-to-face contexts, but these are less effective when we get an email that asks us to do something. For an ideal technology, good use

### **3.2. INSIGHTS FROM PSYCHOLOGY RESEARCH**

---

would be easier than bad use. We have many examples in the physical world: a potato peeler is easier to use for peeling potatoes than a knife is, but a lot harder to use for murder. But we've not always got this right for computer systems yet. Much of the asymmetry between good and bad on which we rely in our daily business doesn't just depend on formal exchanges – which can be automated easily – but on some combination of physical objects, judgment of people, and the supporting social protocols. So, as our relationships with employers, banks and government become more formalised via online communication, and we lose both physical and human context, the forgery of these communications becomes more of a risk.

Deception, of various kinds, is now the principal mechanism used to defeat online security. It can be used to get passwords, to compromise confidential information or to manipulate financial transactions directly. Hoaxes and frauds have always happened, but the Internet makes some of them easier, and lets others be repackaged in ways that may bypass our existing controls (be they personal intuitions, company procedures or even laws).

Another driver for the surge in attacks based on social engineering is that people are getting better at technology. As designers learn how to forestall the easier technical attacks, psychological manipulation of system users or operators becomes ever more attractive. So the security engineer absolutely must understand basic psychology, as a prerequisite for dealing competently with everything from passwords to CAPTCHAs and from phishing to social engineering in general; a working appreciation of risk misperception and scaremongering is also necessary to understand the mechanisms underlying angry online mobs and the societal response to emergencies from terrorism to pandemic disease. So just as research in security economics led to a real shift in perspective between the first and second editions of this book, research in security psychology has made much of the difference to how we view the world between the second edition and this one.

In the rest of this chapter, I'll first survey relevant research in psychology, then work through how we apply the principles to make password authentication mechanisms more robust against attack, to security usability more generally, and beyond that to good design.

## **3.2 Insights from psychology research**

Psychology is a huge subject, ranging from neuroscience through to clinical topics, and spilling over into cognate disciplines from philosophy through artificial intelligence to sociology. Although it has been studied for much longer than computer science, our understanding of the mind is much less complete: the brain is so much more complex. There's one central problem – the nature of consciousness – that we just don't understand at all. We know that 'the mind is what the brain does', yet the mechanisms that underlie our sense of self and of personal history remain obscure.

Nonetheless a huge amount is known about the functioning of the mind and the brain, and we're learning interesting new things all the time. In what

follows I can only offer a helicopter tour of three of the themes in psychology research that are very relevant to our trade: cognitive psychology, which studies topics such as how we remember and what sort of mistakes we make; social psychology, which deals with how we relate to others in groups and to authority; and behavioral economics, which studies the heuristics and biases that lead us to make decisions that are consistently irrational in measurable and exploitable ways.

#### **3.2.1 Cognitive psychology**

Cognitive psychology is the classical approach to the subject – building on early empirical work in the nineteenth century. It deals with how we think, remember, make decisions and even daydream. Twentieth-century pioneers such as Ulric Neisser discovered that human memory doesn't work like a video recorder: our memories are stored in networks across the brain, from which they are reconstructed, so they change over time and can be manipulated [1427]. There are many well-known results. For example, it's easier to memorise things that are repeated frequently, and it's easier to store things in context. Many of these insights are used by marketers and scammers, but misunderstood or just ignored by most system developers.

For example, most of us have heard of George Miller's result that human short-term memory can cope with about seven (plus or minus two) simultaneous choices [1317] and, as a result, many designers limit menu choices to about five. But this is not the right conclusion. People search for information first by recalling where to look, and then by scanning; once you've found the relevant menu, scanning ten items is only twice as hard as scanning five. The real limits on menu size are screen size, which might give you ten choices, and with spoken menus, where the average user has difficulty dealing with more than three or four [1544]. Here, too, Miller's insight is misused because spatio-structural memory is a different faculty from echoic memory. This illustrates why a broad idea like  $7+/-2$  can be hazardous; you need to look at the detail.

In recent years, the centre of gravity in this field has been shifting from applied cognitive psychology to the human-computer interaction (HCI) research community, because of the huge amount of empirical know-how gained not just from lab experiments, but from the iterative improvement of fielded systems. As a result, HCI researchers not only model and measure human performance, including perception, motor control, memory and problem-solving; they have also developed an understanding of how users' mental models of systems work, how they differ from developers' mental models, and of the techniques (such as task analysis and cognitive walkthrough) that we can use to explore how people learn to use and understand systems.

Security researchers need to find ways of turning these ploughshares into swords (the bad guys are already working on it). There are some low-hanging fruit; for example, the safety research community has put a lot of effort into studying the errors people make when operating equipment [1589]. It's said that 'to err is human' and error research confirms this: the predictable varieties of human error are rooted in the very nature of cognition. The schemata, or mental models, that enable us to recognise people, sounds and concepts so

### **3.2. INSIGHTS FROM PSYCHOLOGY RESEARCH**

---

much better than computers, also make us vulnerable when the wrong model gets activated.

Human errors made while operating equipment fall into broadly three categories, depending on where they occur in the ‘stack’: slips and lapses at the level of skill, mistakes at the level of rules, and misconceptions at the cognitive level.

- Actions performed often become a matter of skill, but we can slip when a manual skill fails – for example, pressing the wrong button – and we can also have a lapse where we use the wrong skill. For example, when you intend to go to the supermarket on the way home from work you may take the road home by mistake, if that’s what you do most days (this is also known as a *capture error*). Slips are exploited by typosquatters, who register domains similar to popular ones, and harvest people who make typing errors; other attacks exploit the fact that people are trained to click ‘OK’ to pop-up boxes to get their work done. So when designing a system you need to ensure that dangerous actions, such as installing software, require action sequences that are quite different from routine ones. Errors also commonly follow interruptions and perceptual confusion. One example is the *post-completion error*: once they’ve accomplished their immediate goal, people are easily distracted from tidying-up actions. More people leave cards behind in ATMs that give them the money first and the card back second.
- Actions that people take by following rules are open to errors when they follow the wrong rule. Various circumstances – such as information overload – can cause people to follow the strongest rule they know, or the most general rule, rather than the best one. Phishermen use many tricks to get people to follow the wrong rule, ranging from using `https` (because ‘it’s secure’) to starting URLs with the impersonated bank’s name, as `www.citibank.secureauthentication.com` – for most people, looking for a name is a stronger rule than parsing its position.
- The third category of mistakes are those made by people for cognitive reasons – either they simply don’t understand the problem, or pretend that they do, and ignore advice in order to get their work done. The seminal paper on security usability, Alma Whitten and Doug Tygar’s “Why Johnny Can’t Encrypt”, demonstrated that the encryption program PGP was simply too hard for most college students to use as they didn’t understand the subtleties of private versus public keys, encryption and signatures [2018]. And there’s growing realisation that many security bugs occur because most programmers can’t use security mechanisms either. Both access control mechanisms and security APIs are hard to understand and fiddly to use; security testing tools are often not much better. Programs often appear to work even when protection mechanisms are used in quite mistaken ways. Engineers then copy code from each other, and from online code-sharing sites, so misconceptions and errors are propagated widely [11]. They often know this is bad, but there’s just not the time to do better.

### **3.2. INSIGHTS FROM PSYCHOLOGY RESEARCH**

---

There is some important science behind all this, and here are just two examples. James Gibson developed the concept of action possibilities or *affordances*: the physical environment may be climbable or fall-off-able or get-under-able for an animal, and similarly a seat is sit-on-able. People have developed great skill at creating environments that induce others to behave in certain ways: we build stairways and doorways, we make objects portable or graspable; we make pens and swords [762]. Often perceptions are made up of affordances, which can be more fundamental than value or meaning. In exactly the same way, we design software artefacts to train and condition our users' choices, so the affordances of the systems we use can affect how we think in all sorts of ways. We can also design traps for the unwary: an animal that mistakes a pitfall for solid ground is in trouble.

Gibson also came up with the idea of optical flows, further developed by Christopher Longuet-Higgins [1185]. As our eyes move relative to the environment, the resulting *optical flow field* lets us interpret the image, understanding the size, distance and motion of objects in it. There is an elegant mathematical theory of optical parallax, but our eyes deal with it differently: they contain receptions for specific aspects of this flow field which assume that objects in it are rigid, which then enables us to resolve rotational and translational components. Optical flows enable us to understand the shapes of objects around us, independently of binocular vision. We use them for some critical tasks such as landing an aeroplane and driving a car.

In short, cognitive science gives useful insights into how to design system interfaces so as to make certain courses of action easy, hard or impossible. It is increasingly tied up with research into computer human interaction. You can make mistakes more or less likely by making them easy or difficult; in section 28.2.2 I give real examples of usability failures causing serious accidents involving both medical devices and aircraft. Yet security can be even harder than safety if we have a sentient attacker who can provoke exploitable errors.

What can the defender expect attackers to do? They will use errors whose effect is predictable, such as capture errors; they will exploit perverse affordances; they will disrupt the flows on which safe operation relies; and they will look for, or create, exploitable dissonances between users' mental models of a system and its actual logic. To look for these, you should try a cognitive walkthrough aimed at identifying attack points, just as a code walkthrough can be used to search for software vulnerabilities. Attackers also learn by experiment and share techniques with each other, and develop tools to look efficiently for known attacks. So it's important to be aware of the attacks that have already worked. (That's one of the functions of this book.)

#### **3.2.2 Gender, diversity and interpersonal variation**

Many women die because medical tests and technology assume that patients are men, or because engineers use male crash-test dummies when designing cars; protective equipment, from sportswear through stab-vests to spacesuits, gets tailored for men by default [498]. So do we have problems with information systems too? They are designed by men, and young geeky men at that, yet over half their users may be women. This realisation has led to research on

### **3.2. INSIGHTS FROM PSYCHOLOGY RESEARCH**

---

*gender HCI* – on how software should be designed so that women can also use it effectively. Early experiments started from the study of behaviour: experiments showed that women use peripheral vision more, and it duly turned out that larger displays reduce gender bias. Work on American female programmers suggested that they tinker less than males, but more effectively [202]. But how much is nature, and how much nurture? Societal factors matter, and US women who program appear to be more thoughtful, but lower self-esteem and higher risk-aversion leads them to use fewer features.

Gender has become a controversial topic in psychology research. In the early 2000s, discussion of male aptitude for computer science was sometimes in terms of an analysis by Simon Baron-Cohen which gives people separate scores as systemisers (good at geometry and some kinds of symbolic reasoning) and as empathisers (good at intuiting the emotions of others and social intelligence generally) [176]. Most men score higher at systematising, while most women do better at empathising. The correspondence isn't exact; a minority of men are better at empathising while a minority of women are better at systematising. Baron-Cohen's research is in Asperger's and autism spectrum disorder, which he sees as an extreme form of male brain. This theory gained some traction among geeks who saw an explanation of why we're often introverted with more aptitude for understanding things than for understanding people. If we're born that way, it's not our fault. It also suggests an explanation for why geek couples often have kids on the spectrum.

Might this explain why men are more interested in computer science than women, with women consistently taking about a sixth of CS places in the USA and the UK? But here, we run into trouble. Women make up a third of CS students in the former communist countries of Poland, Romania and the Baltic states, while numbers in India are close to equal. Male dominance of software is also a fairly recent phenomenon. When I started out in the 1970s, there were almost as many women programmers as men, and many of the pioneers were women, whether in industry, academia or government. This suggests that the relevant differences are more cultural than genetic or developmental. The argument for a ‘male brain / female brain’ explanation has been progressively undermined by work such as that of Daphna Joel and colleagues who've shown by extensive neuroimaging studies that while there are recognisable male and female features in brains, the brains of individuals are a mosaic of both [985]. And although these features are visible in imaging, that does not mean they're all laid down at birth: our brains have a lot of plasticity. As with our muscles the tissues we exercise grow bigger. Perhaps nothing else might have been expected given the variance in gender identity, sexual preference, aggression, empathy and so on that we see all around us.

Other work has shown that gender performance differences are absent in newborns, and appear round about age 6–7, by which time children have long learned to distinguish gender and adapt to the social cues all around them, which are reinforced in developed countries by a tsunami of blue/pink gendered toys and marketing. (Some believe that women are happier to work in computing in India because India escaped the home computer boom in the 1980s and its evolution into gaming.) This is reinforced in later childhood and adolescence by gender stereotypes that they internalise as part of their identity;

### **3.2. INSIGHTS FROM PSYCHOLOGY RESEARCH**

---

in cultures where girls aren't supposed to be good at maths or interested in computers, praise for being 'good at maths' can evoke a *stereotype threat* (the fear of confirming a negative stereotype about a group to which one belongs). Perhaps as a result, men react better to personal praise ('That was really clever of you!') while women are motivated better by performance praise ('You must have put in a hell of a lot of effort'). So it may not be surprising that we see a deficit of women in disciplines that praise genius, such as mathematics. What's more, similar mechanisms appear to underlie the poorer academic performance of ethnic groups who have been sigmatised as non-academic. In short, people are not just born different; we learn to be different, shaped by power, by cultural attitudes, by expectations and by opportunities. There are several layers between gene and culture with emergent behaviour, including the cell and the circuit. So if we want more effective interventions in the pipeline from school through university to professional development, we need a better understanding of the underlying neurological and cultural mechanisms. For a survey of this, see Gina Rippon [1605].

Gender matters at many levels of the stack, from what a product should do through how it does it. For example, should a car be faster or safer? This is entangled with social values. Are men better drivers because they win car races, or are women better drivers because they have fewer insurance claims? Digging down, we find gendered and cultural attitudes to risk. In US surveys, risks are judged lower by white people and by men, and on closer study this is because about 30% of white males judge risks to be extremely low. This bias is consistent across a wide range of hazards but is particularly strong for handguns, second-hand cigarette smoke, multiple sexual partners and street drugs. Asian males show similarly low sensitivity to some hazards, such as motor vehicles. White males are more trusting of technology, and less of government [693].

We engineers must of course work with the world as it is, not as it might be if our education system and indeed our culture had less bias; but we must be alert to the possibility that computer systems discriminate because they are built by men for men, just like cars and spacesuits. For example, Tyler Moore and I did an experiment to see whether anti-phishing advice given by banks to their customers was easier for men to follow than women, and we found that indeed it was [1337]. No-one seems to have done much work on gender and security usability, so there's an opportunity.

But the problem is much wider. Many systems will continue to be designed by young fit straight clever men who are white or Asian and may not think hard or at all about the various forms of prejudice and disability that they do not encounter directly. You need to think hard about how you mitigate the effects. It's not enough to just have your new product tested by a token geek girl on your development team; you have to think also of the less educated and the vulnerable – including older people, children and women fleeing abusive relationships (about which I'll have more to say later). You really have to think of the whole stack. Diversity matters in corporate governance, market research, product design, software development and testing. If you can't fix the imbalance in dev, you'd better make it up elsewhere. You need to understand your users; it's also good to understand how power and culture feed the imbalance.

As many of the factors relevant to group behaviour are of social origin, we

next turn to social psychology.

#### **3.2.3 Social psychology**

This attempts to explain how the thoughts, feelings, and behaviour of individuals are influenced by the actual, imagined, or implied presence of others. It has many aspects, from the identity that people derive from belonging to groups – whether of gender, tribe, team, profession or even religion – through the self-esteem we get by comparing ourselves with others. The results that put it on the map were three early papers that laid the groundwork for understanding the abuse of authority and its relevance to propaganda, interrogation and aggression. They were closely followed by work on the bystander effect which is also highly relevant to crime and security.

##### **3.2.3.1 Authority and its abuse**

In 1951, Solomon Asch showed that people could be induced to deny the evidence of their own eyes in order to conform to a group. Subjects judged the lengths of lines after hearing wrong opinions from other group members, who were actually the experimenter's stooges. Most subjects gave in and conformed, with only 29% resisting the bogus majority [135].

Stanley Milgram was inspired by the 1961 trial of Nazi war criminal Adolf Eichmann to investigate how many experimental subjects were prepared to administer severe electric shocks to an actor playing the role of a 'learner' at the behest of an experimenter while the subject played the role of the 'teacher' – even when the 'learner' appeared to be in severe pain and begged the subject to stop. This experiment was designed to measure what proportion of people will obey an authority rather than their conscience. Most did – Milgram found that consistently over 60% of subjects would do downright immoral things if they were told to [1312]. This experiment is now controversial but had real influence on the development of the subject.

The third was the Stanford Prisoner Experiment which showed that normal people can behave wickedly even in the absence of orders. In 1971, experimenter Philip Zimbardo set up a 'prison' at Stanford where 24 students were assigned at random to the roles of 12 warders and 12 inmates. The aim of the experiment was to discover whether prison abuses occurred because warders (and possibly prisoners) were self-selecting. However, the students playing the role of warders rapidly became sadistic authoritarians, and the experiment was halted after six days on ethical grounds [2073]. This experiment is also controversial now and it's unlikely that a repeat would get ethical approval today. But abuse of authority, whether real or ostensible, is a real issue if you are designing operational security measures for a business.

During the period 1995–2005, a telephone hoaxter calling himself 'Officer Scott' ordered the managers of over 68 US stores and restaurants in 32 US states (including at least 17 McDonald's stores) to detain some young employee on suspicion of theft and strip-search them. Various other degradations were ordered, including beatings and sexual assaults [2033]. A former prison guard

### **3.2. INSIGHTS FROM PSYCHOLOGY RESEARCH**

---

was tried for impersonating a police officer but acquitted. At least 13 people who obeyed the caller and did searches were charged with crimes, and seven were convicted. McDonald's got sued for not training its store managers properly, even years after the pattern of hoax calls was established; and in October 2007, a jury ordered them to pay \$6.1 million dollars to one of the victims, who had been strip-searched when she was an 18-year-old employee. It was a nasty case, as she was left by the store manager in the custody of her boyfriend, who then committed a further indecent assault on her. The boyfriend got five years, and the manager pleaded guilty to unlawfully detaining her. McDonald's argued that she was responsible for whatever damages she suffered for not realizing it was a hoax, and that the store manager had failed to apply common sense. A Kentucky jury didn't buy this and ordered McDonald's to pay up. The store manager also sued, claiming to be another victim of the firm's negligence to warn her of the hoax, and got \$1.1 million [1088]. So US employers now risk heavy damages if they fail to train their staff to resist the abuse of authority.

#### **3.2.3.2 The bystander effect**

On March 13, 1964, a young lady called Kitty Genovese was stabbed to death in the street outside her apartment in Queens, New York. The press reported that thirty-eight separate witnesses had failed to help or even to call the police, although the assault lasted almost half an hour. Although these reports were later found to be exaggerated, the crime led to the nationwide 911 emergency number, and also to research on why bystanders often don't get involved.

John Darley and Bibb Latané reported experiments in 1968 on what factors modulated the probability of a bystander helping someone who appeared to be having an epileptic fit. They found that a lone bystander would help 85% of the time, while someone who thought that four other people could see the victim would help only 31% of the time; group size dominated all other effects. Whether another bystander was male, female or even medically qualified made essentially no difference [513]. The diffusion of responsibility has visible effects in many other contexts. If you want something done, you'll email one person to ask, not three people. Of course, security is usually seen as something that other people deal with.

However, if you ever find yourself in danger, the real question is whether at least one of the bystanders will help, and here the recent research is much more positive. Lasse Liebst, Mark Levine and others have surveyed CCTV footage of a number of public conflicts in several countries over the last ten years, finding that in 9 out of 10 cases, one or more bystanders intervened to de-escalate a fight, and that the more bystanders intervene, the more successful they are [1163]. So it would be wrong to assume that bystanders generally pass by on the other side; so the bystander effect's name is rather misleading.

#### **3.2.4 The social-brain theory of deception**

Our second big theme, which also fits into social psychology, is the growing body of research into deception. How does deception work, how can we detect and measure it, and how can we deter it?

### **3.2. INSIGHTS FROM PSYCHOLOGY RESEARCH**

---

The modern approach started in 1976 with the social intelligence hypothesis. Until then, anthropologists had assumed that we evolved larger brains in order to make better tools. But the archaeological evidence doesn't support this. All through the paleolithic period, while our brains evolved from chimp size to human size, we used the same simple stone axes. They only became more sophisticated in the neolithic period, by which time our ancestors were anatomically modern homo sapiens. So why, asked Nick Humphrey, did we evolve large brains if we didn't need them yet? Inspired by observing the behaviour of both caged and wild primates, his hypothesis was that the primary function of the intellect was social. Our ancestors didn't evolve bigger brains to make better tools, but to use other primates better as tools [934]. This is now supported by a growing body of evidence, and has transformed psychology as a discipline. Social psychology had been a poor country cousin until then and was not seen as rigorous; since then, people have realised it was probably the driving force of cognitive evolution. Almost all intelligent species developed in a social context. (One exception is the octopus, but even it has to understand how predators and prey react.)

The primatologist Andy Whiten then collected much of the early evidence on tactical deception, and recast social intelligence as the Machiavellian brain hypothesis: we became smart in order to deceive others, and to detect deception too [360]. Not everyone agrees completely with this characterisation, as the positive aspects of socialisation, such as empathy, also matter. But Hugo Mercier and Dan Sperber have recently collected masses of evidence that the modern human brain is more a machine for arguing than anything else [1294]. Our goal is persuasion rather than truth; rhetoric comes first, and logic second.

The second thread coming from the social intellect hypothesis is theory of mind, an idea due to David Premack and Guy Woodruff in 1978 but developed by Heinz Wimmer and Josef Perner in a classic 1983 experiment to determine when children are first able to tell that someone has been deceived [2029]. In this experiment, the Sally-Anne test, a child sees a sweet hidden under a cup by Sally while Anne and the child watch. Anne then leaves the room and Sally switches the sweet to be under a different cup. Anne then comes back and the child is asked where Anne thinks the sweet is. Normal children get the right answer from about age five; this is when they acquire the ability to discern others' beliefs and intentions. Simon Baron-Cohen, Alan Leslie and Uta Frith then showed that children on the Aspergers / autism spectrum acquire this ability significantly later [177].

Many computer scientists and engineers appear to be on the spectrum to some extent, and we're generally not as good at deception as neurotypical people are. This has all sorts of implications! We're under-represented in politics, among senior executives and in marketing. Oh, and there was a lot less cybercrime before underground markets brought together geeks who could write wicked code with crooks and spooks who could use it for wicked purposes. Geeks are also more likely to be whistleblowers; we're less likely to keep quiet about an uncomfortable truth just to please others, as we place less value on their opinions. But this is a complex field. Some well-known online miscreants who are on the spectrum were hapless more than anything else; Gary McKinnon claimed to have hacked the Pentagon to discover the truth about flying saucers

### **3.2. INSIGHTS FROM PSYCHOLOGY RESEARCH**

---

and didn't anticipate the ferocity of the FBI's response. And other kinds of empathic deficit are involved in many crimes. Other people with dispositional empathy deficits include psychopaths who disregard the feelings of others but understand them well enough to manipulate them, while there are many people whose deficits are situational, ranging from Nigerian scammers who think that any white person who falls for their lure deserves it as they must be a racist, to soldiers and terrorists who consider their opponents to be less than human or to be morally deserving of death. I'll discuss radicalisation in more detail later in section 26.4.2.

The third thread is self-deception. Robert Trivers argues that we've evolved the ability to deceive ourselves in order to better deceive others: "If deceit is fundamental in animal communication, then there must be strong selection to spot deception and this ought, in turn, to select for a degree of self-deception, rendering some facts and motives unconscious so as to not betray – by the subtle signs of self-knowledge – the deception being practiced" [904]. We forget inconvenient truths and rationalise things we want to believe. There may well be a range of self-deception abilities from honest geeks through to the great salesmen who have a magic ability to believe completely in their product. But it's controversial, and at a number of levels. For example, if Tony Blair really believed that Iraq had weapons of mass destruction when he persuaded Britain to go to war in 2003, was it actually a lie? How do you define sincerity? How can you measure it? And would you even elect a national leader if you expected that they'd be unable to lie to you? There is a lengthy discussion in [904], and the debate is linked to other work on motivated reasoning. Russell Golman, David Hagman and George Loewenstein survey research on how people avoid information, even when it is free and could lead to better decision-making: people at risk of illness avoid medical tests, managers avoid information that might show they made bad decisions, and investors look at their portfolios less when markets are down [781]. This strand of research goes all the way back to Sigmund Freud, who described various aspects of the *denial* of unpleasant information, including the ways in which we try to minimise our feelings of guilt for the bad things we do, and to blame others for them.

It also links up with filter-bubble effects on social media. People prefer to listen to others who confirm their beliefs and biases, and this can be analysed in terms of the hedonic value of information. People think of themselves as honest and try to avoid the *ethical dissonance* that results from deviations [172]; criminologists use the term *neutralisation* to describe the strategies that rule-breakers use to minimise the guilt that they feel about their actions (there's an overlap with both filter effects and self-deception). A further link is to Hugo Mercier and Dan Sperber's work on the brain as a machine for argument, which I mentioned above.

The fourth thread is intent. The detection of hostile intent was a big deal in our ancestral evolutionary environment; in pre-state societies, perhaps a quarter of men and boys die of homicide, and further back many of our ancestors were killed by animal predators. So we appear to have evolved a sensitivity to sounds and movements that might signal the intent of a person, an animal or even a god. As a result, we now spend too much on defending against threats that involve hostile intent, such as terrorism, and not enough on defending against against

### **3.2. INSIGHTS FROM PSYCHOLOGY RESEARCH**

---

epidemic disease, which kills many more people, or climate change, which could kill even more.

There are other reasons why we might want to think about intent more carefully. In cryptography, we use logics of belief to analyse the security of authentication protocols, and to deal with statements such as ‘Alice believes that Bob believes that Charlie controls the key  $K$ '; we'll come to this in the next chapter. And now we realise that people use theories of mind to understand each other, philosophers have got engaged too. Dan Dennett derived the intentional stance in philosophy, arguing that the propositional attitudes we use when reasoning – beliefs, desires and perceptions – come down to the intentions of people and animals.

A related matter is socially-motivated reasoning: people do logic much better if the problem is set in a social role. In the Wason test, subjects are told they have to inspect some cards with a letter grade on one side, and a numerical code on the other, and given a rule such as “If a student has a grade D on the front of their card, then the back must be marked with code 3”. They are shown four cards displaying (say) D, F, 3 and 7 and then asked “Which cards do you have to turn over to check that all cards are marked correctly?” Most subjects get this wrong; in the original experiment, only 48% of 96 subjects got the right answer of D and 7. However the evolutionary psychologists Leda Cosmides and John Tooby found the same problem becomes easier if the rule is changed to ‘If a person is drinking beer, he must be 20 years old' and the individuals are a beer drinker, a coke drinker, a 25-year-old and a 16-year old. Now three-quarters of subjects deduce that the bouncer should check the age of the beer drinker and the drink of the 16-year-old [483]. Cosmides and Tooby argue that our ability to do logic and perhaps arithmetic evolved as a means of policing social exchanges.

The next factor is rationalisation or minimisation – the process by which people justify bad actions or make their harm appear to be less. I mentioned Nigerian scammers who think that white people who fall for their scam must think Africans are stupid, so they deserve it; there are many more examples of scammers seeing foreign targets as fair game. The criminologist Donald Cressey developed a *Fraud Triangle* theory to explain the factors that lead to fraud: as well as motive and opportunity, there must be a rationalisation. People may feel that their employer has underpaid them so it's justifiable to fiddle expenses, or that the state is wasting money on welfare when they cheat on their taxes. Minimisation is very common in cybercrime. Kids operating DDoS-for-hire services reassured each other that offering a ‘web stresser’ service was legal, and said on their websites that the service could only be used for legal purposes. So undermining minimisation can work as a crime-fighting tool. The UK National Crime Agency bought Google ads to ensure that anyone searching for a web stresser service would see an official warning that DDoS was a crime. A mere £3,000 spent between January and June 2018 suppressed demand growth; DDoS revenues remained constant in the UK while they grew in the USA [454].

Finally, the loss of social context is a factor in online disinhibition. People speak more frankly online, and this has both positive and negative effects. Shy people can find partners, but we also see vicious flame wars. John Suler analyses the factors as anonymity, invisibility, asynchronicity and the loss of symbols of authority and status; in addition there are effects relating to psychic boundaries

### **3.2. INSIGHTS FROM PSYCHOLOGY RESEARCH**

---

and self-imagination which lead us to drop our guard and express feelings from affection to aggression that we normally rein in for social reasons [1845].

Where all this leads is that the nature and scale of online deception can be modulated by suitable interaction design. Nobody is as happy as they appear on Facebook, as attractive as they appear on Instagram or as angry as they appear on Twitter. They let their guard down on closed groups such as those supported by WhatsApp, which offer neither celebrity to inspire performance, nor anonymity to promote trolling. However, people are less critical in closed groups, which makes them more suitable for spreading conspiracy theories, and for radicalisation [523].

#### **3.2.5 Heuristics, biases and behavioural economics**

One field of psychology that has been applied by security researchers since the mid-2000s has been *decision science*, which sits at the boundary of psychology and economics and studies the heuristics that people use, and the biases that influence them, when making decisions. It is also known as *behavioural economics*, as it examines the ways in which people's decision processes depart from the rational behaviour modeled by economists. An early pioneer was Herb Simon – both an early computer scientist and a Nobel-prizewinning economist – who noted that classical rationality meant doing whatever maximizes your expected utility regardless of how hard that choice is to compute. So how would people behave in a realistic world of bounded rationality? The real limits to human rationality have been explored extensively in the years since, and Daniel Kahneman won the Nobel prize in economics in 2002 for his major contributions to this field (along with the late Amos Tversky) [1004].

##### **3.2.5.1 Prospect theory and risk misperception**

Kahneman and Tversky did extensive experimental work on how people made decisions faced with uncertainty. They first developed *prospect theory* which models risk appetite: in many circumstances, people dislike losing \$100 they already have more than they value winning \$100. Framing an action as avoiding a loss can make people more likely to take it; phishermen hook people by sending messages like 'Your PayPal account has been frozen, and you need to click here to unlock it.' We're also bad at calculating probabilities, and use all sorts of heuristics to help us make decisions:

- we often base a judgment on an initial guess or comparison and then adjust it if need be – the *anchoring effect*;
- we base inferences on the ease of bringing examples to mind – the *availability heuristic*, which was OK for lion attacks 50,000 years ago but gives the wrong answers when mass media bombard us with images of terrorism;
- we're more likely to be sceptical about things we've heard than about things we've seen, perhaps as we have more neurons processing vision;
- we worry too much about events that are very unlikely but have very bad consequences;

### **3.2. INSIGHTS FROM PSYCHOLOGY RESEARCH**

---

- we're more likely to believe things we've worked out for ourselves rather than things we've been told.

Behavioral economics is not just relevant to working out how likely people are to click on links in phishing emails, but to the much deeper problem of the perception of risk. Many people perceive terrorism to be a much worse threat than epidemic disease, road traffic accidents or even food poisoning: this is wrong, but hardly surprising to a behavioural economist. We overestimate the small risk of dying in a terrorist attack not just because it's small but because of the visual effect of the 9/11 TV coverage, the ease of remembering the event, the outrage of an enemy attack, and the effort we put into thinking and worrying about it. (There are further factors, which we'll explore in Part III when we discuss terrorism.)

The misperception of risk underlies many other public-policy problems. The psychologist Daniel Gilbert, in an article provocatively entitled 'If only gay sex caused global warming', compares our fear of terrorism with our fear of climate change. First, we evolved to be much more wary of hostile intent than of nature; 100,000 years ago, a man with a club (or a hungry lion) was a much worse threat than a thunderstorm. Second, global warming doesn't violate anyone's moral sensibilities; third, it's a long-term threat rather than a clear and present danger; and fourth, we're sensitive to rapid changes in the environment rather than slow ones [764]. There are many more risk biases: we are less afraid when we're in control, such as when driving a car, as opposed to being a passenger in a car or airplane; and we are more afraid of uncertainty, that is, when the magnitude of the risk is unknown (even when it's small) [1671, 1675]. We also indulge in *satisficing* which means we go for an alternative that's 'good enough' rather than going to the trouble of trying to work out the odds perfectly, especially for small transactions. (The misperception here is not that of the risk taker, but of the economists who ignored the fact that real people include transaction costs in their calculations.)

So, starting out from the folk saying that a bird in the hand is worth two in the bush, we can develop quite a lot of machinery to help us understand and model people's attitudes towards risk.

#### **3.2.5.2 Present bias and hyperbolic discounting**

Saint Augustine famously prayed 'Lord, make me chaste, but not yet.' We find a similar sentiment with applying security updates, where people may pay more attention to the costs as they're immediate and determinate in time, storage and bandwidth, than the unpredictable future benefits. This *present bias* causes many people to decline updates, which was the major source of technical vulnerability online for many years. One way software companies pushed back was by allowing people to delay updates: Windows has 'restart / pick a time / snooze'. Reminders cut the ignore rate from about 90% to about 34%, and may ultimately double overall compliance [726]. A better design is to make updates so painless that they can be made mandatory, or nearly so; this is the approach now followed by some web browsers, and by cloud-based services generally.

### **3.2. INSIGHTS FROM PSYCHOLOGY RESEARCH**

---

*Hyperbolic discounting* is a model used by decision scientists to quantify present bias. Intuitive reasoning may lead people to use utility functions that discount the future so deeply that immediate gratification seems to be the best course of action, even when it isn't. Such models have been applied to try to explain the *privacy paradox* – why people say in surveys that they care about privacy but act otherwise online. I discuss this in more detail in section 8.6.6: other factors, such as uncertainty about the risks and about the efficacy of privacy measures, play a part too. Taken together, the immediate and determinate positive utility of getting free stuff outweighs the random future costs of disclosing too much personal information, or disclosing it to dubious websites.

#### **3.2.5.3 Defaults and nudges**

This leads to the importance of defaults. Many people usually take the easiest path and use the standard configuration of a system, as they assume it will be good enough. In 2009, Richard Thaler and Cass Sunstein wrote a best-seller '*Nudge*' exploring this, pointing out that governments can achieve many policy goals without infringing personal liberty simply by setting the right defaults [1876]. For example, if a firm's staff are enrolled in a pension plan by default, most will not bother to opt out, while if it's optional most will not bother to opt in. A second example is that many more organs are made available for transplant in Spain, where the law lets a dead person's organs be used unless they objected, than in Britain where donors have to consent actively. A third example is that tax evasion can be cut by having the taxpayer declare that the information in the form is true when they start to fill it out, rather than at the end. The set of choices people have to make, the order in which they make them, and the defaults if they do nothing, are called the *choice architecture*. Sunstein got a job in the Obama administration implementing some of these ideas while Thaler won the 2017 economics Nobel prize.

Defaults matter in security too, but often they are set by an adversary so as to trip you up. For example, Facebook defaults to fairly open information sharing, and whenever enough people have figured out how to increase their privacy settings, the architecture is changed so you have to opt out all over again. This exploits not just hazardous defaults but also the *control paradox* – providing the illusion of control causes people to share more information. We like to feel in control; we feel more comfortable driving in our cars than letting someone else fly us in an airplane – even if the latter is an order of magnitude safer. “Privacy control settings give people more rope to hang themselves,” as behavioral economist George Loewenstein puts it. “Facebook has figured this out, so they give you incredibly granular controls.” [1533]

#### **3.2.5.4 The default to intentionality**

Behavioral economists follow a long tradition in psychology of seeing the mind as composed of interacting rational and emotional components – ‘heart’ and ‘head’, or ‘affective’ and ‘cognitive’ systems. Studies of developmental biology have shown that, from an early age, we have different mental processing systems for social phenomena (such as recognising parents and siblings) and physical

### **3.2. INSIGHTS FROM PSYCHOLOGY RESEARCH**

---

phenomena. Paul Bloom argues that the tension between them explains why many people believe that mind and body are basically different [268]. Children try to explain what they see using physics, but when their understanding falls short, they explain phenomena in terms of intentional action. This has survival value to the young, as it disposes them to get advice from parents or other adults about novel natural phenomena. Bloom suggests that it has an interesting side effect: it predisposes humans to believe that body and soul are different, and thus lays the ground for religious belief. This argument may not overwhelm the faithful (who will retort that Bloom simply stumbled across a mechanism created by the Intelligent Designer to cause us to have faith in Him). But it may have relevance for the security engineer.

First, it goes some way to explaining the *fundamental attribution error* – people often err by trying to explain things from intentionality rather than from context. Second, attempts to curb phishing by teaching users about the gory design details of the Internet – for example, by telling them to parse URLs in emails that seem to come from a bank – will be of limited value once they get bewildered. If the emotional is programmed to take over whenever the rational runs out, then engaging in a war of technical instruction and counter-instruction with the phishermen is unsound, as they'll be better at it. Safe defaults would be better.

#### **3.2.5.5 The affect heuristic**

Nudging people to think in terms of intent rather than of mechanism can exploit the *affect heuristic*, explored by Paul Slovic and colleagues [1787]. The idea is that while the human brain can handle multiple threads of cognitive processing, our emotions remain resolutely single-threaded, and they are even less good at probability theory than the rational part of our brains. So by making emotion salient, a marketer or a fraudster can try to get you to answer questions using emotion rather than reason, and using heuristics rather than calculation. A common trick is to ask an emotional question (whether ‘How many dates did you have last month?’ or even ‘What do you think of President Trump?’) to make people insensitive to probability.

So it should not surprise anyone that porn websites have been used to install a lot of malware – as have church websites, which are often poorly maintained and easy to hack. Similarly, events that evoke a feeling of dread – from cancer to terrorism – not only scare people more than the naked probabilities justify, but also make those probabilities harder to calculate, and deter people from even making the effort.

Other factors that can reinforce our tendency to explain things by intent include cognitive overload, where the rational part of the brain simply gets tired. Our capacity for self-control is also liable to fatigue, both physical and mental; some mental arithmetic will increase the probability that we'll pick up a chocolate rather than an apple. So a bank that builds a busy website may be able to sell more life insurance, but it's also likely to make its customers more vulnerable to phishing.

#### 3.2.5.6 Cognitive dissonance

Another interesting offshoot of social psychology is cognitive dissonance theory. People are uncomfortable when they hold conflicting views; they seek out information that confirms their existing views of the world and of themselves, and try to reject information that conflicts with their views or might undermine their self-esteem. One practical consequence is that people are remarkably able to persist in wrong courses of action in the face of mounting evidence that things have gone wrong [1863]. Admitting to yourself or to others that you were duped can be painful; hustlers know this and exploit it. A security professional should ‘feel the hustle’ – that is, be alert for a situation in which recently established social cues and expectations place you under pressure to ‘just do’ something about which you’d normally have reservations. That’s the time to step back and ask yourself whether you’re being had. But training people to perceive this is hard enough, and getting the average person to break the social flow and say ‘stop!’ is hard. There have been some experiments, for example with training health-service staff to not give out health information on the phone, and training people in women’s self-defence classes to resist demands for extra personal information. The problem with mainstreaming such training is that the money available for it is orders of magnitude less than the marketing budgets of the firms whose business model is to hustle their customers.

#### 3.2.5.7 The risk thermostat

Some interesting empirical work has been done on how people manage their exposure to risk. John Adams studied mandatory seat belt laws, and established that they don’t actually save lives: they just transfer casualties from vehicle occupants to pedestrians and cyclists [20]. Seat belts make drivers feel safer, so they drive faster in order to bring their perceived risk back up to its previous level. He calls this a *risk thermostat* and the model is borne out in other applications too [19]. The lesson is that testing needs to have ecological validity: you need to evaluate the effect of a proposed intervention in as realistic a setting as possible.

### 3.3 Deception in practice

This takes us from the theory to the practice. Deception often involves an abuse of the techniques developed by *compliance professionals* – those people whose job it is to get other people to do things. While a sales executive might dazzle you with an offer of a finance plan for a holiday apartment, a police officer might nudge you by their presence to drive more carefully, a park ranger might tell you to extinguish campfires carefully and not feed the bears, and a corporate lawyer might threaten you into taking down something from your website.

The behavioural economics pioneer and apostle of ‘nudge’, Dick Thaler, refers to the selfish use of behavioural economics as ‘sludge’ [1875]. But it’s odd that economists ever thought that the altruistic use of such techniques would ever be more common than the selfish ones. Not only do marketers push

### **3.3. DECEPTION IN PRACTICE**

---

the most profitable option rather than the best value, but they use every other available trick too. Stanford's Persuasive Technology Lab has been at the forefront of developing techniques to keep people addicted to their screens, and one of their alumni, ex-Googler Tristan Harris, has become a vocal critic. Sometimes dubbed 'Silicon valley's conscience', he explains how tech earns its money by manipulating not just defaults but choices, and asks how this can be done ethically [867]. Phones and other screens present menus and thus control choices, but there's more to it than that. Two techniques that screens have made mainstream are the casino's technique of using intermittent variable rewards to create addiction (we check our phones 150 times a day to see if someone has rewarded us with attention) and bottomless message feeds (to keep us consuming even when we aren't hungry any more). But there are many older techniques that predate computers.

#### **3.3.1 The salesman and the scamster**

Deception is the twin brother of marketing, so one starting point is the huge literature about sales techniques. One eminent writer is Robert Cialdini, a psychology professor who took summer jobs selling everything from used cars to home improvements and life insurance in order to document the tricks of the trade. His book '*Influence: science and Practice*' is widely read by sales professionals and describes six main classes of technique used to influence people and close a sale [424].

These are:

1. Reciprocity: most people feel the need to return favours;
2. Commitment and consistency: people suffer cognitive dissonance if they feel they're being inconsistent;
3. Social proof: most people want the approval of others. This means following others in a group of which they're a member, and the smaller the group the stronger the pressure;
4. Liking: most people want to do what a good-looking or otherwise likeable person asks;
5. Authority: most people are deferential to authority figures (recall the Milgram study mentioned above);
6. Scarcity: we're afraid of missing out, if something we might want could suddenly be unavailable.

All of these are psychological phenomena that are the subject of continuing research. They are also traceable to pressures in our ancestral evolutionary environment, where food scarcity was a real threat, strangers could be dangerous and group solidarity against them (and in the provision of food and shelter) was vital. All are used repeatedly in the advertising and other messages we encounter constantly.

### ***3.3. DECEPTION IN PRACTICE***

---

Frank Stajano and Paul Wilson built on this foundation to analyse the principles behind scams. Wilson researched and appeared in nine seasons of TV programs on the most common scams – ‘The Real Hustle’ – where the scams would be perpetrated on unsuspecting members of the public, who would then be given their money back, debriefed and asked permission for video footage to be used on TV. The know-how from experimenting with several hundred frauds on thousands of marks over several years was distilled into the following seven principles [1820].

1. Distraction – the fraudster gets the mark to concentrate on the wrong thing. This is at the heart of most magic performances.
2. Social compliance – society trains us not to question people who seem to have authority, leaving people vulnerable to conmen who pretend to be from their bank or from the police.
3. The herd principle – people let their guard down when everyone around them appears to share the same risks. This is a mainstay of the three-card trick, and a growing number of scams on social networks.
4. Dishonesty – if the mark is doing something dodgy, they’re less likely to complain. Many are attracted by the idea that ‘you’re getting a good deal because it’s illegal’, and whole scam families – such as the resale of fraudulently obtained plane tickets – turn on this.
5. Kindness – this is the flip side of dishonesty, and an adaptation of Cialdini’s principle of reciprocity. Many social engineering scams rely on the victims’ helpfulness, from tailgating into a building to phoning up with a sob story to ask for a password reset.
6. Need and greed – sales trainers tell us we should find what someone really wants and then show them how to get it. A good fraudster can help the mark dream a dream and use this to milk them.
7. Time pressure – this causes people to act viscerally rather than stopping to think. Normal marketers use this all the time (‘only 2 seats left at this price’); so do crooks.

The relationship with Cialdini’s principles should be obvious. A cynic might say that fraud is just a subdivision of marketing; or perhaps that, as marketing becomes ever more aggressive, it comes to look ever more like fraud. When we investigated online accommodation scams we found it hard to code detectors, since many real estate agents use the same techniques. In fact, the fraudsters’ behaviour was already well described by Cialdini’s model, except the scamsters added appeals to sympathy, arguments to establish their own credibility, and ways of dealing with objections [2062]. (These are also found elsewhere in the regular marketing literature.)

Oh, and we find the same in software, where there’s a blurry dividing line between illegal malware and just-about-legal ‘Potentially Unwanted Programs’ (PUPs) such as browser plugins that replace your ads with different ones. One good distinguisher seems to be technical: malware is distributed by many small

### **3.3. DECEPTION IN PRACTICE**

---

botnets because of the risk of arrest, while PUPs are mostly distributed by one large network [954]. But crooks use regular marketing channels too: Ben Edelman found in 2006 that while 2.73% of companies ranked top in a web search were bad, 4.44% of companies that appeared alongside in the search ads were bad [612]. Bad companies were also more likely to exhibit cheap trust signals, such as TRUSTe privacy certificates on their websites. Similarly, bogus landlords often send reference letters or even copies of their ID to prospective tenants, something that genuine landlords never do.

And then there are the deceptive marketing practices of ‘legal’ businesses. To take just one of many studies, a 2019 crawl of 11K shopping websites by Arunesh Mathur and colleagues found 1,818 instances of ‘dark patterns’ – manipulative marketing practices such as hidden subscriptions, hidden costs, pressure selling, sneak-into-basket tactics and forced account opening. Of these at least 183 were clearly deceptive [1242]. What’s more, the bad websites were among the most popular; perhaps a quarter to a third of websites you visit, weighted by traffic, try to hustle you. This constant pressure from scams that lie just short of the threshold for a fraud prosecution has a chilling effect on trust generally. People are less likely to believe security warnings if they are mixed with marketing, or smack of marketing in any way. And we even see some loss of trust in software updates; people say in surveys that they’re less likely to apply a security-plus-features upgrade than a security patch, though the field data on upgrades don’t (yet) show any difference [1591].

#### **3.3.2 Social engineering**

Hacking systems through the people who operate them is not new. Military and intelligence organisations have always targeted each other’s staff; most of the intelligence successes of the old Soviet Union were of this kind [118]. Private investigation agencies have not been far behind.

Investigative journalists, private detectives and fraudsters developed the false-pretext phone call into something between an industrial process and an art form in the latter half of the 20th century. An example of the industrial process was how private detectives tracked people in Britain. Given that the country has a National Health Service with which everyone’s registered, the trick was to phone up someone with access to the administrative systems in the area you thought the target was, pretend to be someone else in the health service, and ask. Colleagues of mine did an experiment in England in 1996 where they trained the staff at a local health authority to identify and report such calls<sup>1</sup>. They detected about 30 false-pretext calls a week, which would scale to 6000 a week or 300,000 a year for the whole of Britain. That eventually got sort-of fixed but it took over a decade. The real fix wasn’t the enforcement of privacy law, but that administrators simply stopped answering the phone.

Another old scam from the 20th century is to steal someone’s ATM card and then phone them up pretending to be from the bank asking whether their card’s been stolen. On hearing that it has, the conman says ‘We thought so. Please

---

<sup>1</sup>The story is told in detail in chapter 9 of the second edition of this book, available free online.

### **3.3. DECEPTION IN PRACTICE**

---

just tell me your PIN now so I can go into the system and cancel your card.' The most rapidly growing recent variety is the 'authorised push payment', where the conman again pretends to be from the bank, and persuades the customer to make a transfer to another account, typically by confusing the customer about the bank's authentication procedures, which most customers find rather mysterious anyway<sup>2</sup>.

As for art form, one of the most disturbing security books ever published is Kevin Mitnick's '*Art of Deception*'. Mitnick, who was arrested and convicted for breaking into US phone systems, related after his release from prison how almost all of his exploits had involved social engineering. His typical hack was to pretend to a phone company employee that he was a colleague, and solicit 'help' such as a password. Ways of getting past a company's switchboard and winning its people's trust are a staple of sales-training courses, and hackers apply these directly. A harassed system administrator is called once or twice on trivial matters by someone claiming to be the CEO's personal assistant; once this idea has been accepted, the caller demands a new password for the boss. Mitnick became an expert at using such tricks to defeat company security procedures, and his book recounts a fascinating range of exploits [1325].

Social engineering became world headline news in September 2006 when it emerged that Hewlett-Packard chairwoman Patricia Dunn had hired private investigators who used pretexting to obtain the phone records of other board members of whom she was suspicious, and of journalists she considered hostile. She was forced to resign. The detectives were convicted of fraudulent wire communications and sentenced to do community service [138]. In the same year, the UK privacy authorities prosecuted a private detective agency that did pretexting jobs for top law firms [1138].

Amid growing publicity about social engineering, there was an audit of the IRS in 2007 by the Treasury Inspector General for Tax Administration, whose staff called 102 IRS employees at all levels, asked for their user IDs, and told them to change their passwords to a known value; 62 did so. What's worse, this happened despite similar audit tests in 2001 and 2004 [1673]. Since then, a number of audit firms have offered social engineering as a service; they phish their audit clients to show how easy it is. Since the mid-2010s, opinion has shifted against this practice, as it causes a lot of distress to staff without changing behaviour very much.

Social engineering isn't limited to stealing private information. It can also be about getting people to believe bogus public information. The quote from Bruce Schneier at the head of this chapter appeared in a report of a stock scam, where a bogus press release said that a company's CEO had resigned and its earnings would be restated. Several wire services passed this on, and the stock dropped 61% until the hoax was exposed [1670]. Fake news of this kind has been around forever, but the Internet has made it easier to promote and social media seem to be making it ubiquitous. We'll revisit this issue when I discuss censorship in section 26.4.

---

<sup>2</sup>Very occasionally, a customer can confuse the bank; a 2019 innovation was the 'callhamer' attack, where someone phones up repeatedly to 'correct' the spelling of 'his name' and changes it one character at a time into another one.

### **3.3.3 Phishing**

While phone-based social engineering was the favoured tactic of the 20th century, online phishing seems to have replaced it as the main tactic of the 21st. The operators include both spooks and crooks, while the targets are both your staff and your customers. It is difficult enough to train your staff; training the average customer is even harder. They'll assume you're trying to hustle them, ignore your warnings and just figure out the easiest way to get what they want from your system. And you can't design simply for the average. If your systems are not safe to use by people who don't speak English well, or who are dyslexic, or who have learning difficulties, you are asking for serious legal trouble. So the easiest way to use your system had better be the safest.

The word 'phishing' appeared in 1996 in the context of the theft of AOL passwords. By then, attempts to crack email accounts to send spam had become common enough for AOL to have a 'report password solicitation' button on its web page; and the first reference to 'password fishing' is in 1990, in the context of people altering terminal firmware to collect Unix logon passwords [443]. Also in 1996, Tony Greening reported a systematic experimental study: 336 computer science students at the University of Sydney were sent an email message asking them to supply their password on the pretext that it was required to 'validate' the password database after a suspected break-in. 138 of them returned a valid password. Some were suspicious: 30 returned a plausible looking but invalid password, while over 200 changed their passwords without official prompting. But very few of them reported the email to authority [812].

Phishing attacks against banks started seven years later in 2003, with half-a-dozen attempts reported [441]. The early attacks imitated bank websites, but were both crude and greedy; the attackers asked for all sorts of information such as ATM PINs, and their emails were also written in poor English. Most customers smelt a rat. By about 2008, the attackers learned to use better psychology; they often reused genuine bank emails, with just the URLs changed, or sent an email saying something like 'Thank you for adding a new email address to your PayPal account' to provoke the customer to log on to complain that they hadn't. Of course, customers who used the provided link rather than typing in [www.paypal.com](http://www.paypal.com) or using an existing bookmark would get their accounts emptied. By then phishing was being used by state actors too; I described in section 2.2.2 how Chinese intelligence compromised the Dalai Lama's private office during the 2008 Olympic games. They used crimeware tools that were originally used by Russian fraud gangs, which they seemed to think gave them some deniability afterwards.

Fraud losses grew rapidly but stabilised by about 2015. A number of countermeasures helped bring things under control, including more complex logon schemes (using two-factor authentication, or its low-cost cousin, the request for some random letters of your password); a move to webmail systems that filter spam better; and back-end fraud engines that look for cashout patterns. The competitive landscape was rough, in that the phishermen would hit the easiest targets at any time in each country, both in terms of stealing their customer credentials and using their accounts to launder stolen funds. Concentrated losses caused the targets to wake up and take action. Since then, we've seen large-

### *3.3. DECEPTION IN PRACTICE*

---

scale attacks on non-financial firms like Amazon; in the late 2000s, the crook would change your email and street address, then use your credit card to order a wide-screen TV. Since about 2016, the action has been in gift vouchers.

As we noted in the last chapter, phishing is also used at scale by botmasters to recruit new machines to their botnets, and in targeted ways both by crooks aiming at specific people or firms, and by intelligence agencies. There's a big difference between attacks conducted at scale, where the economics dictate that the cost of recruiting a new machine to a botnet can be at most a few cents, and targeted attacks, where spooks can spend years trying to hack the phone of a rival head of government, or a smart crook can spend weeks or months of effort stalking a chief financial officer in the hope of a large payout. The lures and techniques used are different, even if the crimeware installed on the target's laptop or phone comes from the same stable. Cormac Herley argues that this gulf between the economics of targeted crime and volume crime is one of the reasons why cybercrime isn't much worse than it is [887]. After all, given that we depend on computers, and that all computers are insecure, and that there are attacks all the time, how come civilisation hasn't collapsed? Cybercrime can't always be as easy as it looks.

Another factor is that it takes time for innovations to be developed and disseminated. We noted that it took seven years for the bad guys to catch up with Tony Greening's 1995 phishing work. As another example, a 2007 paper by Tom Jagatic and colleagues showed how to make phishing much more effective by automatically personalising each phish using context mined from the target's social network [971]. I cited that in the second edition of this book, and in 2016 we saw it in the wild: a gang sent hundreds of thousands of phish with US and Australian banking Trojans to individuals working in finance departments of companies, with their names and job titles apparently scraped from LinkedIn [1297]. This seems to have been crude and hasn't really caught on, but once the bad guys figure it out we may see spear-phishing at scale in the future, and it's interesting to think of how we might respond. The other personalised bulk scams we see are blackmail attempts where the victims get email claiming that their personal information has been compromised and including a password or the last four digits of a credit card number as evidence, but the yield from such scams seems to be low.

As I write, crime gangs have been making ever more use of spear-phishing in targeted attacks on companies where they install ransomware, steal gift coupons and launch other scams. In 2020, a group of young men hacked Twitter, where over a thousand employees had access to internal tools that enabled them to take control of user accounts; the gang sent bitcoin scam tweets from the accounts of such well-known users as Bill Gates, Barack Obama and Elon Musk [1292]. They appear to have honed their spear-phishing skills on SIM swap fraud, which I'll discuss later in sections 3.4.1 and 12.7.4. The spread of such 'transferable skills' among crooks is similar in many ways to the adoption of mainstream technology.

### 3.3.4 Opsec

Getting your staff to resist attempts by outsiders to inveigle them into revealing secrets, whether over the phone or online, is known in military circles as *operational security* or Opsec. Protecting really valuable secrets, such as unpublished financial data, not-yet-patented industrial research and military plans, depends on limiting the number of people with access, and also on doctrines about what may be discussed with whom and how. It's not enough for rules to exist; you have to train the staff who have access, explain the reasons behind the rules, and embed them socially in the organisation. In our medical privacy case, we educated health service staff about pretext calls and set up a strict callback policy: they would not discuss medical records on the phone unless they had called a number they had got from the health service internal phone book rather than from a caller. Once the staff have detected and defeated a few false-pretext calls, they talk about it and the message gets embedded in the way everybody works.

Another example comes from a large Silicon Valley service firm, which suffered intrusion attempts when outsiders tailgated staff into buildings on campus. Stopping this with airport-style ID checks, or even card-activated turnstiles, would have changed the ambience and clashed with the culture. The solution was to create and embed a social rule that when someone holds open a building door for you, you show them your badge. The critical factor, as with the bogus phone calls, is social embedding rather than just training.

Often the hardest people to educate are the most senior; a consultancy sent the finance directors of 500 publicly-quoted companies a USB memory stick as part of an anonymous invitation saying 'For Your Chance to Attend the Party of a Lifetime', and 46% of them put it into their computers [1031]. In my own experience in banking, the people you couldn't train were those who were paid more than you, such as traders in the dealing rooms.

Some operational security measures are common sense, such as not throwing sensitive papers in the trash. Less obvious is the need to train the people you trust. A leak of embarrassing emails that appeared to come from the office of UK Prime Minister Tony Blair and was initially blamed on 'hackers' turned out to have been fished out of the trash at his personal pollster's home by a private detective [1208].

People operate systems however they have to, and this usually means breaking some of the rules in order to get their work done. Research shows that company staff have only so much *compliance budget*, that is, they're only prepared to put so many hours a year into tasks that are not obviously helping them achieve their goals [196]. You need to figure out what this budget is, and use it wisely. If there's some information you don't want your staff to be tricked into disclosing, it's safer to design systems so that they just can't disclose it, or at least so that disclosures involve talking to other staff members or jumping through other hoops.

But what about a firm's customers? There is a lot of scope for phishermen to simply order bank customers to reveal their security data, and this happens at scale, against both retail and business customers. There are also the many

### **3.4. PASSWORDS**

---

small scams that customers try on when they find vulnerabilities in your business processes. I'll discuss both types of fraud further in the chapter on banking and bookkeeping.

#### **3.3.5 Deception research**

Finally, a word on deception research. Since 9/11, huge amounts of money have been spent by governments trying to find better lie detectors, and deception researchers are funded across about five different subdisciplines of psychology. The polygraph measures stress via heart rate and skin conductance; it has been around since the 1920s and is used by some US states in criminal investigations, as well as by the Federal government in screening people for Top Secret clearances. The evidence on its effectiveness is patchy at best, and surveyed extensively by Aldert Vrij [1970]. While it can be an effective prop in the hands of a skilled interrogator, the key factor is the skill rather than the prop. When used by unskilled people in a lab environment, against experimental subjects telling low-stakes lies, its output is little better than random. As well as measuring stress via skin conductance, you can measure distraction using eye movements and guilt by upper body movements. In a research project with Sophie van der Zee, we used body motion-capture suits and also the gesture-recognition cameras in an Xbox and got slightly better results than a polygraph [2063]. However such technologies can at best augment the interrogator's skill, and claims that they work well should be treated as junk science. Thankfully, the government dream of an effective interrogation robot is some way off.

A second approach to dealing with deception is to train a machine-learning classifier on real customer behaviour. This is what credit-card fraud engines have been doing since the late 1990s, and recent research has pushed into other fields too. For example, Noam Brown and Tuomas Sandholm have created a poker-playing bot called Pluribus that beat a dozen expert players over a 12-day marathon of 10,000 hands of Texas Hold 'em. It doesn't use psychology but game theory, playing against itself millions of times and tracking regret at bids that could have given better outcomes. That it can consistently beat experts without access to 'tells' such as its opponents' facial gestures or body language is itself telling. Dealing with deception using statistical machine learning rather than physiological monitoring may also be felt to intrude less into privacy.

## **3.4 Passwords**

The management of passwords gives an instructive context in which usability, applied psychology and security meet. Passwords have been one of the biggest practical problems facing security engineers since perhaps the 1970s. In fact, as the usability researcher Angela Sasse puts it, it's hard to think of a worse authentication mechanism than passwords, given what we know about human memory: people can't remember infrequently-used or frequently-changed items; we can't forget on demand; recall is harder than recognition; and non-meaningful words are more difficult.

To place the problem in context, most passwords you're asked to set are not

### 3.4. PASSWORDS

---

for your benefit but for somebody else's. The modern media ecosystem is driven by websites seeking to maximise both their page views and their registered user bases so as to maximise their value when they are sold. That's why, when you're pointed to a news article that's so annoying you feel you have to leave a comment, you find you have to register. Click, and there's a page of ads. Fill out the form with an email address and submit. Got the CAPTCHA wrong, so do it again and see another page of ads. Click on the email link, and see a page with another ad. Now you can add a comment that nobody will ever read. In such circumstances you're better to type random garbage and let the browser remember it; or better still, don't bother. Even major news sites use passwords against the reader's interest, for example by limiting the number of free page views you get per month unless you register again with a different browser. This ecosystem is described in detail by Ryan Holiday [913].

Turning now to the more honest uses, the password system used by a big modern service firm has a number of components.

1. The visible part is the logon page, which asks you to choose a password when you register and probably checks its strength in some way. It later asks for this password whenever you log on.
2. There will be recovery mechanisms that enable you to deal with a forgotten password or even a compromised account, typically by asking further security questions, or via your primary email account, or by sending an SMS to your phone.
3. Behind this lie technical protocol mechanisms for password checking, typically routines that encrypt your password when you enter it at your laptop or phone, and then either compare it with a local encrypted value, or take it to a remote server for checking.
4. There are often protocol mechanisms to synchronise passwords across multiple platforms, so that if you change your password on your laptop, your phone won't let you use that service until you enter the new one there too. And these mechanisms may enable you to blacklist a stolen phone without having to reset the passwords for all the services it was able to access.
5. There will be intrusion-detection mechanisms to propagate an alarm if one of your passwords is used somewhere it probably shouldn't be.
6. There are single-signon mechanisms to use one logon for many websites, as when you use your Google or Facebook account to log on to a newspaper.

Let's work up from the bottom. Developing a full-feature password management system can be a lot of work, and providing support for password recovery also costs money (a few years ago, the UK phone company BT had two hundred people in its password-reset centre). So outsourcing 'identity management' can make business sense. In addition, intrusion detection works best at scale: if someone uses my gmail password in an Internet cafe in Peru while Google knows I'm in Scotland, they send an SMS to my phone to check, and a small website can't do that. The main cause of attempted password abuse is when one firm gets hacked, disclosing millions of email addresses and passwords, which the bad

### 3.4. PASSWORDS

---

guys try out elsewhere; big firms spot this quickly while small ones don't. The big firms also help their customers maintain situational awareness, by alerting you to logons from new devices or from strange places. Again, it's hard to do that if you're a small website or one that people visit infrequently.

As for syncing passwords between devices, only the device vendors can really do that well; and the protocol mechanisms for encrypting passwords in transit to a server that verifies them will be discussed in the next chapter. That brings us to password recovery.

#### 3.4.1 Password recovery

The experience of the 2010s, as the large service firms scaled up and people moved en masse to smartphones, is that password recovery is often the hardest aspect of authentication. If people you know, such as your staff, forget their passwords, you can get them to interact with an administrator or manager who knows them. But for people you don't know such as your online customers it's harder. And as a large service firm will be recovering tens of thousands of accounts every day, you need some way of doing it without human intervention in the vast majority of cases.

During the 1990s and 2000s, many websites did password recovery using 'security questions' such as asking for your favourite team, the name of your pet or even that old chestnut, your mother's maiden name. Such near-public information is often easy to guess so it gave an easier way to break into accounts than guessing the password itself. This was made even worse by everyone asking the same questions. In the case of celebrities – or abuse by a former intimate partner – there may be no usable secrets. This was brought home to the public in 2008, when a student hacked the Yahoo email account of US Vice-Presidential candidate Sarah Palin via the password recovery questions – her date of birth and the name of her first school. Both of these were public information. Since then, crooks have learned to use security questions to loot accounts when they can; at the US Social Security Administration, a common fraud was to open an online account for a pensioner who's dealt with their pension by snail mail in the past, and redirect the payments to a different bank account. This peaked in 2013; the countermeasure that fixed it was to always notify beneficiaries of account changes by snail mail.

In 2015, five Google engineers published a thorough analysis of security questions, and many turned out to be extremely weak. For example, an attacker could get a 19.7% success rate against 'Favourite food?' in English. Some 37% of people provided wrong answers, in some cases to make them stronger, but sometimes not. Fully 16% of people's answers were public. In addition to being insecure, the 'security questions' turned out to be hard to use: 40% of English-speaking US users were unable to recall the answers when needed, while twice as many could recover accounts using an SMS reset code [291].

Given these problems with security and memorability, most websites now let you recover your password by an email to the address with which you first registered. But if someone compromises that email account, they can get all your dependent accounts too. Email recovery may be adequate for websites

### 3.4. PASSWORDS

---

where a compromise is of little consequence, but for important accounts – such as banking and email itself – standard practice is now to use a second factor. This is typically a code sent to your phone by SMS, or better still using an app that can encrypt the code and tie it to a specific handset. Many service providers that allow email recovery are nudging people towards using such a code instead where possible. Google research shows that SMSs stop all bulk password guessing by bots, 96% of bulk phishing and 76% of targeted attacks [574].

But this depends on phone companies taking care over who can get a replacement SIM card, and many don't. The problem in 2020 is rapid growth in attacks based on intercepting SMS authentication codes, which mostly seem to involve SIM swap, where the attacker pretends to be you to your mobile phone company and gets a replacement SIM card for your account. SIM-swap attacks started in South Africa in 2007, became the main form of bank fraud in Nigeria, then caught on in America – initially as a means of taking over valuable Instagram accounts, then to loot people's accounts at bitcoin exchanges, then for bank fraud more generally [1092]. I will discuss SIM-swap attacks in more detail in section 12.7.4.

Attackers have also exploited the SS7 signalling protocol to wiretap targets' mobile phones remotely and steal codes [489]. I'll discuss such attacks in more detail in the chapters on phones and on banking. The next step in the arms race will be moving customers from SMS messages for authentication and account recovery to an app; the same Google research shows that this improves these last two figures to 99% for bulk phishing and 90% for targeted attacks [574]. As for the targeted attacks, other research by Ariana Mirian along with colleagues from UCSD and Google approached gangs who advertised 'hack-for-hire' services online and asked them to phish Gmail passwords. Three of the gangs succeeded, defeating SMS-based 2fa with a middleperson attack; forensics then revealed 372 other attacks on Gmail users from the same IP addresses during March to October 2018 [1322]. This is still an immature criminal market, but to stop such attacks an app or authentication token is the way to go. It also raises further questions about account recovery. If I use a hardware security key on my Gmail, do I need a second one in a safe as a recovery mechanism? (Probably.) If I use one app on my phone to do banking and another as an authenticator, do I comply with rules on two-factor authentication? (See section 12.7.4 in the chapter on banking.)

Email notification is the default for telling people not just of suspicious login attempts, but of logins to new devices that succeeded with the help of a code. That way, if someone plants malware on your phone, you have some chance of detecting it. How a victim recovers then is the next question. If all else fails, a service provider may eventually let them speak to a real person. But when designing such a system, never forget that it's only as strong as the weakest fallback mechanism – be it a recovery email loop with an email provider you don't control, a phone code that's vulnerable to SIM swapping or mobile malware, or a human who's open to social engineering.

### 3.4.2 Password choice

Many accounts are compromised by guessing PINs or passwords. There are botnets constantly breaking into online accounts by guessing passwords and password-recovery questions, as I described in 2.3.1.4, in order to use email accounts to send spam and to recruit machines to botnets. And as people invent new services and put passwords on them, the password guessers find new targets. A recent example is cryptocurrency wallets: an anonymous ‘bitcoin bandit’ managed to steal \$50m by trying lots of weak passwords for ethereum wallets [809]. Meanwhile, billions of dollars’ worth of cryptocurrency has been lost because passwords were forgotten. So passwords matter, and there are basically three broad concerns, in ascending order of importance and difficulty:

1. Will the user enter the password correctly with a high enough probability?
2. Will the user remember the password, or will they have to either write it down or choose one that’s easy for the attacker to guess?
3. Will the user break the system security by disclosing the password to a third party, whether accidentally, on purpose, or as a result of deception?

### 3.4.3 Difficulties with reliable password entry

The first human-factors issue is that if a password is too long or complex, users might have difficulty entering it correctly. If the operation they’re trying to perform is urgent, this might have safety implications. If customers have difficulty entering software product activation codes, this can generate expensive calls to your support desk. And the move from laptops to smartphones during the 2010s has made password rules such as ‘at least one lower-case letter, upper-case letter, number and special character’ really fiddly and annoying. This is one of the factors pushing people toward longer but simpler secrets, such as passphrases of three or four words. But will people be able to enter them without making too many errors?

An interesting study was done for the STS prepayment meters used to sell electricity in many less-developed countries. The customer hands some money to a sales agent, and gets a 20-digit number printed out on a receipt. They take this receipt home, enter the numbers at a keypad in the meter, and the lights come on. The STS designers worried that since a lot of the population was illiterate, and since people might get lost halfway through entering the number, the system might be unusable. But illiteracy was not a problem: even people who could not read had no difficulty with numbers (‘everybody can use a phone’, as one of the engineers said). The biggest problem was entry errors, and these were dealt with by printing the twenty digits in two rows, with three groups of four digits in the first row followed by two in the second [93]. I’ll describe this in detail in section 14.2.

A quite different application is the firing codes for US nuclear weapons. These consist of only 12 decimal digits. If they are ever used, the operators will be under extreme stress, and possibly using improvised or obsolete communications channels. Experiments suggested that 12 digits was the maximum that

could be conveyed reliably in such circumstances. I'll discuss how this evolved in 15.2.

#### 3.4.4 Difficulties with remembering the password

Our second psychological issue is that people often find passwords hard to remember [2076]. Twelve to twenty digits may be easy to copy from a telegram or a meter ticket, but when customers are expected to memorize passwords, they either choose values that are easy for attackers to guess, or write them down, or both. In fact, standard password advice has been summed up as: “Choose a password you can't remember, and don't write it down”.

The problems are not limited to computer access. For example, one chain of cheap hotels in France introduced self service. You'd turn up at the hotel, swipe your credit card in the reception machine, and get a receipt with a numerical access code to unlock your room door. To keep costs down, the rooms did not have en-suite bathrooms. A common failure mode was that you'd get up in the middle of the night to go to the bathroom, forget your access code, and realise you hadn't taken the receipt with you. So you'd have to sleep on the bathroom floor until the staff arrived the following morning.

Password memorability can be discussed under five main headings: naïve choice, user abilities and training, design errors, operational failures and vulnerability to social-engineering attacks.

##### 3.4.4.1 Naïve choice

Since the mid-1980s, people have studied what sort of passwords people choose, and found they use spouses' names, single letters, or even just hit carriage return giving an empty string as their password. Cryptanalysis of tapes from a 1980 Unix system showed that of the pioneers, Dennis Ritchie used ‘dmac’ (his middle name was MacAlistair); the later Google chairman Eric Schmidt used ‘wendy!!!’ (his wife's name) and Brian Kernighan used ‘././.’ [795]. Fred Grampp and Robert Morris's classic 1984 paper on Unix security [805] reports that after software became available which forced passwords to be at least six characters long and have at least one nonletter, they made a file of the 20 most common female names, each followed by a single digit. Of these 200 passwords, at least one was in use on each of several dozen machines they examined. At the time, Unix systems kept encrypted passwords in a file `/etc/passwd` that all system users could read, so any user could verify a guess of any other user's password. Other studies showed that requiring a non-letter simply changed the most popular password from ‘password’ to ‘password1’ [1672].

In 1990, Daniel Klein gathered 25,000 Unix passwords and found that 21–25% of passwords could be guessed depending on the amount of effort put in [1056]. Dictionary words accounted for 7.4%, common names for 4%, combinations of user and account name 2.7%, and so on down a list of less probable choices such as words from science fiction (0.4%) and sports terms (0.2%). Other password guesses used patterns, such as by taking an account ‘*klone*’ belonging to the user ‘Daniel V. Klein’ and trying passwords such as *klone*, *klone1*,

### **3.4. PASSWORDS**

---

klone123, dvk, dvkdvk, leinad, neilk, DvkkvD, and so on. The following year, Alec Muffett released ‘crack’, software that would try to brute-force Unix passwords using dictionaries and patterns derived from them by a set of mangling rules.

The largest academic study of password choice of which I am aware is by Joe Bonneau, who in 2012 analysed tens of millions of passwords in leaked password files, and also interned at Yahoo where he instrumented the login system to collect live statistics on the choices of 70 million users. He also worked out the best metrics to use for password guessability, both in standalone systems and where attackers use passwords harvested from one system to crack accounts on another [289]. This work informed the design of password strength checkers and other current practices at the big service firms.

#### **3.4.4.2 User abilities and training**

Sometimes you can train the users. Password checkers have trained them to use longer passwords with numbers as well as letters, and the effect spills over to websites that don’t use them [444]. But you do not want to drive customers away, so the marketing folks will limit what you can do. In fact, research shows that password rule enforcement is not a function of the value at risk, but of whether the website is a monopoly. Such websites typically have very annoying rules, while websites with competitors, such as Amazon, are more usable, placing more reliance on back-end intrusion-detection systems.

In a corporate or military environment you can enforce password choice rules, or password change rules, or issue random passwords. But then people will have to write them down. So you can insist that passwords are treated the same way as the data they protect: bank master passwords go in the vault overnight, while military ‘Top Secret’ passwords must be sealed in an envelope, in a safe, in a room that’s locked when not occupied, in a building patrolled by guards. You can send guards round at night to clean all desks and bin everything that hasn’t been locked up. But if you want to hire and retain good people, you’d better think things through a bit more carefully. For example, one Silicon Valley firm had a policy that the root password for each machine would be written down on a card and put in an envelope taped to the side of the machine – a more human version of the rule that passwords be treated the same way as the data they protect. The domestic equivalent is the card in the back of your wifi router with the password.

While writing the first edition of this book, I could not find any account of experiments on training people in password choice that would hold water by the standards of applied psychology (i.e., randomized controlled trials with adequate statistical power). The closest I found was a study of the recall rates, forgetting rates, and guessing rates of various types of password [345]; this didn’t tell us the actual effects of giving users various kinds of advice. We therefore decided to see what could be achieved by training, and selected three groups of about a hundred volunteers from our first-year science students [2055]:

- the red (control) group was given the usual advice (password at least six characters long, including one nonletter)

### 3.4. PASSWORDS

---

- the green group was told to think of a passphrase and select letters from it to build a password. So ‘It’s 12 noon and I am hungry’ would give ‘I’S12&IAH’
- the yellow group was told to select eight characters (alpha or numeric) at random from a table we gave them, write them down, and destroy this note after a week or two once they’d memorized the password.

What we expected to find was that the red group’s passwords would be easier to guess than the green group’s which would in turn be easier than the yellow group’s; and that the yellow group would have the most difficulty remembering their passwords (or would be forced to reset them more often), followed by green and then red. But that’s not what we found.

About 30% of the control group chose passwords that could be guessed using Alec Muffett’s ‘crack’ software, versus about 10 percent for the other two groups. So passphrases and random passwords seemed to be about equally effective. When we looked at password reset rates, there was no significant difference between the three groups. When we asked the students whether they’d found their passwords hard to remember (or had written them down), the yellow group had significantly more problems than the other two; but there was no significant difference between red and green.

The conclusions we drew were as follows.

- For users who follow instructions, passwords based on mnemonic phrases offer the best of both worlds. They are as easy to remember as naively selected passwords, and as hard to guess as random passwords.
- The problem then becomes one of *user compliance*. A significant number of users (perhaps a third of them) just don’t do what they’re told.

So when the army gives soldiers randomly-selected passwords, its value comes from the fact that the password assignment compels user compliance, rather than from the fact that they’re random (as mnemonic phrases would do just as well).

But centrally-assigned passwords are often inappropriate. When you are offering a service to the public, your customers expect you to present broadly the same interfaces as your competitors. So you must let users choose their own website passwords, subject to some lightweight algorithm to reject passwords that are ‘clearly bad’. (GCHQ suggests using a ‘bad password list’ of the 100,000 passwords most commonly found in online password dumps.) In the case of bank cards, users expect a bank-issued initial PIN plus the ability to change the PIN afterwards to one of their choosing (though again you may block a ‘clearly bad’ PIN such as 0000 or 1234). Over half of cardholders keep a random PIN, but about a quarter choose PINs such as children’s birth dates which have less entropy than random PINs would, and have the same PIN on different cards. The upshot is that a thief who steals a purse or wallet may have a chance of about one in eleven to get lucky, if he tries the most common PINs on all the cards first in offline mode and then in online mode, so he gets six goes at each. Banks that forbid popular choices such as 1234 can increase the odds to about one in eighteen [295].

#### 3.4.4.3 Design errors

Attempts to make passwords memorable are a frequent source of severe design errors. The classic example of how not to do it is to ask for ‘your mother’s maiden name’. A surprising number of banks, government departments and other organisations still authenticate their customers in this way, though nowadays it tends to be not a password but a password recovery question. You could always try to tell ‘Yngstrom’ to your bank, ‘Jones’ to the phone company, ‘Geraghty’ to the travel agent, and so on; but data are shared extensively between companies, so you could easily end up confusing their systems – not to mention yourself. And if you try to phone up your bank and tell them that you’ve decided to change your mother’s maiden name from Yngstrom to `yGt5r4ad` – or even Smith – then good luck. In fact, given the large number of data breaches, you might as well assume that anyone who wants to can get all your common password recovery information – including your address, your date of birth, your first school and your social security number, as well as your mother’s maiden name.

Some organisations use contextual security information. A bank I once used asks its business customers the value of the last check from their account that was cleared. In theory, this could be helpful: if someone overhears me doing a transaction on the telephone, then it’s not a long-term compromise. The details bear some attention though. When this system was first introduced, I wondered whether a supplier, to whom I’d just written a check, might impersonate me, and concluded that asking for the last three checks’ values would be safer. But the problem we actually had was unexpected. Having given the checkbook to our accountant for the annual audit, we couldn’t talk to the bank. I also don’t like the idea that someone who steals my physical post can also steal my money.

The sheer number of applications demanding a password nowadays exceeds the powers of human memory. A 2007 study by Dinei Florêncio and Cormac Herley of half a million web users over three months showed that the average user has 6.5 passwords, each shared across 3.9 different sites; has about 25 accounts that require passwords; and types an average of 8 passwords per day. Bonneau published more extensive statistics in 2012 [289] but since then the frequency of user password entry has fallen, thanks to smartphones. Modern web browsers also cache passwords; see the discussion of password managers at section 3.4.11 below. But many people use the same password for many different purposes and don’t work out special processes to deal with their high-value logons such as to their bank, their social media accounts and their email. So you have to expect that the password chosen by the customer of the electronic banking system you’ve just designed, may be known to a Mafia-operated porn site as well. (There’s even a website, <http://haveibeenpwned.com>, that will tell you which security breaches have leaked your email address and password.)

One of the most pervasive and persistent errors has been forcing users to change passwords regularly. When I first came across enforced monthly password changes in the 1980s, I observed that it led people to choose passwords such as ‘julia03’ for March, ‘julia04’ for April, and so on, and said as much in the first (2001) edition of this book (chapter 3, page 48). However, in 2003, Bill Burr of NIST wrote password guidelines recommending regular update [1096].

### 3.4. PASSWORDS

---

This was adopted by the Big Four auditors, who pushed it out to all their audit clients<sup>3</sup>. Meanwhile, security usability researchers conducted survey after survey showing that monthly change was suboptimal. The first systematic study by Yinqian Zhang, Fabian Monrose and Mike Reiter of the password transformation techniques users invented showed that in a system with forced expiration, over 40% of passwords could be guessed from previous ones, that forced change didn't do much to help people who chose weak passwords, and that the effort of regular password choice may also have diminished password quality [2070]. Finally a survey was written by usability guru Lorrie Cranor while she was Chief Technologist at the FTC [492], and backed up by an academic study [1505]. In 2017, NIST recanted; they now recommend long passphrases that are only changed on compromise<sup>4</sup>. Other governments' agencies such as Britain's GCHQ followed, and Microsoft finally announced the end of password-expiration policies in Windows 10 from April 2019. However, many firms are caught by the PCI standards set by the credit-card issuers, which haven't caught up and still dictate three-monthly changes; another problem is that the auditors dictate compliance to many companies, and will no doubt take time to catch up.

The current fashion, in 2020, is to invite users to select passphrases of three or more random dictionary words. This was promoted by a famous xkcd cartoon which suggested 'correct horse battery staple' as a password. Empirical research, however, shows that real users select multi-word passphrases with much less entropy than they'd get if they really did select at random from a dictionary; they tend to go for common noun bigrams, and moving to three or four words brings rapidly diminishing returns [296]. The Electronic Frontier Foundation now promotes using dice to pick words; they have a list of 7,776 words ( $6^5$ , so five dice rolls to pick a word) and note that a six-word phrase has 77 bits of entropy and is memorable [290].

#### 3.4.4.4 Operational failures

The most pervasive operational error is failing to reset default passwords. This has been a chronic problem since the early dial access systems in the 1980s attracted attention from mischievous schoolkids. A particularly bad example is where systems have default passwords that can't be changed, checked by software that can't be patched. We see ever more such devices in the Internet of Things; they remain vulnerable for their operational lives. The Mirai botnets have emerged to recruit and exploit them, as I described in Chapter 2.

Passwords in plain sight are another long-running problem, whether on sticky notes or some electronic equivalent. A famous early case was *R v Gold and Schifreen*, where two young hackers saw a phone number for the development version of Prestel, an early public email service run by British Telecom, in a note stuck on a terminal at an exhibition. They dialed in later, and found the welcome screen had a maintenance password displayed on it. They tried this

---

<sup>3</sup>Our university's auditors wrote in their annual report for three years in a row that we should have monthly enforced password change, but couldn't provide any evidence to support this and weren't even aware that their policy came ultimately from NIST. Unimpressed, we asked the chair of our Audit Committee to appoint a new lot of auditors, and eventually that happened.

<sup>4</sup>NIST SP 800-63-3

### 3.4. PASSWORDS

---

on the live system too, and it worked! They proceeded to hack into the Duke of Edinburgh's electronic mail account, and sent mail 'from' him to someone they didn't like, announcing the award of a knighthood. This heinous crime so shocked the establishment that when prosecutors failed to persuade the courts to convict the young men, Britain's parliament passed its first Computer Misuse Act.

A third operational issue is asking for passwords when they're not really needed, or wanted for dishonest reasons, as I discussed at the start of this section. Most of the passwords you're forced to set up on websites are there for marketing reasons – to get your email address or give you the feeling of belonging to a 'club' [294]. So it's perfectly rational for users who never plan to visit that site again to express their exasperation by entering '123456' or even ruder words in the password field.

A fourth is atrocious password management systems: some don't encrypt passwords at all, and there are reports from time to time of enterprising hackers smuggling back doors into password management libraries [427].

But perhaps the biggest operational issue is vulnerability to social-engineering attacks.

#### 3.4.4.5 Social-engineering attacks

Careful organisations communicate security context in various ways to help staff avoid making mistakes. The NSA, for example, had different colored internal and external telephones, and when an external phone in a room is off-hook, classified material can't even be discussed in the room – let alone on the phone.

Yet while many banks and other businesses maintain some internal security context, they often train their customers to act in unsafe ways. Because of pervasive phishing, it's not prudent to try to log on to your bank by clicking on a link in an email, so you should always use a browser bookmark or type in the URL by hand. Yet bank marketing departments send out lots of emails containing clickable links. Indeed much of the marketing industry is devoted to getting people to click on links. Many email clients – including Apple's, Microsoft's, and Google's – make plaintext URLs clickable, so their users may never see a URL that isn't. Bank customers are well trained to do the wrong thing.

A prudent customer should also be cautious if a web service directs them somewhere else – yet bank systems use all sorts of strange URLs for their services. A spam from the Bank of America directed UK customers to `mynew-card.com` and got the certificate wrong (it was for `mynewcard.bankofamerica.com`). There are many more examples of major banks training their customers to practice unsafe computing – by disregarding domain names, ignoring certificate warnings, and merrily clicking links [582]. As a result, even security experts have difficulty telling bank spam from phish [443].

It's not prudent to give out security information over the phone to unidentified callers – yet we all get phoned by bank staff who demand security information. Banks also call us on our mobiles now and expect us to give out security information to a whole train carriage of strangers, rather than letting us text

a response. (I've had a card blocked because a bank security team phoned me while I was driving; it would have been against the law to deal with the call other than in hands-free mode, and there was nowhere safe to stop.) It's also not prudent to put a bank card PIN into any device other than an ATM or a PIN entry device (PED) in a store; and Citibank even asks customers to disregard and report emails that ask for personal information, including PIN and account details. So what happened? You guessed it – it sent its Australian customers an email asking customers 'as part of a security upgrade' to log on to its website and authenticate themselves using a card number and an ATM PIN [1087]. And in one 2005 case, the Halifax sent a spam to the mother of a student of ours who contacted the bank's security department, which told her it was a phish. The student then contacted the ISP to report abuse, and found that the URL and the service were genuine [1241]. The Halifax disappeared during the crash of 2008, and given that their own security department couldn't tell spam from phish, perhaps that was justice (though it cost us taxpayers a shedload of money).

#### 3.4.4.6 Customer education

After phishing became a real threat to online banking in the mid-2000s, banks tried to train their customers to look for certain features in websites. This has been partly risk reduction, but partly risk dumping – seeing to it that customers who don't understand or can't follow instructions can be held responsible for the resulting loss. The general pattern has been that as soon as customers are trained to follow some particular rule, the phishermen exploit this, as the reasons for the rule are not adequately explained.

At the beginning, the advice was 'Check the English', so the bad guys either got someone who could write English, or simply started using the banks' own emails but with the URLs changed. Then it was 'Look for the lock symbol', so the phishing sites started to use SSL (or just forging it by putting graphics of lock symbols on their web pages). Some banks started putting the last four digits of the customer account number into emails; the phishermen responded by putting in the first four (which are constant for a given bank and card product). Next the advice was that it was OK to click on images, but not on URLs; the phishermen promptly put in links that appeared to be images but actually pointed at executables. The advice then was to check where a link would really go by hovering your mouse over it; the bad guys then either inserted a non-printing character into the URL to stop Internet Explorer from displaying the rest, or used an unmanageably long URL (as many banks also did).

This sort of arms race is most likely to benefit the attackers. The countermeasures become so complex and counterintuitive that they confuse more and more users – exactly what the phishermen need. The safety and usability communities have known for years that 'blame and train' is not the way to deal with unusable systems – the only real fix is to design for safe usability in the first place [1451].

#### 3.4.4.7 Phishing warnings

Part of the solution is to give users better tools. Modern browsers alert you to wicked URLs, with a range of mechanisms under the hood. First, there are lists of bad URLs collated by the anti-virus and threat intelligence community. Second, there's logic to look for expired certificates and other compliance failures (as the majority of those alerts are false alarms).

There has been a lot of research, in both industry and academia, about how you get people to pay attention to warnings. We see so many of them, most are irrelevant, and many are designed to shift risk to us from someone else. So when do people pay attention? In our own work, we tried a number of things and found that people paid most attention when the warnings were not vague and general ('*Warning - visiting this web site may harm your computer!*') but specific and concrete ('*The site you are about to visit has been confirmed to contain software that poses a significant risk to you, with no tangible benefit. It would try to infect your computer with malware designed to steal your bank account and credit card details in order to defraud you*') [1327]. Subsequent research by Adrienne Porter Felt and Google's usability team has tried many ideas including making warnings psychologically salient using faces (which doesn't work), simplifying the text (which helps) and making the safe defaults both attractive and prominent (which also helps). Optimising these factors improves compliance from about 35% to about 50% [675]. However, if you want to stop the great majority of people from clicking on known-bad URLs, then voluntary compliance isn't enough. You either have to block them at your firewall, or block them at the browser (as both Chrome and Firefox do for different types of certificate error – a matter to which we'll return in 21.6.1).

#### 3.4.5 System issues

Not all phishing attacks involve psychology. Some involve technical mechanisms to do with password entry and storage together with some broader system issues.

As we already noted, a key question is whether we can restrict the number of password guesses. Security engineers sometimes refer to password systems as 'online' if guessing is limited (as with ATM PINs) and 'offline' if it is not (this originally meant systems where a user could fetch the password file and take it away to try to guess the passwords of other users, including more privileged users). But the terms are no longer really accurate. Some offline systems can restrict guesses, such as payment cards which use physical tamper-resistance to limit you to three PIN guesses, while some online systems cannot. For example, if you log on using Kerberos, an opponent who taps the line can observe your key encrypted with your password flowing from the server to your client, and then data encrypted with that key flowing on the line; so they can take their time to try out all possible passwords. The most common trap here is the system that normally restricts password guesses but then suddenly fails to do so, when it gets hacked and a one-way encrypted password file is leaked, together with the encryption keys. Then the bad guys can try out their entire password dictionary against each account at their leisure.

Password guessability ultimately depends on the entropy of the chosen pass-

### 3.4. PASSWORDS

---

words and the number of allowed guesses, but this plays out in the context of a specific threat model, so you need to consider the type of attacks you are trying to defend against. Broadly speaking, these are as follows.

**Targeted attack on one account:** an intruder tries to guess a specific user's password. They might try to guess a rival's logon password at the office, in order to do mischief directly.

**Attempt to penetrate any account belonging to a specific target:** an enemy tries to hack any account you own, anywhere, to get information that might help take over other accounts, or do harm directly.

**Attempt to penetrate any account on a target system:** the intruder tries to get a logon as any user of the system. This is the classic case of the phisher trying to hack any account at a target bank so he can launder stolen money through it.

**Attempt to penetrate any account on any system:** the intruder merely wants an account at any system in a given domain but doesn't care which one. Examples are bad guys trying to guess passwords on any online email service so they can send spam from the compromised account, and a targeted attacker who wants a logon to any random machine in the domain of a target company as a beachhead.

**Attempt to use a breach of one system to penetrate a related one:** the intruder has got a beachhead and now wants to move inland to capture higher-value targets.

**Service denial attack:** the attacker may wish to block one or more legitimate users from using the system. This might be targeted on a particular account or system-wide.

This taxonomy helps us ask relevant questions when evaluating a password system.

#### 3.4.6 Can you deny service?

There are basically three ways to deal with password guessing when you detect it: lockout, throttling, and protective monitoring. Banks may freeze your card after three wrong PINs; but if they freeze your online account after three bad password attempts they open themselves up to a denial-of-service attack. Service can also fail by accident; poorly-configured systems can generate repeat fails with stale credentials. So many commercial websites nowadays use throttling rather than lockout. In a military system, you might not want even that, in case an enemy who gets access to the network could jam it with a flood of false logon attempts. In this case, protective monitoring might be the preferred option, with a plan to abandon rate-limiting if need be in a crisis. Joe Bonneau and Soren Preibusch collected statistics of how many major websites use account locking versus various types of rate control [294]. They found that popular, growing, competent sites tend to be more secure, as do payment sites, while

content sites do worst. Microsoft Research’s Yuan Tian, Cormac Herley and Stuart Schechter investigated how to do locking or throttling properly; among other things, it’s best to penalise guesses of weak passwords (as otherwise an attacker gets advantage by guessing them first), to be more aggressive when protecting users who have selected weak passwords, and to not punish IPs or clients that repeatedly submit the same wrong password [1888].

#### 3.4.7 Protecting oneself or others?

Next, to what extent does the system need to protect users and subsystems from each other? In global systems on which anyone can get an account – such as mobile phone systems and cash machine systems – you must assume that the attackers are already legitimate users, and see to it that no-one can use the service at someone else’s expense. So knowledge of one user’s password will not allow another user’s account to be compromised. This has both personal aspects, and system aspects.

On the personal side, don’t forget what we said about intimate partner abuse in 2.5.4: the passwords people choose are often easy for their spouses or partners to guess, and the same goes for password recovery questions: so some thought needs to be given to how abuse victims can recover their security.

On the system side, there are all sorts of passwords used for mutual authentication between subsystems, few mechanisms to enforce password quality in server-server environments, and many well-known issues (for example, the default password for the Java trusted keystore file is ‘changeit’). Development teams often share passwords that end up in live systems, even 30 years after this practice led to the well-publicised hack of the Duke of Edinburgh’s email described in section 3.4.4.4. Within a single big service firm you can lock stuff down by having named crypto keys and seeing to it that each name generates a call to an underlying hardware security module; or you can even use mechanisms like SGX to tie keys to known software. But that costs real money, and money isn’t the only problem. Enterprise system components are often hosted at different service companies, which makes adoption of better practices a hard coordination problem too. As a result, server passwords often appear in scripts or other plaintext files, which can end up in Dropbox or Splunk. So it is vital to think of password practices beyond end users. In later chapters we’ll look at protocols such as Kerberos and ssh; for now, recall Ed Snowden’s remark that it was trivial to hack the typical large company: just spear-phish a sysadmin and then chain your way in. Much of this chapter is about the ‘spear-phish a sysadmin’ part; but don’t neglect the ‘chain your way in’ part.

#### 3.4.8 Attacks on password entry

Password entry is often poorly protected.

#### 3.4.8.1 Interface design

Thoughtless interface design is all too common. Some common makes of cash machine have a vertical keyboard at head height, making it simple for a pick-pocket to watch a woman enter her PIN before lifting her purse from her handbag. The keyboards may have been at a reasonable height for the men who designed them, but women who are a few inches shorter are exposed.

When entering a card number or PIN in a public place, I usually cover my typing hand with my body or my other hand – but you can't assume that all your customers will. Many people are uncomfortable shielding a PIN as it's a signal of distrust, especially if they're in a supermarket queue and a friend is standing nearby. UK banks found that 20% of users never shield their PIN [127] – and then used this to blame customers whose PINs were compromised by an overhead CCTV camera, rather than designing better PIN entry devices.

#### 3.4.8.2 Trusted path, and bogus terminals

A *trusted path* is some means of being sure that you're logging into a genuine machine through a channel that isn't open to eavesdropping. False terminal attacks go back to the dawn of time-shared computing. A public terminal would be left running an attack program that looks just like the usual logon screen – asking for a user name and password. When an unsuspecting user did this, it would save the password, reply ‘sorry, wrong password’ and then vanish, invoking the genuine password program. The user assumed they'd made a typing error and just entered the password again. This is why Windows had a *secure attention sequence*; hitting `ctrl-alt-del` was guaranteed to take you to a genuine password prompt. But eventually, in Windows 10, this got removed to prepare the way for Windows tablets, and because almost nobody understood it.

ATM skimmers are devices that sit on an ATM's throat, copy card details, and have a camera to record the customer PIN. There are many variants on the theme. Fraudsters deploy bad PIN entry devices too, and have even been jailed for attaching password-stealing hardware to terminals in bank branches. I'll describe this world in much more detail in the chapter on banking and bookkeeping; the long-term solution has been to move from magnetic-strip cards that are easy to copy to chip cards that are much harder. In any case, if a terminal might contain malicious hardware or software, then passwords alone will not be enough.

#### 3.4.8.3 Technical defeats of password retry counters

Many kids find out that a bicycle combination lock can usually be broken in a few minutes by solving each ring in order of looseness. The same idea worked against a number of computer systems. The PDP-10 TENEX operating system checked passwords one character at a time, and stopped as soon as one of them was wrong. This opened up a *timing attack*: the attacker would repeatedly place a guessed password in memory at a suitable location, have it verified as part of a file access request, and wait to see how long it took to be rejected [1129]. An

error in the first character would be reported almost at once, an error in the second character would take a little longer to report, and in the third character a little longer still, and so on. So you could guess the characters one after another, and instead of a password of  $N$  characters drawn from an alphabet of  $A$  characters taking  $A^N/2$  guesses on average, it took  $AN/2$ . (Bear in mind that in thirty years' time, all that might remain of the system you're building today is the memory of its more newsworthy security failures.)

These same mistakes are being made all over again in the world of embedded systems. With one remote car locking device, as soon as a wrong byte was transmitted from the key fob, the red telltale light on the receiver came on. With some smartcards, it has been possible to determine the customer PIN by trying each possible input value and looking at the card's power consumption, then issuing a reset if the input was wrong. The reason was that a wrong PIN caused a PIN retry counter to be decremented, and writing to the EEPROM memory which held this counter caused a current surge of several millamps – which could be detected in time to reset the card before the write was complete [1105]. These implementation details matter. Timing channels are a serious problem for people implementing cryptography, as we'll discuss at greater length in the next chapter.

A recent high-profile issue was the PIN retry counter in the iPhone. My colleague Sergei Skorobogatov noted that the iPhone keeps sensitive data encrypted in flash memory, and built an adapter that enabled him to save the encrypted memory contents and restore them to their original condition after several PIN attempts. This enabled him to try all 10,000 possible PINs rather than the ten PINs limit that Apple tried to impose [1777]<sup>5</sup>.

#### 3.4.9 Attacks on password storage

Passwords have often been vulnerable where they are stored. In MIT's 'Compatible Time Sharing System' `ctss` – a 1960s predecessor of Multics – it once happened that one person was editing the message of the day, while another was editing the password file. Because of a software bug, the two editor temporary files got swapped, and everyone who logged on was greeted with a copy of the password file! [476].

Another horrible programming error struck a UK bank in the late 1980s, which issued all its customers with the same PIN by mistake [54]. As the procedures for handling PINs meant that no one in the bank got access to anyone's PIN other than their own, the bug wasn't spotted until after thousands of customer cards had been shipped. Big blunders continue: in 2019 the security company that does the Biostar and AEOS biometric lock system for building entry control and whose customers include banks and police forces in 83 countries left a database unprotected online with over a million people's IDs, plaintext passwords, fingerprints and facial recognition data; security researchers who discovered this from an Internet scan were able to add themselves as users [1864].

---

<sup>5</sup>This was done to undermine an argument by then FBI Director James Comey that the iPhone was unhackable and so Apple should be ordered to produce an operating system upgrade that created a backdoor; see section 26.2.8.

Auditing provides another hazard. When systems log failed password attempts, the log usually contains a large number of passwords, as users get the ‘username, password’ sequence out of phase. If the logs are not well protected then someone who sees an audit record of a failed login with a non-existent user name of `e5gv*8yp` just has to try this as a password for all the valid user names.

#### 3.4.9.1 One-way encryption

Such incidents taught people to protect passwords by encrypting them using a one-way algorithm, an innovation due to Roger Needham and Mike Guy. The password, when entered, is passed through a one-way function and the user is logged on only if it matches a previously stored value. However, it’s often implemented wrong. The right way to do it is to generate a random key, historically known in this context as a *salt*; combine the password with the salt using a slow, cryptographically strong one-way function; and store both the salt and the hash.

#### 3.4.9.2 Password cracking

Some systems that use an encrypted password file make it widely readable. Unix used to be the prime example – the password file `/etc/passwd` was readable by all users. So any user could fetch it and try to break passwords by encrypting all the passwords in a dictionary and comparing them with the encrypted values in the file. We already mentioned in 3.4.4.1 the ‘Crack’ software that people have used for years for this purpose.

Most modern operating systems have sort-of fixed this problem; in modern Linux distributions, for example, passwords are salted, hashed using 5000 rounds of SHA-512, and stored in a file that only the root user can read. But there are still password-recovery tools to help you if, for example, you’ve encrypted an Office document with a password you’ve forgotten [1674]. Such tools can also be used by a crook who has got root access, and there are still lots of badly designed systems out there where the password file is vulnerable in other ways.

There is also *credential stuffing*: when a system is hacked and passwords are cracked (or were even found unencrypted), they are then tried out on other systems to catch the many people who reused them. This remains a live problem. So password cracking is still worth some attention. One countermeasure worth considering is deception, which can work at all levels in the stack. You can have honeypot systems that alarm if anyone ever logs on to them, honeypot accounts on a system, or password canaries – bogus encrypted passwords for genuine accounts [996].

#### 3.4.9.3 Remote password checking

Many systems check passwords remotely, using cryptographic protocols to protect the password in transit, and the interaction between password security and network security can be complex. Local networks often use a protocol called Kerberos, where a server sends you a key encrypted under your password; if you

### 3.4. PASSWORDS

---

know the password you can decrypt the key and use it to get tickets that give you access to resources. I'll discuss this in the next chapter, in section 4.7.4; it doesn't always protect weak passwords against an opponent who can wiretap encrypted traffic. Web servers mostly use a protocol called TLS to encrypt your traffic from the browser on your phone or laptop; I discuss TLS in the following chapter, in section 5.7.5. TLS does not protect you if the server gets hacked. However there is a new protocol called Simultaneous Authentication of Equals (SAE) which is designed to set up secure sessions even where the password is guessable, and which has been adopted from 2018 in the WPA3 standard for WiFi authentication. I'll discuss this later too.

And then there's OAuth, a protocol which allows access delegation, so you can grant one website the right to authenticate you using the mechanisms provided by another. Developed by Twitter from 2006, it's now used by the main service providers such as Google, Microsoft and Facebook to let you log on to media and other sites; an authorisation server issues access tokens for the purpose. We'll discuss the mechanisms later too. The concomitant risk is cross-site attacks; we are now (2019) seeing OAuth being used by state actors in authoritarian countries to phish local human-rights defenders. The technique is to create a malicious app with a plausible name (say 'Outlook Security Defender') and send an email, purportedly from Microsoft, asking for access. If the target responds they end up at a Microsoft web page where they're asked to authorise the app to have access to their data [46].

#### 3.4.10 Absolute limits

If you have confidence in the cryptographic algorithms and operating-system security mechanisms that protect passwords, then the probability of a successful password guessing attack is a function of the entropy of passwords, if they are centrally assigned, and the psychology of users if they're allowed to choose them. Military sysadmins often prefer to issue random passwords, so the probability of password guessing attacks can be managed. For example, if  $L$  is the maximum password lifetime,  $R$  is login attempt rate,  $S$  is the size of the password space, then the probability that a password can be guessed in its lifetime is  $P = LR/S$ , according to the US Department of Defense password management guideline [546].

There are issues with such a 'provable security' doctrine, starting with the attackers' goal. Do they want to crack a target account, or just any account? If an army has a million possible passwords and a million users, and the alarm goes off after three bad password attempts on any account, then the attacker can just try one password for every different account. If you want to stop this, you have to do rate control not just for every account, but for all accounts.

To take a concrete example, Unix systems used to be limited to eight character passwords, so there were  $96^8$  or about  $2^{52}$  possible passwords. Some UK government systems used to issue passwords randomly selected with a fixed template of consonants, vowels and numbers designed to make them easier to remember, such as CVCNCVCN (e.g. fuR5xEb8). If passwords are not case sensitive, the guess probability is cut drastically, to only one in  $21^4 \cdot 5^2 \cdot 10^2$  or about  $2^{-29}$ . So if an attacker could guess 100 passwords a second – perhaps

### 3.4. PASSWORDS

---

distributed across 10,000 accounts on hundreds of machines on a network, so as not to raise the alarm – then they would need about 5 million seconds, or two months, to get in. If you’re defending such a system, you might find it prudent to do rate control: set a limit of say one password guess per ten seconds per user account, and perhaps by source IP address. You might also count the failed logon attempts and analyse them: is there a constant series of guesses that suggests an attacker using a botnet, or some other attempted intrusion? And what will you do once you notice one? Will you close the system down? Welcome back to the world of service denial.

With a commercial website, 100 passwords per second may translate to one compromised user account per second, because of poor user password choices. That may not be a big deal for a web service with 100 million accounts – but it may still be worth trying to identify the source of any industrial-scale password-guessing attacks. If they’re from a small number of IP addresses, you can block them, but doing this properly is harder than it looks, as we noted in section 3.4.6 above. And if an automated guessing attack does persist, then another way of dealing with it is the CAPTCHA, which I’ll describe in section 3.5.

#### 3.4.11 Using a password manager

Since the 1980s, companies have been selling single sign-on systems that remember your passwords for multiple applications, and when browsers came along in the mid-1990s and people started logging into dozens of websites, password managers became a mass-market product. Browser vendors noticed, and started providing much the same functionality for free.

Choosing random passwords and letting your browser remember them can be a pragmatic way of operating. The browser will only enter the password into a web page with the right URL (IE) or the same hostname and field name (Firefox). Browsers let you set a master password, which encrypts all the individual site passwords and which you only have to enter when your browser is updated. The main drawbacks of password managers in general are that you might forget the master password; and that all your passwords may be compromised at once, since malware writers can work out how to hack common products. This is a particular issue when using a browser, and another is that a master password is not always the default so many users don’t set one. (The same holds for other security services you get as options with platforms, such as encrypting your phone or laptop.) An advantage of using the browser is that you may be able to sync passwords between the browser in your phone and that in your laptop.

Third-party password managers can offer more, such as choosing long random passwords for you, identifying passwords shared across more than one website, and providing more controllable ways for you to manage the backup and recovery of your password collection. (With a browser, this comes down to backing up your whole laptop or phone.) They can also help you track your accounts, so you can see whether you had a password on a system that’s announced a breach. The downside is that many products are truly dreadful, with even some hardware password managers storing all your secrets in the clear [130], while the top five software products suffer from serious and systemic vulnerabilities, from autocomplete to ignoring subdomains [389]. How do you know that

### 3.4. PASSWORDS

---

any given product is actually sound?

Many banks try to disable storage, whether by setting `autocomplete="off"` in their web pages or using other tricks that block password managers too. Banks think this improves security, but I'm not at all convinced. Stopping people using password managers or the browser's own storage will probably make most of them use weaker passwords. The banks may argue that killing autocomplete makes compromise following device theft harder, and may stop malware stealing the password from the database of your browser or password manager, but the phishing defence provided by that product is disabled – which may expose the average customer to greater risk [1355]. It's also inconvenient; one bank that suddenly disabled password storage had to back down the following day, because of the reaction from customers [1278]. People manage risk in all sorts of ways. I personally use different browsers for different purposes, and let them store low-value passwords; for important accounts, such as email and banking, I always enter passwords manually, and always navigate to them via bookmarks rather than by clicking on links. But most people are less careful. And be sure to think through backup and recovery, and exercise it to make sure it works. What happens when your laptop dies? When your phone dies? When someone persuades your phone company to link your phone number to their SIM? When you die – or when you fall ill and your partner needs to manage your stuff? Do they know where to find the master passwords? Writing them down in a book can make sense, if all you (and your executor) have to remember is ‘page 169, Great Expectations.’ Writing them down in a diary you tote with you, on a page saying ‘passwords’, is not so great. Very few people get all this right.

#### 3.4.12 Will we ever get rid of passwords?

Passwords are annoying, so many people have discussed getting rid of them, and the move from laptops to phones gives us a chance. The proliferation of IoT devices that don't have keyboards will force us to do without them for some purposes. A handful of firms have tried to get rid of them completely. One example is the online bank Monzo, which operates exclusively via an app. They leave it up to the customer whether they protect their phone using a fingerprint, a pattern lock, a PIN or a password. However they still use email to prompt people to upgrade, and to authenticate people who buy a new phone, so account takeover involves either phone takeover, or guessing a password or a password recovery question. The most popular app that uses SMS to authenticate rather than a password may be WhatsApp. I expect that this will become more widespread; so we'll see more attacks based on phone takeover, from SIM swaps through Android malware, SS7 and RCS hacking, to simple physical theft. In such cases, recovery often means an email loop, making your email password more critical than ever – or phoning a call centre and telling them your mother's maiden name. So things may change less than they seem.

Joe Bonneau and colleagues analysed the options in 2012 [292]. There are many criteria against which an authentication system can be evaluated, and we've worked through them here: resilience to theft, to physical observation, to guessing, to malware and other internal compromise, to leaks from other verifiers, to phishing and to targeted impersonation. Other factors include ease

### 3.4. PASSWORDS

---

of use, ease of learning, whether you need to carry something extra, error rate, ease of recovery, cost per user, and whether it's an open design that anyone can use. They concluded that most of the schemes involving net benefits were variants on single sign-on – and OpenID has indeed become widespread, with many people logging in to their newspaper using Google or Facebook, despite the obvious privacy cost<sup>6</sup>. Beyond that, any security improvements involve giving up one or more of the benefits of passwords, namely that they're easy, efficient and cheap.

Bonneau's survey gave high security ratings to physical authentication tokens such as the CAP reader, which enables people to use their bank cards to log on to online banking; bank regulators have already mandated two-factor authentication in a number of countries. Using something tied to a bank card gives a more traditional root of trust, at least with traditional high-street banks; a customer can walk into a branch and order a new card<sup>7</sup>. Firms that are targets of state-level attackers, such as Google and Microsoft, now give authentication tokens of some kind or another to all their staff.

Did the survey miss anything? Well, the old saying is ‘something you have, something you know, or something you are’ – or, as Simson Garfinkel engagingly puts it, ‘something you had once, something you've forgotten, or something you once were’. The third option, biometrics, has started coming into wide use since high-end mobile phones started offering fingerprint readers. Some countries, like Germany, issue their citizens with ID cards containing a fingerprint, which may provide an alternate root of trust for when everything else goes wrong. We'll discuss biometrics in its own chapter later in the book.

Both tokens and biometrics are still mostly used with passwords, first as a backstop in case a device gets stolen, and second as part of the process of security recovery. So passwords remain the (shaky) foundation on which much of information security is built. What may change this is the growing number of devices that have no user interface at all, and so have to be authenticated using other mechanisms. One approach that's getting ever more common is trust on first use, also known as the ‘resurrecting duckling’ after the fact that a duckling bonds on the first moving animal it sees after it hatches. We'll discuss this in the next chapter, and also when we dive into specific applications such as security in vehicles.

Finally, you should think hard about how to authenticate customers or other people who exercise their right to demand copies of their personal information under data-protection law. In 2019, James Pavur sent out 150 such requests to companies, impersonating his fiancée [1886]. 86 firms admitted they had infor-

<sup>6</sup>Government attempts to set up single sign-on for public services have been less successful, with the UK ‘Verify’ program due to be shuttered in 2020 [1392]. There have been many problems around attempts to entrench government’s role in identity assurance, which I’ll discuss further in the chapter on biometrics, and which spill over into issues from online services to the security of elections. It was also hard for other private-sector firms to compete because of the network effects enjoyed by incumbents. However in 2019 Apple announced that it would provide a new, more privacy-friendly single sign-on mechanism, and use the market power of its app store to force websites to support it. Thus the quality and nature of privacy on offer is becoming a side-effect of battles fought for other motives. We'll analyse this in more depth in the chapter on economics.

<sup>7</sup>This doesn’t work for branchless banks like Monzo; but they do take a video of you when you register so that their call centre can recognise you later.

mation about her, and many had the sense to demand her logon and password to authenticate her. But about a quarter were prepared to accept an email address or phone number as authentication; and a further 16 percent asked for easily forgeable ID. He collected full personal information about her, including her credit card number, her social security number and her mother's maiden name. A threat intelligence firm with which she'd never interacted sent a list of her accounts and passwords that had been compromised. Given that firms face big fines in the EU if they don't comply with such requests within 30 days, you'd better work out in advance how to cope with them, rather than leaving it to an assistant in your law office to improvise a procedure. If you abolish passwords, and a former customer claims their phone was stolen, what do you do then? And if you hold personal data on people who have never been your customers, how do you identify them?

### 3.5 CAPTCHAs

Can we have protection mechanisms that use the brain's strengths rather than its weaknesses? The most successful innovation in this field is probably the CAPTCHA – the ‘Completely Automated Public Turing Test to Tell Computers and Humans Apart’. These are the little visual puzzles that you often have to solve to post to a blog, to register for a free online account, or to recover a password. The idea is that people can solve such problems easily, while computers find them hard.

CAPTCHAs first came into use in a big way in 2003 to stop spammers using scripts to open thousands of accounts on free email services, and to make it harder for attackers to try a few simple passwords with each of a large number of existing accounts. They were invented by Luis von Ahn and colleagues [1969], who were inspired by the test famously posed by Alan Turing as to whether a computer was intelligent: you put a computer in one room and a human in another, and invite a human to try to tell them apart. The test is turned round so that a computer can tell the difference between human and machine.

Early versions set out to use a known ‘hard problem’ in AI such as the recognition of distorted text against a noisy background. The idea is that breaking the CAPTCHA was equivalent to solving the AI problem, so an attacker would actually have to do the work by hand, or come up with a real innovation in computer science. Humans were good at reading distorted text, while programs were less good. It turned out to be harder than it seemed. A lot of the attacks on CAPTCHAs, even to this day, exploit the implementation details.

Many of the image recognition problems posed by early systems also turned out not to be too hard at all once smart people tried hard to solve them. There are also protocol-level attacks; von Ahn mentioned that in theory a spammer could get people to solve them as the price of access to free porn [1968]. This soon started to happen: spammers created a game in which you undress a woman by solving one CAPTCHA after another [191]. Within a few years, we saw commercial CAPTCHA-breaking tools arriving on the market [843]. Within a few more, generic attacks using signal-processing techniques inspired by the human visual system had become fairly efficient at solving at least a subset

### **3.6. SUMMARY**

---

of most types of text CAPTCHA [746]. And security-economics research in underground markets has shown that by 2011 the action had moved to using humans; people in countries with incomes of a few dollars a day will solve CAPTCHAs for about 50c per 1000.

From 2014, the CAPTCHA has been superseded by the ReCAPTCHA, another of Luis von Ahn’s inventions. Here the idea is to get a number of users to do some useful piece of work, and check their answers against each other. The service initially asked people to transcribe fragments of text from Google books that confused OCR software; more recently you get a puzzle with eight pictures asking ‘click on all images containing a shop front’, which helps Google train its vision-recognition AI systems<sup>8</sup>. It pushes back on the cheap-labour attack by putting up two or three multiple-choice puzzles and taking tens of seconds over it, rather than allowing rapid responses.

The implementation of CAPTCHAs is often thoughtless, with accessibility issues for users who are visually impaired. And try paying a road toll in Portugal where the website throws up a CAPTCHA asking you to identify pictures with an object, if you can’t understand Portuguese well enough to figure out what you’re supposed to look for!

## **3.6 Summary**

Psychology matters to the security engineer, because of deception and because of usability. Most real attacks nowadays target the user. Various kinds of phishing are the main national-security threat, the principal means of developing and maintaining the cybercrime infrastructure, and one of the principal threats to online banking systems. Other forms of deception account for much of the rest of the cybercrime ecosystem, which is roughly equal to legacy crime in both volume and value.

Part of the remedy is security usability, yet research in this field was long neglected, being seen as less glamorous than cryptography or operating systems. That was a serious error on our part, and from the mid-2000s we have started to realise the importance of making it easier for ordinary people to use systems in safe ways. Since the mid-2010s we’ve also started to realise that we also have to make things easier for ordinary programmers; many of the security bugs that have broken real systems have been the result of tools that were just too hard to use, from cryptographic APIs that used unsafe defaults to the C programming language. Getting usability right also helps business directly: PayPal has built a \$100bn business through being a safer and more convenient way to shop online<sup>9</sup>.

In this chapter, we took a whistle-stop tour through psychology research relevant to deception and to the kinds of errors people make, and then tackled authentication as a case study. Much of the early work on security usability focused on password systems, which raise dozens of interesting questions. We

---

<sup>8</sup>There’s been pushback from users who see a ReCAPTCHA saying ‘click on all images containing a helicopter’ and don’t want to help in military AI research. Google’s own staff protested at this research too and the program was discontinued. But other users still object to working for Google for free.

<sup>9</sup>Full disclosure: I consult for them.

### **3.6. SUMMARY**

---

now have more and more data not just on things we can measure in the lab such as guessability, memorability, and user trainability, but also on factors that can only be observed in the field such as how real systems break, how real attacks scale and how the incentives facing different players lead to unsafe equilibria.

At the end of the first workshop on security and human behavior in 2008, the psychologist Nick Humphrey summed up a long discussion on risk. “We’re all agreed,” he said, “that people pay too much attention to terrorism and not enough to cybercrime. But to a psychologist this is obvious. If you want people to be more relaxed in airports, take away the tanks and guns, put in some nice sofas and Mozart in the loudspeakers, and people will relax soon enough. And if you want people to be more wary online, make everyone use Jaws as their screen saver. But that’s not going to happen as the computer industry goes out of its way to make computers seem a lot less scary than they used to be.” And of course governments want people to be anxious about terrorism, as it bids up the police budgets and helps politicians get re-elected. So we give people the wrong signals as well as spending our money on the wrong things. Understanding the many tensions between the demands of psychology, economics and engineering is essential to building robust systems at global scale.

## **Research problems**

Security psychology is one of the hot topics in 2020. In the second edition of this book, I noted that the whole field of security economics had sprung into life since the first edition in 2001, and wrote ‘We also need more fundamental thinking about the relationship between psychology and security’. Security usability has become a discipline too, with the annual Symposium on Usable Privacy and Security, and we’ve been running workshops to bring security engineers together with anthropologists, psychologists, philosophers and others who work on risk and how people cope with it.

My meta-algorithm for finding research topics is to look first at applications and then at neighbouring disciplines. An example of the first is safe usability: as safety-critical products from cars to medical devices acquire not just software and Internet connections, but complex interfaces and even their own apps, how can we design them so that they won’t harm people by accident, or as a result of malice?

An example of the second, and the theme of the Workshop on Security and Human Behaviour, is what we can learn from disciplines that study how people deal with risk, ranging from anthropology and psychology to sociology, history and philosophy. Our 2020 event is hosting leading criminologists. The pandemic now suggests that maybe we should work with architects too. They’re now working out how people can be physically distant but socially engaged, and their skill is understanding how form facilitates human experience and human interaction. There’s more to design than just hacking code.

## Further reading

The Real Hustle videos are probably the best tutorial on deception; a number of episodes are on YouTube. Meanwhile, the best book on social engineering is still Kevin Mitnick's '*The Art of Deception*' [1325]. Amit Katwala wrote a short survey of deception detection technologies [1024] while Tony Docan-Morgan has edited a 2019 handbook on the state of deception research with 51 chapters by specialists on its many aspects [569].

For how social psychology gets used and abused in marketing, the must-read book is Tim Wu's '*The Attention Merchants*' which tells the history of advertising [2050].

In the computer science literature, perhaps a good starting point is James Reason's '*Human Error*', which tells us what the safety-critical systems community has learned from many years studying the cognate problems in their field [1589]. Then there are standard HCI texts such as [1544], while early papers on security usability appeared as [493] and on phishing appeared as [976]. As we move to a world of autonomous devices, there is a growing body of research on how we can get people to trust robots more by Disneyfication – for example, giving library robots eyes that follow the direction of travel, and making them chirp with happiness when they help a customer [1687]. Similar research on autonomous vehicles shows that people trust such vehicles more if they're given some personality, and the passengers are given some strategic control such as the ability to select routes or even just to order the car to stop.

As for behavioral economics, I get my students to read Danny Kahneman's Nobel prize lecture. For more technical detail, there's a volume of papers Danny edited just before that with Tom Gilovich and Dale Griffin [769], or the pop science book '*Thinking, Fast and Slow*' that he wrote afterwards [1005]. An alternative view, which gives the whole history of behavioral economics, is Dick Thaler's '*Misbehaving: The Making of Behavioural Economics*' [1874]. For the applications of this theory in government and elsewhere, the standard reference is Dick Thaler and Cass Sunstein's '*Nudge*' [1876]. Dick's later second thoughts about 'Sludge' are at [1875].

For a detailed history of passwords and related mechanisms, as well as many empirical results and an analysis of statistical techniques for measuring both guessability and recall, I strongly recommend Joe Bonneau's thesis [289], a number of whose chapters ended up as papers I cited above.

Finally, if you're interested in the dark side, '*The Manipulation of Human Behavior*' by Albert Biderman and Herb Zimmer reports experiments on interrogation carried out after the Korean War with US Government funding [239]. Known as the Torturer's Bible, it describes the relative effectiveness of sensory deprivation, drugs, hypnosis, social pressure and so on when interrogating and brainwashing prisoners. As for the polygraph and other deception-detection techniques used nowadays, the standard reference is by Aldert Vrij [1970].

# Chapter 5

# Cryptography

ZHQM ZMGM ZMFM  
– G JULIUS CAESAR

KXJEY UREBE ZWEHE WRYTU HEYFS KREHE GOYFI  
WTTTU OLKSY CAJPO BOTEI ZONTX BYBWT GONEY  
CUZWR GDSON SXBOU YWRHE BAAHY USEDQ  
– JOHN F KENNEDY

## 5.1 Introduction

Cryptography is where security engineering meets mathematics. It gives us the tools that underlie most modern security protocols. It is the key technology for protecting distributed systems, yet it is surprisingly hard to do right. As we've already seen in Chapter 4, "Protocols," cryptography has often been used to protect the wrong things, or to protect them in the wrong way. Unfortunately, the available crypto tools aren't always very usable.

But no security engineer can ignore cryptology. A medical friend once told me that while she was young, she worked overseas in a country where, for economic reasons, they'd shortened their medical degrees and concentrated on producing specialists as quickly as possible. One day, a patient who'd had both kidneys removed and was awaiting a transplant needed her dialysis shunt redone. The surgeon sent the patient back from the theater on the grounds that there was no urinalysis on file. It just didn't occur to him that a patient with no kidneys couldn't produce any urine.

Just as a doctor needs to understand physiology as well as surgery, so a security engineer needs to be familiar with at least the basics of crypto (and much else). There are, broadly speaking, three levels at which one can approach crypto. The first consists of the underlying intuitions; the second of the mathematics that we use to clarify these intuitions, provide security proofs where possible and tidy up the constructions that cause the most confusion; and the third is the cryptographic engineering – the tools we commonly use, and the

experience of what can go wrong with them. In this chapter, I assume you have no training in crypto and set out to explain the basic intuitions. I illustrate them with engineering, and sketch enough of the mathematics to help give you access to the literature when you need it. One reason you need some crypto know-how is that many common constructions are confusing, and many tools offer unsafe defaults. For example, Microsoft’s Crypto API (CAPI) nudges engineers to use electronic codebook mode; by the end of this chapter you should understand what that is, why it’s bad, and what you should do instead.

Many crypto textbooks assume that their readers are pure maths graduates, so let me start off with non-mathematical definitions. *Cryptography* refers to the science and art of designing ciphers; *cryptanalysis* to the science and art of breaking them; while *cryptology*, often shortened to just crypto, is the study of both. The input to an encryption process is commonly called the *plaintext* or *cleartext*, and the output the *ciphertext*. Thereafter, things get somewhat more complicated. There are a number of basic building blocks, such as *block ciphers*, *stream ciphers*, and *hash functions*. Block ciphers may either have one key for both encryption and decryption, in which case they’re called *shared-key* (also *secret-key* or *symmetric*), or have separate keys for encryption and decryption, in which case they’re called *public-key* or *asymmetric*. A *digital signature scheme* is a special type of asymmetric crypto primitive.

I will first give some historical examples to illustrate the basic concepts. I’ll then fine-tune definitions by introducing the security models that cryptologists use, including perfect secrecy, concrete security, indistinguishability and the random oracle model. Finally, I’ll show how the more important cryptographic algorithms actually work, and how they can be used to protect data. En route, I’ll give examples of how people broke weak ciphers, and weak constructions using strong ciphers.

## 5.2 Historical Background

Suetonius tells us that Julius Caesar enciphered his dispatches by writing ‘D’ for ‘A’, ‘E’ for ‘B’ and so on [1843]. When Augustus Caesar ascended the throne, he changed the imperial cipher system so that ‘C’ was now written for ‘A’, ‘D’ for ‘B’ etcetera. In modern terminology, we would say that he changed the key from ‘D’ to ‘C’. Remarkably, a similar code was used by Bernardo Provenzano, allegedly the *capo di tutti capi* of the Sicilian mafia, who wrote ‘4’ for ‘a’, ‘5’ for ‘b’ and so on. This led directly to his capture by the Italian police in 2006 after they intercepted and deciphered some of his messages [1535].

The Arabs generalised this idea to the *monoalphabetic substitution*, in which a keyword is used to permute the cipher alphabet. We will write the plaintext in lower case letters, and the ciphertext in upper case, as shown in Figure 5.1:

abcdefghijklmnopqrstuvwxyz
SECURITYABDFGHJKLMNOPQVWXZ

Figure 5.1 – monoalphabetic substitution cipher

OYAN RWSGKFR AN AH RHTFANY MSOYRM OYSH SMSEAC NCMAKO; but it's a pencil and paper puzzle to break ciphers of this kind. The trick is that some letters, and combinations of letters, are much more common than others; in English the most common letters are e,t,a,i,o,n,s,h,r,d,l,u in that order. Artificial intelligence researchers have experimented with programs to solve monoalphabetic substitutions. Using letter and digram (letter pair) frequencies alone, they typically need about 600 letters of ciphertext; smarter strategies such as guessing probable words can cut this to about 150 letters; and state-of-the-art systems that use neural networks and approach the competence of human analysts are also tested on deciphering ancient scripts such as Ugaritic and Linear B [1194].

There are basically two ways to make a stronger cipher – the *stream cipher* and the *block cipher*. In the former, you make the encryption rule depend on a plaintext symbol's position in the stream of plaintext symbols, while in the latter you encrypt several plaintext symbols at once in a block.

### 5.2.1 An early stream cipher – the Vigenère

This early stream cipher is commonly ascribed to the Frenchman Blaise de Vigenère, a diplomat who served King Charles IX. It works by adding a key repeatedly into the plaintext using the convention that 'A' = 0, 'B' = 1, ..., 'Z' = 25, and addition is carried out modulo 26 – that is, if the result is greater than 25, we subtract as many multiples of 26 as are needed to bring it into the range [0, ..., 25], that is, [A, ..., Z]. Mathematicians write this as

$$C = P + K \bmod 26$$

So, for example, when we add P (15) to U (20) we get 35, which we reduce to 9 by subtracting 26. 9 corresponds to J, so the encryption of P under the key U (and of U under the key P) is J, or more simply  $U + P = J$ . In this notation, Julius Caesar's system used a fixed key  $K = D$ , while Augustus Caesar's used  $K = C$  and Vigenère used a repeating key, also known as a *running key*. Techniques were developed to do this quickly, ranging from printed tables to brass cipher wheels. Whatever the technology, the encryption using a repeated keyword for the key would look as shown in Figure 5.2:

<i>Plain</i>	tobeornottobethatisthequestion
<i>Key</i>	runrunrunrunrunrunrunrunrunrun
<i>Cipher</i>	KIOVIEEIGKIOVNURNVJNUVKHVMGZIA

Figure 5.2 – Vigenère (polyalphabetic substitution cipher)

A number of people appear to have worked out how to solve polyalphabetic ciphers, from the womaniser Giacomo Casanova to the computing pioneer Charles Babbage. But the first published solution was in 1863 by Friedrich Kasiski, a Prussian infantry officer [1020]. He noticed that given a long enough piece of ciphertext, repeated patterns will appear at multiples of the keyword length.

In Figure 5.2, for example, we see ‘KIOV’ repeated after nine letters, and ‘NU’ after six. Since three divides both six and nine, we might guess a keyword of three letters. Then ciphertext letters one, four, seven and so on were all enciphered under the same keyletter; so we can use frequency analysis techniques to guess the most likely values of this letter, and then repeat the process for the remaining letters of the key.

### 5.2.2 The One-time Pad

One way to make a stream cipher of this type proof against attacks is for the key sequence to be as long as the plaintext, and to never repeat. This is known as the *one-time pad* and was proposed by Gilbert Vernam during World War I [1001]; given any ciphertext, and any plaintext of the same length, there’s a key that decrypts the ciphertext to the plaintext. So regardless of the amount of computation opponents can do, they’re none the wiser, as given any ciphertext, all possible plaintexts of that length are equally likely. This system therefore has *perfect secrecy*.

Here’s an example. Suppose you had intercepted a message from a wartime German agent which you knew started with ‘Heil Hitler’, and the first ten letters of ciphertext were DGTYI BWPJA. So the first ten letters of the one-time pad were wclnb tdefj, as shown in Figure 5.3:

<i>Plain</i>	heilhitler
<i>Key</i>	wclnb tdefj
<i>Cipher</i>	DGTYIBWPJA

Figure 5.3 – A spy’s message

But once he’s burnt the piece of silk with his key material, the spy can claim that he’s actually a member of the underground resistance, and the message actually said ‘Hang Hitler’. This is also possible, as the key material could just as easily have been wggssb tdefj, as shown in Figure 5.4:

<i>Cipher</i>	DGTYIBWPJA
<i>Key</i>	wggssb tdefj
<i>Plain</i>	hanghitler

Figure 5.4 – What the spy can claim he said

Now we rarely get anything for nothing in cryptology, and the price of the perfect secrecy of the one-time pad is that it fails completely to protect message integrity. So if you wanted to get this spy into trouble, you could change the ciphertext to DCYTI BWPJA (Figure 5.4):

<i>Cipher</i>	DCYTIBWPJA
<i>Key</i>	wclnb tdefj
<i>Plain</i>	hanghitler

Figure 5.5 – Manipulating the message to entrap the spy

## 5.2. HISTORICAL BACKGROUND

---

Leo Marks' engaging book on cryptography in the Special Operations Executive in World War II [1224] relates how one-time key material was printed on silk, which agents could conceal inside their clothing; whenever a key had been used it was torn off and burnt. In fact, during the war, Claude Shannon proved that a cipher has perfect secrecy if and only if there are as many possible keys as possible plaintexts, and every key is equally likely; so the one-time pad is the only kind of system that offers perfect secrecy. He was finally allowed to publish this in 1948 [1714, 1715].

The one-time tape was used for top-level communications by both sides from late in World War II, then for strategic communications between NATO allies, and for the US-USSR hotline from 1963. Thousands of machines were produced in total, using paper tapes for key material, until they were eventually replaced by computers from the mid-1980s<sup>1</sup>. But such cryptography is too expensive for most applications as it consumes as much key material as there is traffic. It's more common for stream ciphers to use a pseudorandom number generator to expand a short key into a long keystream. The data is then encrypted by combining the keystream, one symbol at a time, with the data. It's not enough for the keystream to appear "random" in the sense of passing the standard statistical randomness tests: it must also have the property that an opponent who gets his hands on even quite a lot of keystream symbols should not be able to predict any more of them.

An early example was *rotor machines*, mechanical stream-cipher devices that produce a very long sequence of pseudorandom states<sup>2</sup> and combine them with plaintext to get ciphertext. These machines were independently invented by a number of people from the 1920s, many of whom tried to sell them to the banking industry. Banks weren't in general interested, for reasons we'll discuss below, but rotor machines were very widely used by the combatants in World War II to encipher radio traffic, and the efforts made by the Allies to decipher German traffic included the work by Alan Turing and others on Colossus, which helped kickstart the computer industry after the war.

Stream ciphers have been widely used in hardware applications where the number of gates had to be minimised to save power. However, block ciphers are more flexible and are more common in systems being designed now, so let's look at them next.

### 5.2.3 An early block cipher – Playfair

The Playfair cipher was invented in 1854 by Sir Charles Wheatstone, a telegraph pioneer who also invented the concertina and the Wheatstone bridge. The reason it's not called the Wheatstone cipher is that he demonstrated it to Baron Playfair, a politician; Playfair in turn demonstrated it to Prince Albert and to Viscount Palmerston (later Prime Minister), on a napkin after dinner.

This cipher uses a 5 by 5 grid, in which we place the alphabet, permuted by

---

<sup>1</sup>Information about the machines can be seen at the Crypto Museum, <https://www.cryptomuseum.com>.

<sup>2</sup>letters in the case of the Hagelin machine used by the USA, permutations in the case of the German Enigma and the British Typex

## 5.2. HISTORICAL BACKGROUND

---

the key word, and omitting the letter ‘J’ (see Figure 5.6):

P	A	L	M	E
R	S	T	O	N
B	C	D	F	G
H	I	K	Q	U
V	W	X	Y	Z

Figure 5.6 – the Playfair enciphering table

The plaintext is first conditioned by replacing ‘J’ with ‘I’ wherever it occurs, then dividing it into letter pairs, preventing double letters occurring in a pair by separating them with an ‘x’, and finally adding a ‘z’ if necessary to complete the last letter pair. The example Playfair wrote on his napkin was ‘Lord Granville’s letter’ which becomes ‘lo rd gr an vi lx le sl et te rz’.

It is then enciphered two letters at a time using the following rules:

- if the two letters are in the same row or column, they are replaced by the succeeding letters. For example, ‘am’ enciphers to ‘LE’
- otherwise the two letters stand at two of the corners of a rectangle in the table, and we replace them with the letters at the other two corners of this rectangle. For example, ‘lo’ enciphers to ‘MT’.

We can now encipher our specimen text as follows:

Plain	lo rd gr an vi lx le sl et te rz
Cipher	MT TB BN ES WH TL MP TA LN NL NV

Figure 5.7 – example of Playfair enciphering

Variants of this cipher were used by the British army as a field cipher in World War I, and by the Americans and Germans in World War II. It’s a substantial improvement on Vigenère as the statistics that an analyst can collect are of *digraphs* (letter pairs) rather than single letters, so the distribution is much flatter and more ciphertext is needed for an attack.

Again, it’s not enough for the output of a block cipher to just look intuitively “random”. Playfair ciphertexts look random; but they have the property that if you change a single letter of a plaintext pair, then often only a single letter of the ciphertext will change. Thus using the key in Figure 5.7, **rd** enciphers to **TB** while **rf** enciphers to **OB** and **rg** enciphers to **NB**. One consequence is that given enough ciphertext, or a few probable words, the table (or an equivalent one) can be reconstructed [740]. In fact, the quote at the head of this chapter is a Playfair-encrypted message sent by the future President Jack Kennedy when he was a young lieutenant holed up on a small island with ten other survivors after his motor torpedo boat had been sunk in a collision with a Japanese destroyer. Had the Japanese intercepted it, they might possibly have decrypted it, and history could be different. For a stronger cipher, we will want the effects of

small changes in the cipher’s input to diffuse completely through its output. Changing one input bit should, on average, cause half of the output bits to change. We’ll tighten these ideas up in the next section.

The security of a block cipher can also be greatly improved by choosing a longer block length than two characters. For example, the *Data Encryption Standard* (DES), which is widely used in payment systems, has a block length of 64 bits and the *Advanced Encryption Standard* (AES), which has replaced it in most other applications, has a block length of twice this. I discuss the internal details of DES and AES below; for the time being, I’ll just remark that we need more than just an adequate block size.

For example, if a bank account number always appears at the same place in a transaction, then it’s likely to produce the same ciphertext every time a transaction involving it is encrypted with the same key. This might allow an opponent to cut and paste parts of two different ciphertexts in order to produce a valid but unauthorised transaction. Suppose a crook worked for a bank’s phone company, and monitored an enciphered transaction that he knew said “Pay IBM \$10,000,000”. He might wire \$1,000 to his brother causing the bank computer to insert another transaction saying “Pay John Smith \$1,000”, intercept this instruction, and make up a false instruction from the two ciphertexts that decrypted as “Pay John Smith \$10,000,000”. So unless the cipher block is as large as the message, the ciphertext will contain more than one block and we’ll need some way of binding the blocks together.

### 5.2.4 Hash functions

The third classical type of cipher is the *hash function*. This evolved to protect the integrity and authenticity of messages, where we don’t want someone to be able to manipulate the ciphertext in such a way as to cause a predictable change in the plaintext.

After the invention of the telegraph in the mid-19th century, banks rapidly became its main users and developed systems for transferring money electronically. What’s ‘wired’ is a payment instruction, such as:

*‘To Lombard Bank, London. Please pay from our account with you no. 1234567890 the sum of £1000 to John Smith of 456 Chesterton Road, who has an account with HSBC Bank Cambridge no. 301234 4567890123, and notify him that this was for “wedding present from Doreen Smith”. From First Cowboy Bank of Santa Barbara, CA, USA. Charges to be paid by us.’*

Since telegraph messages were relayed from one office to another by human operators, it was possible for an operator to manipulate a payment message.

In the nineteenth century, banks, telegraph companies and shipping companies developed *code books* that could not only protect transactions but also shorten them – which was important given the costs of international telegrams at the time. A code book was essentially a block cipher that mapped words or phrases to fixed-length groups of letters or numbers. So “Please pay from our

## 5.2. HISTORICAL BACKGROUND

---

account with you no.” might become ‘AFVCT’. Sometimes the codes were also enciphered.

The banks realised that neither stream ciphers nor code books protect message authenticity. If, for example, the codeword for ‘1000’ is ‘mauve’ and for ‘1,000,000’ is ‘magenta’, then the crooked telegraph clerk who can compare the coded traffic with known transactions should be able to figure this out and substitute one for the other.

The critical innovation, for the banks’ purposes, was to use a code book but to make the coding one-way by adding the code groups together into a number called a *test key*. (Modern cryptographers would describe it as a *hash value* or *message authentication code*, terms I’ll define more carefully later.)

Here is a simple example. Suppose the bank has a code book with a table of numbers corresponding to payment amounts as in Figure 5.8:

	0	1	2	3	4	5	6	7	8	9
x 1000	14	22	40	87	69	93	71	35	06	58
x 10,000	73	38	15	46	91	82	00	29	64	57
x 100,000	95	70	09	54	82	63	21	47	36	18
x 1,000,000	53	77	66	29	40	12	31	05	87	94

Figure 5.8 – a simple test key system

Now in order to authenticate a transaction for £376,514 we might add together 53 (no millions), 54 (300,000), 29 (70,000) and 71 (6,000) ignoring the less significant digits. This gives us a test key of 207.

Most real systems were more complex than this; they usually had tables for currency codes, dates and even recipient account numbers. In the better systems, the code groups were four digits long rather than two, and in order to make it harder for an attacker to reconstruct the tables, the test keys were compressed: a key of ‘7549’ might become ‘23’ by adding the first and second digits, and the third and fourth digits, ignoring the carry.

This made such test key systems into *one-way functions* in that although it was possible to compute a test from a message, given knowledge of the key, it was not possible to reverse the process and recover either a message or a key from a single test – the test just did not contain enough information. Indeed, one-way functions had been around since at least the seventeenth century. The scientist Robert Hooke published in 1678 the sorted anagram ‘ceiiinossstuu’ and revealed two years later that it was derived from ‘Ut tensio sic vis’ – ‘the force varies as the tension’, or what we now call Hooke’s law for a spring. (The goal was to establish priority for the idea while giving him time to do more work on it.)

Banking test keys are not strong by the standards of modern cryptography. Given between a few dozen and a few hundred tested messages, depending on the design details, a patient analyst could reconstruct enough of the tables to forge a transaction. With a few carefully chosen messages inserted into the banking system by an accomplice, it’s even easier. But the banks got away with it: test keys worked fine from the late nineteenth century through the

1980s. In several years working as a bank security consultant, and listening to elderly auditors' tales over lunch, I only ever heard of two cases of fraud that exploited it: one external attempt involving cryptanalysis, which failed because the attacker didn't understand bank procedures, and one successful but small fraud involving a crooked staff member. I'll discuss the systems that replaced test keys in the chapter on Banking and Bookkeeping.

However, test keys are our historical example of an algebraic function used for authentication. They have important modern descendants in the authentication codes used in the command and control of nuclear weapons, and also with modern block ciphers. The idea in each case is the same: if you can use a unique key to authenticate each message, simple algebra can give you ideal security. Suppose you have a message  $M$  of arbitrary length and want to compute an authentication code  $A$  of 128 bits long, and the property you want is that nobody should be able to find a different message  $M'$  whose authentication code under the same key will also be  $A$ , unless they know the key, except by a lucky guess for which the probability is  $2^{-128}$ . You can simply choose a 128-bit prime number  $p$  and compute  $A = k_1 M + k_2 \pmod{p}$  where the key consists of two 128-bit numbers  $k_1$  and  $k_2$ .

This is secure for the same reason the one-time pad is: given any other message  $M'$  you can find another key  $(k'_1, k'_2)$  that authenticates  $M'$  to  $A$ . So without knowledge of the key, the adversary who sees  $M$  and  $A$  simply has no information of any use in creating a valid forgery. As there are 256 bits of key and only 128 bits of tag, this holds even for an adversary with unlimited computing power: such an adversary can easily find the  $2^{128}$  possible keys for each pair of message and tag but has no way to choose between them. I'll discuss how this *universal hash function* is used with block ciphers below, and how it's used in nuclear command and control in Part 2.

### 5.2.5 Asymmetric primitives

Finally, some modern cryptosystems are asymmetric, in that different keys are used for encryption and decryption. So, for example, most web sites nowadays have a certificate containing a *public key* with which people can encrypt their session using a protocol called TLS; the owner of the web page can decrypt the traffic using the corresponding *private key*. We'll go into the details later.

There are some pre-computer examples of this too; perhaps the best is the postal service. You can send me a private message by addressing it to me and dropping it into a post box. Once that's done, I'm the only person who'll be able to read it. Of course, many things can go wrong: you might get the wrong address for me (whether by error or as a result of deception); the police might get a warrant to open my mail; the letter might be stolen by a dishonest postman; a fraudster might redirect my mail without my knowledge; or a thief might steal the letter from my doormat. Similar things can go wrong with public key cryptography: false public keys can be inserted into the system, computers can be hacked, people can be coerced and so on. We'll look at these problems in more detail in later chapters.

Another asymmetric application of cryptography is the *digital signature*. The

idea here is that I can sign a message using a private *signature key* and then anybody can check this using my public *signature verification key*. Again, there are pre-computer analogues in the form of manuscript signatures and seals; and again, there is a remarkably similar litany of things that can go wrong, both with the old way of doing things and with the new.

### 5.3 Security Models

Before delving into the detailed design of modern ciphers, I want to look more carefully at the various types of cipher and the ways in which we can reason about their security.

Security models seek to formalise the idea that a cipher is “good”. We’ve already seen the model of *perfect secrecy*: given any ciphertext, all possible plaintexts of that length are equally likely. Similarly, an authentication scheme that uses a key only once can be designed so that the best forgery attack on it is a random guess, whose probability of success can be made as low as we want by choosing a long enough tag.

The second model is *concrete security*, where we want to know how much actual work an adversary has to do. At the time of writing, it takes the most powerful adversary in existence – the community of bitcoin miners, burning about as much electricity as the state of Denmark – about ten minutes to solve a 68-bit cryptographic puzzle and mine a new block. So an 80-bit key would take them  $2^{12}$  times as long, or about a month; a 128-bit key, the default in modern systems, is  $2^{48}$  times harder again. So even in 1000 years the probability of finding the right key by chance is  $2^{-35}$  or one in many billion. In general, a system is  $(t, \epsilon)$ -secure if an adversary working for time  $t$  succeeds in breaking the cipher with probability at most  $\epsilon$ .

The third model, which many theoreticians now call the standard model, is about *indistinguishability*. This enables us to reason about the specific properties of a cipher we care about. For example, most cipher systems don’t hide the length of a message, so we can’t define a cipher to be secure by just requiring that an adversary not be able to distinguish ciphertexts corresponding to two messages; we have to be more explicit and require that the adversary not be able to distinguish between two messages  $M_1$  and  $M_2$  of the same length. This is formalised by having the cryptographer and the cryptanalyst play a game in which the analyst wins by finding an efficient discriminator of something she shouldn’t be able to discriminate with more than negligible probability. If the cipher doesn’t have perfect security this can be *asymptotic*, where we typically want the effort to grow faster than any polynomial function of a security parameter  $n$  – say the length of the key in bits. A security proof typically consists of a *reduction* where we show that if there exists a randomised (i.e. probabilistic) algorithm running in time polynomial in  $n$  that learns information it shouldn’t with non-negligible probability, then this would give an efficient discriminator for an underlying cryptographic primitive that we already trust. Finally, a construction is said to have *semantic security* if there’s no efficient distinguisher for the plaintext regardless of any side information the analyst may have about it; even if she knows all but one bit of it, and even if she can get a decryption of any

other ciphertext, she can't learn anything more from the target ciphertext. This skips over quite a few mathematical details, which you can find in a standard text such as Katz and Lindell [1022].

The fourth model is the random oracle model, which is not as general as the standard model but which often leads to more efficient constructions. We call a cryptographic primitive *pseudorandom* if there's no efficient way of distinguishing it from a random function of that type, and in particular it passes all the statistical and other randomness tests we apply. Of course, the cryptographic primitive will actually be an algorithm, implemented as an array of gates in hardware or a program in software; but the outputs should "look random" in that they're indistinguishable from a suitable random oracle given the type and the number of tests that our model of computation permits.

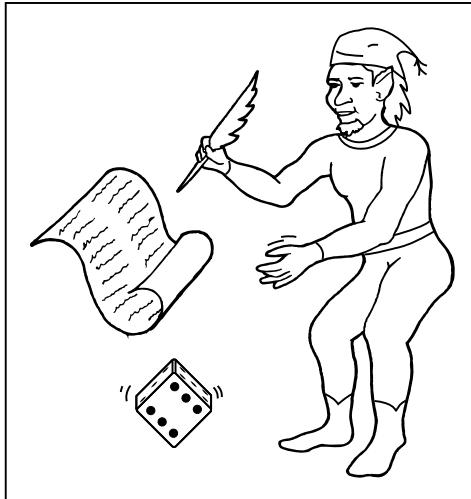


Figure 5.9: – the random oracle

To visualise a random oracle, we might imagine an elf sitting in a black box with a source of physical randomness and some means of storage (see Figure 5.9) – represented in our picture by the dice and the scroll. The elf will accept inputs of a certain type, then look in the scroll to see whether this query has ever been answered before. If so, it will give the answer it finds there; if not, it will generate an answer at random by throwing the dice, and keep a record for future reference. We'll further assume finite bandwidth – the elf will only answer so many queries every second. What's more, our oracle can operate according to several different rules.

### 5.3.1 Random functions – hash functions

The first type of random oracle is the random function. A random function accepts an input string of any length and outputs a string of fixed length, say  $n$  bits long. The same input gives the same output, but the set of outputs appears random. So the elf just has a simple list of inputs and outputs, which grows steadily as it works.

Random functions are our model for *cryptographic hash functions*. These were first used in computer systems for one-way encryption of passwords in the 1960s and have many more uses today. For example, if the police seize your laptop, the standard forensic tools will compute checksums on all the files, to identify which files are already known (such as system files) and which are novel (such as user data). These hash values will change if a file is corrupted and so can assure the court that the police haven't tampered with evidence. And if we want evidence that we possessed a given electronic document by a certain date, we might submit it to an online time-stamping service or have it mined into the Bitcoin blockchain. However, if the document is still secret – for example an invention for which we want to establish a priority date – then we would not upload the whole document, but just the message hash. This is the modern equivalent of Hooke's anagram that we discussed in section 5.2.4 above.

#### 5.3.1.1 Properties

The first main property of a random function is one-wayness. Given knowledge of an input  $x$  we can easily compute the hash value  $h(x)$ , but it is very difficult given  $h(x)$  to find  $x$  if such an input is not already known. (The elf will only pick outputs for given inputs, not the other way round.) As the output is random, the best an attacker can do to invert a random function is to keep on feeding in more inputs until he gets lucky; with an  $n$ -bit output this will take about  $2^{n-1}$  guesses on average. A pseudorandom function will have the same properties, or they could be used to distinguish it from a random function, contrary to our definition. So a pseudorandom function will also be a *one-way function*, provided there are too many possible outputs for the opponent to guess an input that has a desired target output by chance. This means choosing  $n$  so that the opponent can't do anything near  $2^n$  computations. If we claim, for example, that SHA256 is a pseudorandom function, then we're saying that there's no practical way to find an input that hashes to a given 256-bit value, unless you knew it already and used it to compute that value.

A second property of pseudorandom functions is that the output will not give any information at all about even part of the input. So we can get a one-way encryption of the value  $x$  by concatenating it with a secret key  $k$  and computing  $h(x, k)$ . If the hash function isn't random enough, though, using it for one-way encryption in this manner is asking for trouble. (I'll discuss an example later in section 22.2.1: the hash function used by many phone companies in the 1990s and early 2000s to authenticate mobile phone users wasn't random enough, which led to attacks.)

A third property of pseudorandom functions with sufficiently long outputs is that it is hard to find *collisions*, that is, different messages  $M_1 \neq M_2$  with  $h(M_1) = h(M_2)$ . Unless the opponent can find a shortcut attack (which would mean the function wasn't pseudorandom) then the best way of finding a collision is to collect a large set of messages  $M_i$  and their corresponding hashes  $h(M_i)$ , sort the hashes, and look for a match. If the hash function output is an  $n$ -bit number, so that there are  $2^n$  possible hash values, then the number of hashes the enemy will need to compute before he can expect to find a match will be about the square root of this, namely  $2^{n/2}$  hashes. This fact is of huge importance in

security engineering, so let's look at it more closely.

#### 5.3.1.2 The birthday theorem

The birthday theorem gets its name from the following problem. A maths teacher asks a class of 30 pupils what they think is the probability that two of them have the same birthday. Most pupils intuitively think it's unlikely, and the maths teacher then asks the pupils to state their birthdays one after another. The odds of a match exceed 50% once 23 pupils have been called. As this surprises most people, it's also known as the 'birthday paradox'.

The birthday theorem was first used in the 1930's to count fish, so it's also known as *capture-recapture statistics* [1665]. Suppose there are  $N$  fish in a lake and you catch  $m$  of them, ring them and throw them back, then when you first catch a fish you've ringed already,  $m$  should be 'about' the square root of  $N$ . The intuitive reason why this holds is that once you have  $\sqrt{N}$  samples, each could potentially match any of the others, so the number of possible matches is about  $\sqrt{N} \times \sqrt{N}$  or  $N$ , which is what you need<sup>3</sup>.

This theorem has many applications for the security engineer. For example, if we have a biometric system that can authenticate a person's claim to identity with a probability of only one in a million that two randomly selected subjects will be falsely identified as the same person, this doesn't mean that we can use it as a reliable means of identification in a university with a user population of twenty thousand staff and students. This is because there will be almost two hundred million possible pairs. In fact, you expect to find the first *collision* – the first pair of people who can be mistaken for each other by the system – once you have somewhat over a thousand people enrolled. It may well, however, be OK to use it to verify a claimed identity (though many other things can go wrong; see the chapter on Biometrics in Part 2 for a discussion).

There are some applications where collision-search attacks aren't a problem, such as in challenge-response protocols where an attacker has to find the answer to the challenge just issued, and where you can prevent challenges repeating. In identify-friend-or-foe (IFF) systems, for example, common equipment has a response length of 48 to 80 bits. You can't afford much more than that, as it costs radar accuracy.

But there are other applications in which collisions are unacceptable. When we design digital signature systems, we typically pass the message  $M$  through a cryptographic hash function first, and then sign the hash  $h(M)$ , for a number of reasons we'll discuss later. In such an application, if it were possible to find collisions with  $h(M_1) = h(M_2)$  but  $M_1 \neq M_2$ , then a Mafia owned bookstore's web site might precalculate suitable pairs  $M_1, M_2$ , get you to sign an  $M_1$  saying something like "I hereby order a copy of Rubber Fetish volume 7 for \$32.95" and then present the signature together with an  $M_2$  saying something like "I hereby mortgage my house for \$75,000 and please send the funds to Mafia Holdings Inc., Bermuda."

For this reason, hash functions used with digital signature schemes have  $n$

---

<sup>3</sup>More precisely, the probability that  $m$  fish chosen randomly from  $N$  fish are different is  $\beta = N(N-1)\dots(N-m+1)/N^m$  which is asymptotically solved by  $N \simeq m^2/2\log(1/\beta)$  [1037].

large enough to make them collision-free. Historically, the two most common hash functions have been MD5, which has a 128-bit output and will thus require at most  $2^{64}$  computations to break, and SHA1 with a 160-bit output and a work factor for the cryptanalyst of at most  $2^{80}$ . However, collision search gives at best an upper bound on the strength of a hash function, and both these particular functions have turned out to be disappointing, with cryptanalytic attacks that I'll describe later in section 5.6.2.

To sum up: if you need a cryptographic hash function to be collision resistant, then you'd better choose a function with an output of at least 256 bits, such as SHA-2 or SHA-3. However if you only need to be sure that nobody will find a second preimage for an existing, externally given hash, then you can perhaps make do with less.

### 5.3.2 Random generators – stream ciphers

The second basic cryptographic primitive is the *random generator*, also known as a *keystream generator* or *stream cipher*. This is also a random function, but it's the reverse of the hash function in that it has a short input and a long output. If we had a good pseudorandom function whose input and output were long enough, we could turn it into a hash function by throwing away all but a few hundred bits of the output, and turn it into a stream cipher by padding all but a few hundred bits of the input with a constant and using the output as a keystream.

It can be used to protect the confidentiality of our backup data as follows: we go to the keystream generator, enter a key, get a long file of random bits, and exclusive-or it with our plaintext data to get ciphertext, which we then send to our backup service in the cloud. (This is also called an *additive stream cipher* as exclusive-or is addition modulo 2.) We can think of the elf generating a random tape of the required length each time he is presented with a new key, giving it to us and keeping a copy on his scroll for reference in case he's given the same input key again. If we need to recover the data, we go back to the generator, enter the same key, get the same keystream, and exclusive-or it with our ciphertext to get our plaintext back again. Other people with access to the keystream generator won't be able to generate the same keystream unless they know the key. Note that this would not give us any guarantee of file integrity; as we saw in the discussion of the one-time pad, adding a keystream to plaintext can protect confidentiality, but it can't detect modification of the file. For that, we might make a hash of the file and keep that somewhere safe. It may be easier to protect the hash from modification than the whole file.

One-time pad systems are a close fit for our theoretical model, except in that they are used to secure communications across space rather than time: the two communicating parties have shared a copy of a keystream in advance. Vernam's original telegraph cipher machine used punched paper tape; Marks describes how SOE agents' silken keys were manufactured in Oxford by retired ladies shuffling counters; we'll discuss modern hardware random number generators in the chapter on Physical Security.

A real problem with keystream generators is to prevent the same keystream

being used more than once, whether to encrypt more than one backup tape or to encrypt more than one message sent on a communications channel. During World War II, the amount of Russian diplomatic traffic exceeded the quantity of one-time tape they had distributed in advance to their embassies, so it was reused. But if  $M_1 + K = C_1$  and  $M_2 + K = C_2$ , then the opponent can combine the two ciphertexts to get a combination of two messages:  $C_1 - C_2 = M_1 - M_2$ , and if the messages  $M_i$  have enough redundancy then they can be recovered. Text messages do in fact contain enough redundancy for much to be recovered; in the case of the Russian traffic this led to the Venona project in which the US and UK decrypted large amounts of wartime Russian traffic from 1943 onwards and broke up a number of Russian spy rings. In the words of one former NSA chief scientist, it became a “two-time tape”.

To avoid this, the normal engineering practice is to have not just a key but also a *seed* (also known as an *initialisation vector* or IV) so we start the keystream at a different place each time. The seed  $N$  may be a sequence number, or generated from a protocol in a more complex way. Here, you need to ensure that both parties synchronise on the right working key even in the presence of an adversary who may try to get you to reuse old keystream.

#### 5.3.3 Random permutations – block ciphers

The third type of primitive, and the most important in modern cryptography, is the block cipher, which we model as a *random permutation*. Here, the function is invertible, and the input plaintext and the output ciphertext are of a fixed size. With Playfair, both input and output are two characters; with DES, they’re both bit strings of 64 bits. Whatever the number of symbols and the underlying alphabet, encryption acts on a block of fixed length. (So if you want to encrypt a shorter input, you have to pad it as with the final ‘z’ in our Playfair example.)

We can visualise block encryption as follows. As before, we have an elf in a box with dice and a scroll. This has on the left a column of plaintexts and on the right a column of ciphertexts. When we ask the elf to encrypt a message, it checks in the left hand column to see if it has a record of it. If not, it rolls the dice to generate a random ciphertext of the appropriate size (and which doesn’t appear yet in the right hand column of the scroll), and then writes down the plaintext/ciphertext pair in the scroll. If it does find a record, it gives us the corresponding ciphertext from the right hand column.

When asked to decrypt, the elf does the same, but with the function of the columns reversed: he takes the input ciphertext, looks for it on the right hand scroll, and if he finds it he gives the message with which it was previously associated. If not, he generates a new message at random, notes it down and gives it to us.

A *block cipher* is a keyed family of pseudorandom permutations. For each key, we have a single permutation that’s independent of all the others. We can think of each key as corresponding to a different scroll. The intuitive idea is that a cipher machine should output the ciphertext given the plaintext and the key, and output the plaintext given the ciphertext and the key, but given only the plaintext and the ciphertext it should output nothing. Furthermore, nobody

### **5.3. SECURITY MODELS**

---

should be able to infer any information about plaintexts or ciphertexts that it has not yet produced.

We will write a block cipher using the notation established for encryption in the chapter on protocols:

$$C = \{M\}_K$$

The random permutation model also allows us to define different types of attack on block ciphers. In a *known plaintext attack*, the opponent is just given a number of randomly chosen inputs and outputs from the oracle corresponding to a target key. In a *chosen plaintext attack*, the opponent is allowed to put a certain number of plaintext queries and get the corresponding ciphertexts. In a *chosen ciphertext attack* he gets to make a number of ciphertext queries. In a *chosen plaintext/ciphertext attack* he is allowed to make queries of either type. Finally, in a *related key attack* he can make queries that will be answered using keys related to the target key  $K$ , such as  $K + 1$  and  $K + 2$ .

In each case, the objective of the attacker may be either to deduce the answer to a query he hasn't already made (a *forgery attack*), or to recover the key (unsurprisingly known as a *key recovery attack*).

This precision about attacks is important. When someone discovers a vulnerability in a cryptographic primitive, it may or may not be relevant to your application. Often it won't be, but will have been hyped by the media – so you will need to be able to explain clearly to your boss and your customers why it's not a problem. So you have to look carefully to find out exactly what kind of attack has been found, and what the parameters are. For example, the first major attack announced on the Data Encryption Standard algorithm (differential cryptanalysis) required  $2^{47}$  chosen plaintexts to recover the key, while the next major attack (linear cryptanalysis) improved this to  $2^{43}$  known plaintexts. While these attacks were of huge scientific importance, their practical engineering effect was zero, as no practical systems make that much known text (let alone chosen text) available to an attacker. Such impractical attacks are often referred to as *certification* as they affect the cipher's security certification rather than providing a practical exploit. They can have a commercial effect, though: the attacks on DES undermined confidence and started moving people to other ciphers. In some other cases, an attack that started off as certification has been developed by later ideas into an exploit.

Which sort of attacks you should be worried about depends on your application. With a broadcast entertainment system, for example, a hacker can buy a decoder, watch a lot of movies and compare them with the enciphered broadcast signal; so a *known-plaintext attack* might be the main threat. But there are surprisingly many applications where *chosen-plaintext attacks* are possible. A historic example is from World War II, where US analysts learned of Japanese intentions for an island 'AF' which they suspected meant Midway. So they arranged for Midway's commander to send an unencrypted message reporting problems with its fresh water condenser, and then intercepted a Japanese report that 'AF is short of water'. Knowing that Midway was the Japanese objective, Admiral Chester Nimitz was waiting for them and sank four Japanese carriers, turning the tide of the war [1001].

The other attacks are more specialised. *Chosen plaintext/ciphertext* attacks may be a worry where the threat is a *lunchtime attack*: someone who gets temporary access to a cryptographic device while its authorised user is out, and tries out the full range of permitted operations for a while with data of their choice. *Related-key attacks* are a concern where the block cipher is used as a building block in the construction of a hash function (which we'll discuss below). To exclude all such attacks, the goal is semantic security, as discussed above; the cipher should not allow the inference of unauthorised information (whether of plaintexts, ciphertexts or keys) other than with negligible probability.

#### 5.3.4 Public key encryption and trapdoor one-way permutations

A *public-key encryption* algorithm is a special kind of block cipher in which the elf will perform the encryption corresponding to a particular key for anyone who requests it, but will do the decryption operation only for the key's owner. To continue with our analogy, the user might give a secret name to the scroll that only she and the elf know, use the elf's public one-way function to compute a hash of this secret name, publish the hash, and instruct the elf to perform the encryption operation for anybody who quotes this hash. This means that a principal, say Alice, can publish a key and if Bob wants to, he can now encrypt a message and send it to her, even if they have never met. All that is necessary is that they have access to the oracle.

The simplest variation is the *trapdoor one-way permutation*. This is a computation that anyone can perform, but which can be reversed only by someone who knows a *trapdoor* such as a secret key. This model is like the ‘one-way function’ model of a cryptographic hash function. Let us state it formally nonetheless: a public key encryption primitive consists of a function which given a random input  $R$  will return two keys,  $KR$  (the public encryption key) and  $KR^{-1}$  (the private decryption key) with the properties that

1. Given  $KR$ , it is infeasible to compute  $KR^{-1}$  (so it's not possible to compute  $R$  either);
2. There is an encryption function  $\{\dots\}$  which, applied to a message  $M$  using the encryption key  $KR$ , will produce a ciphertext  $C = \{M\}_{KR}$ ; and
3. There is a decryption function which, applied to a ciphertext  $C$  using the decryption key  $KR^{-1}$ , will produce the original message  $M = \{C\}_{KR^{-1}}$ .

For practical purposes, we will want the oracle to be replicated at both ends of the communications channel, and this means either using tamper-resistant hardware or (more commonly) implementing its functions using mathematics rather than metal.

In most real systems, the encryption is randomised, so that every time someone uses the same public key to encrypt the same message, the answer is different; this is necessary for semantic security, so that an opponent cannot check whether a guess of the plaintext of a given ciphertext is correct. There are

even more demanding models than this, for example to analyse security in the case where the opponent can get ciphertexts of their choice decrypted, with the exception of the target ciphertext. But this will do for now.

### **5.3.5 Digital signatures**

The final cryptographic primitive we'll define here is the *digital signature*. The basic idea is that a signature on a message can be created by only one principal, but checked by anyone. It can thus perform the same function in the electronic world that ordinary signatures do in the world of paper. Applications include signing software updates, so that a PC can tell that an update to Windows was really produced by Microsoft rather than by a foreign intelligence agency.

Signature schemes, too, can be deterministic or randomised: in the first, computing a signature on a message will always give the same result and in the second, it will give a different result. (The latter is more like handwritten signatures; no two are ever alike but the bank has a means of deciding whether a given specimen is genuine or forged.) Also, signature schemes may or may not support *message recovery*. If they do, then given the signature, anyone can recover the message on which it was generated; if they don't, then the verifier needs to know or guess the message before they can perform the verification.

Formally, a signature scheme, like a public key encryption scheme, has a keypair generation function which given a random input  $R$  will return two keys,  $\sigma R$  (the private signing key) and  $VR$  (the public signature verification key) with the properties that

1. Given the public signature verification key  $VR$ , it is infeasible to compute the private signing key  $\sigma R$ ;
2. There is a digital signature function which given a message  $M$  and a private signature key  $\sigma R$ , will produce a signature  $Sig_{\sigma R}\{M\}$ ; and
3. There is a verification function which, given a signature  $Sig_{\sigma R}\{M\}$  and the public signature verification key  $VR$ , will output TRUE if the signature was computed correctly with  $\sigma R$  and otherwise output FALSE.

Where we don't need message recovery, we can model a simple digital signature algorithm as a random function that reduces any input message to a one-way hash value of fixed length, followed by a special kind of block cipher in which the elf will perform the operation in one direction, known as *signature*, for only one principal. In the other direction, it will perform verification for anybody.

For this simple scheme, signature verification means that the elf (or the signature verification algorithm) only outputs TRUE or FALSE depending on whether the signature is good. But in a scheme with *message recovery*, anyone can input a signature and get back the message corresponding to it. In our elf model, this means that if the elf has seen the signature before, it will give the message corresponding to it on the scroll, otherwise it will give a random value (and record the input and the random output as a signature and message

pair). This is sometimes desirable: when sending short messages over a low bandwidth channel, it can save space if only the signature has to be sent rather than the signature plus the message. An application that uses message recovery is machine-printed postage stamps, or *indicia*: the stamp consists of a 2-d barcode with a digital signature made by the postal meter and which contains information such as the value, the date and the sender's and recipient's post codes. We discuss this at the end of section 16.3.2.

In the general case we do not need message recovery; the message to be signed may be of arbitrary length, so we first pass it through a hash function and then sign the hash value. We need the hash function to be not just one-way, but also collision resistant.

## 5.4 Symmetric crypto algorithms

Now that we've tidied up the definitions, we'll look under the hood to see how they can be implemented in practice. While most explanations are geared towards graduate mathematics students, the presentation I'll give here is based on one I developed over the years with computer science undergraduates, to help the non-specialist grasp the essentials. In fact, even at the research level, most of cryptography is as much computer science as mathematics: modern attacks on ciphers are put together from guessing bits, searching for patterns, sorting possible results and so on, and require ingenuity and persistence rather than anything particularly highbrow.

### 5.4.1 SP-networks

Claude Shannon suggested in the 1940s that strong ciphers could be built by combining substitution with transposition repeatedly. For example, one might add some key material to a block of input text, and then shuffle subsets of the input, and continue in this way a number of times. He described the properties of a cipher as being *confusion* and *diffusion* – adding unknown key values will confuse an attacker about the value of a plaintext symbol, while diffusion means spreading the plaintext information through the ciphertext. Block ciphers need diffusion as well as confusion.

The earliest block ciphers were simple networks which combined substitution and permutation circuits, and so were called SP-networks [1009]. Figure 5.10 shows an SP-network with sixteen inputs, which we can imagine as the bits of a sixteen-bit number, and two layers of four-bit invertible substitution boxes (or *S-boxes*), each of which can be visualised as a lookup table containing some permutation of the numbers 0 to 15.

The point of this arrangement is that if we were to implement an arbitrary 16 bit to 16 bit function in digital logic, we would need  $2^{20}$  bits of memory – one lookup table of  $2^{16}$  bits for each single output bit. That's hundreds of

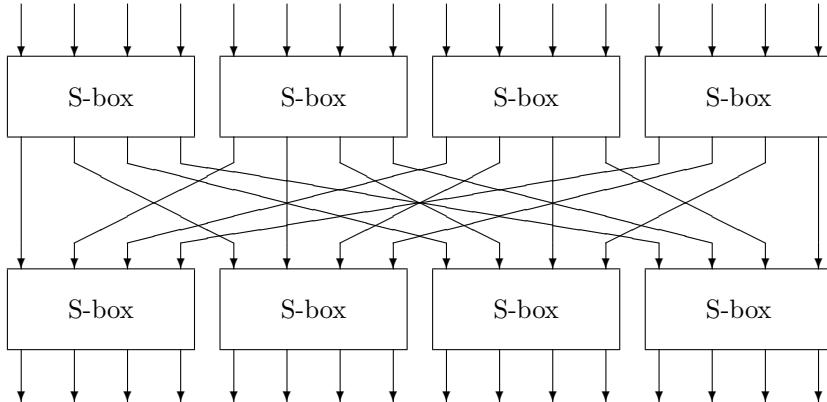


Figure 5.10: – a simple 16-bit SP-network block cipher

thousands of gates, while a four bit to four bit function takes only  $4 \times 2^4$  or 64 bits of memory. One might hope that with suitable choices of parameters, the function produced by iterating this simple structure would be indistinguishable from a random 16 bit to 16 bit function to an opponent who didn't know the value of the key. The key might consist of some choice of a number of four-bit S-boxes, or it might be added at each round to provide confusion and the resulting text fed through the S-boxes to provide diffusion.

Three things need to be done to make such a design secure:

1. the cipher needs to be “wide” enough
2. it needs to have enough rounds, and
3. the S-boxes need to be suitably chosen.

#### 5.4.1.1 Block size

First, a block cipher which operated on sixteen bit blocks would be rather limited, as an opponent could just build a dictionary of plaintext and ciphertext blocks as they were observed. The birthday theorem tells us that even if the input plaintexts were random, he'd expect to find a match as soon as he had seen a few hundred blocks. So a practical block cipher will usually deal with plaintexts and ciphertexts of 64 bits, 128 bits or even more. So if we are using four-bit to four-bit S-boxes, we may have 16 of them (for a 64 bit block size) or 32 of them (for a 128 bit block size).

#### 5.4.1.2 Number of rounds

Second, we have to have enough rounds. The two rounds in Figure 5.10 are completely inadequate, as an opponent can deduce the values of the S-boxes by tweaking input bits in suitable patterns. For example, he could hold the rightmost 12 bits constant and try tweaking the leftmost four bits, to deduce the values in the top left S-box. (The attack is slightly more complicated than

this, as sometimes a tweak in an input bit to an S-box won't produce a change in any output bit, so we have to change one of its other inputs and tweak again. But it is still a basic student exercise.)

The number of rounds we need depends on the speed with which data diffuse through the cipher. In our simple example, diffusion is very slow because each output bit from one round of S-boxes is connected to only one input bit in the next round. Instead of having a simple permutation of the wires, it is more efficient to have a linear transformation in which each input bit in one round is the exclusive-or of several output bits in the previous round. If the block cipher is to be used for decryption as well as encryption, this linear transformation will have to be invertible. We'll see some concrete examples below in the sections on AES and DES.

#### 5.4.1.3 Choice of S-boxes

The design of the S-boxes also affects the number of rounds required for security, and studying bad choices gives us our entry into the deeper theory of block ciphers. Suppose that the S-box were the permutation that maps the inputs  $(0,1,2,\dots,15)$  to the outputs  $(5,7,0,2,4,3,1,6,8,10,15,12,9,11,14,13)$ . Then the most significant bit of the input would come through unchanged as the most significant bit of the output. If the same S-box were used in both rounds in the above cipher, then the most significant bit of the input would pass through to become the most significant bit of the output. We certainly couldn't claim that our cipher was pseudorandom.

#### 5.4.1.4 Linear Cryptanalysis

Attacks on real block ciphers are usually harder to spot than in this example, but they use the same ideas. It might turn out that the S-box had the property that bit one of the input was equal to bit two plus bit four of the output; more commonly, there will be linear approximations to an S-box which hold with a certain probability. *Linear cryptanalysis* [895, 1244] proceeds by collecting a number of relations such as "bit 2 plus bit 5 of the input to the first S-box is equal to bit 1 plus bit 8 of the output, with probability  $13/16$ ", then searching for ways to glue them together into an algebraic relation between input bits, output bits and key bits that holds with a probability different from one half. If we can find a linear relationship that holds over the whole cipher with probability  $p = 0.5 + 1/M$ , then according to the sampling theorem in probability theory we can expect to start recovering keybits once we have about  $M^2$  known texts. If the value of  $M^2$  for the best linear relationship is greater than the total possible number of known texts (namely  $2^n$  where the inputs and outputs are  $n$  bits wide), then we consider the cipher to be secure against linear cryptanalysis.

#### 5.4.1.5 Differential Cryptanalysis

*Differential Cryptanalysis* [245, 895] is similar but is based on the probability that a given change in the input to an S-box will give rise to a certain change in the output. A typical observation on an 8-bit S-box might be that "if we

flip input bits 2, 3, and 7 at once, then with probability 11/16 the only output bits that will flip are 0 and 1”. In fact, with any nonlinear Boolean function, tweaking some combination of input bits will cause some combination of output bits to change with a probability different from one half. The analysis procedure is to look at all possible input difference patterns and look for those values  $\delta_i$ ,  $\delta_o$  such that an input change of  $\delta_i$  will produce an output change of  $\delta_o$  with particularly high (or low) probability.

As in linear cryptanalysis, we then search for ways to join things up so that an input difference which we can feed into the cipher will produce a known output difference with a useful probability over a number of rounds. Given enough chosen inputs, we will see the expected output and be able to make deductions about the key. As in linear cryptanalysis, it’s common to consider the cipher to be secure if the number of texts required for an attack is greater than the total possible number of different texts for that key. (We have to be careful of pathological cases, such as if you had a cipher with a 32-bit block and a 128-bit key with a differential attack whose success probability given a single pair was  $2^{-40}$ . Given a lot of text under a number of keys, we’d eventually solve for the current key.)

There are many variations on these two themes. For example, instead of looking for high probability differences, we can look for differences that can’t happen (or that happen only rarely). This has the charming name of *impossible cryptanalysis*, but it is quite definitely possible against many systems [242]<sup>4</sup>.

Block cipher design involves a number of trade-offs. For example, we can reduce the per-round information leakage, and thus the required number of rounds, by designing the rounds carefully. But a complex design might be slow in software, or need a lot of gates in hardware, so using simple rounds but more of them might have been better. Simple rounds may also be easier to analyse. A prudent designer will also use more rounds than are strictly necessary to block the attacks known today, in order to give some safety margin, as attacks only ever get better. But while we may be able to show that a cipher resists all the attacks we know of, and with some safety margin, this says little about whether it will resist novel types of attack. (A general security proof for a block cipher would appear to imply a result such as  $P \neq NP$  that would revolutionise computer science.)

#### 5.4.2 The Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) is an algorithm originally known as Rijndael after its inventors Vincent Rijmen and Joan Daemen [507]. It acts on 128-bit blocks and can use a key of 128, 192 or 256 bits in length. It is an SP-network; in order to specify it, we need to fix the S-boxes, the linear transformation between the rounds, and the way in which the key is added into the computation.

AES uses a single S-box that acts on a byte input to give a byte output. For implementation purposes it can be regarded simply as a lookup table of

---

<sup>4</sup>This may have been used first at Bletchley in World War II where a key insight into breaking the German Enigma machine was that no letter ever enciphered to itself.

256 bytes; it is actually defined by the equation  $S(x) = M(1/x) + b$  over the field  $GF(2^8)$  where  $M$  is a suitably chosen matrix and  $b$  is a constant. This construction gives tight differential and linear bounds.

The linear transformation is based on arranging the 16 bytes of the value being enciphered in a square and then doing bytewise shuffling and mixing operations. The first step is the *shuffle*, in which the top row of four bytes is left unchanged while the second row is shifted one place to the left, the third row by two places and the fourth row by three places. The second step is a column-mixing step in which the four bytes in a column are mixed using matrix multiplication. This is illustrated in Figure 5.11, which shows, as an example, how a change in the value of the third byte in the first column is propagated. The effect of this combination is that a change in the input to the cipher can potentially affect all of the output after just two rounds – an *avalanche* effect that makes both linear and differential attacks harder.

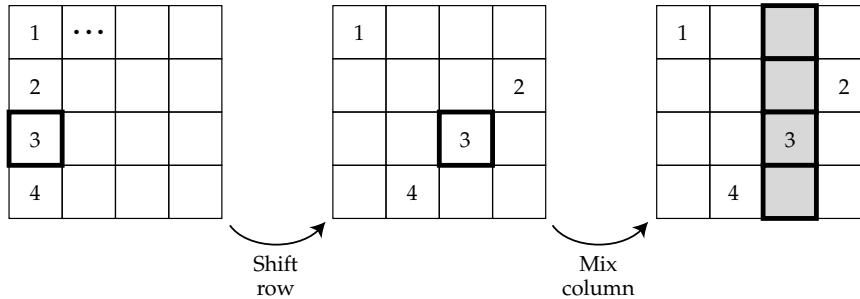


Figure 5.11: – the AES linear transformation, illustrated by its effect on byte 3 of the input

The key material is added byte by byte after the linear transformation. This means that 16 bytes of key material are needed per round; they are derived from the user supplied key material by means of a recurrence relation.

The algorithm uses 10 rounds with 128-bit keys, 12 rounds with 192-bit keys and 14 rounds with 256-bit keys. These are enough to give practical, but not certificational, security – as indeed we expected at the time of the AES competition, and as I described in earlier editions of this chapter. The first key-recovery attacks use a technique called biclique cryptanalysis and were discovered in 2009 by Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger [273]; they give only a very small advantage, with complexity now estimated at  $2^{126}$  for 128-bit AES and  $2^{254.3}$  for 256-bit AES, as opposed to  $2^{127}$  and  $2^{255}$  for brute-force search. Faster shortcut attacks are known for the case where we have related keys. But none of these attacks make any difference in practice, as they require infeasibly large numbers of texts or very special combinations of related keys.

Should we trust AES? The governments of Russia, China and Japan try to get firms to use local ciphers instead, and the Japanese offering, Camellia, is found in a number of crypto libraries alongside AES and another AES competition finalist, Bruce Schneier's Twofish. (Camellia was designed by a team whose own AES candidate was knocked out at the first round.) Conspiracy theorists

note that the US government picked the weakest of the five algorithms that were finalists in the AES competition. Well, I was one of the designers of the AES finalist Serpent [94], which came second in the competition: the winner Rijndael got 86 votes, Serpent 59 votes, Twofish 31 votes, RC6 23 votes and MARS 13 votes. Serpent has a simple structure that makes it easy to analyse – the structure of Figure 5.10, but modified to be wide enough and to have enough rounds – and was designed to have a much larger security margin than Rijndael in anticipation of the attacks that have now appeared. Yet the simple fact is that while Serpent is more secure, Rijndael is faster; industry and crypto researchers voted for it at the last AES conference, and NIST approved it as the standard.

Having been involved in the whole process, and having worked on the analysis and design of shared-key ciphers for much of the 1990s, I have a high level of confidence that AES is secure against practical attacks based on mathematical cryptanalysis. And even though AES is less secure than Serpent, practical security is all about implementation, and we now have enormous experience at implementing AES. Practical attacks include timing analysis and power analysis. In the former, the main risk is that an opponent observes cache misses and uses them to work out the key. In the latter, an opponent uses measurements of the current drawn by the device doing the crypto – think of a bank smartcard that a customer places in a terminal in a Mafia-owned shop. I discuss both in detail in Part 2, in the chapter on Emission Security; countermeasures include special operations in many CPUs to do AES, which are available precisely because the algorithm is now a standard. It does not make sense to implement Serpent as well, ‘just in case AES is broken’: having swappable algorithms is known as *pluggable cryptography*, yet the risk of a fatal error in the algorithm negotiation protocol is orders of magnitude greater than the risk that anyone will come up with a production attack on AES. (We’ll see a number of examples later where using multiple algorithms caused something to break horribly.)

The back story is that, back in the 1970s, the NSA manipulated the choice and parameters of the previous standard block cipher, the *Data Encryption Standard* (DES) in such a way as to deliver a cipher that was good enough for US industry at the time, while causing foreign governments to believe it was insecure, so they used their own weak designs instead. I’ll discuss this in more detail below, once I’ve described the design of DES. AES seems to have followed this playbook; by selecting an algorithm that was only just strong enough mathematically and whose safe implementation requires skill and care, the US government saw to it that firms in Russia, China, Japan and elsewhere will end up using systems that are less secure because less skill and effort has been invested in the implementation. However, this was probably luck rather than Machiavellian cunning: the relevant committee at NIST would have had to have a lot of courage to disregard the vote and choose another algorithm instead. Oh, and the NSA has since 2005 approved AES with 128-bit keys for protecting information up to SECRET and with 192-bit or 256-bit keys for TOP SECRET. So I recommend that you use AES instead of GOST, or Camellia, or even Serpent. The definitive specification of AES is Federal Information Processing Standard 197, and its inventors have written a book describing its design in detail [507].

### 5.4.3 Feistel ciphers

Many block ciphers use a more complex structure, which was invented by Feistel and his team while they were developing the Mark XII IFF in the late 1950s and early 1960s. Feistel then moved to IBM and founded a research group that produced the Data Encryption Standard (DES) algorithm, which is still a mainstay of payment system security.

A Feistel cipher has the ladder structure shown in Figure 5.12. The input is split up into two blocks, the left half and the right half. A *round function*  $f_1$  of the left half is computed and combined with the right half using exclusive-or (binary addition without carry), though in some Feistel ciphers addition with carry is also used. (We use the notation  $\oplus$  for exclusive-or.) Then, a function  $f_2$  of the right half is computed and combined with the left half, and so on. Finally (if the number of rounds is even) the left half and right half are swapped.

A notation which you may see for the Feistel cipher is  $\psi(f, g, h, \dots)$  where  $f, g, h, \dots$  are the successive round functions. Under this notation, the above cipher is  $\psi(f_1, f_2, \dots, f_{2k-1}, f_{2k})$ . The basic result that enables us to decrypt a Feistel cipher – and indeed the whole point of his design – is that:

$$\psi^{-1}(f_1, f_2, \dots, f_{2k-1}, f_{2k}) = \psi(f_{2k}, f_{2k-1}, \dots, f_2, f_1)$$

In other words, to decrypt, we just use the round functions in the reverse order. Thus the round functions  $f_i$  do not have to be invertible, and the Feistel structure lets us turn any one-way function into a block cipher. This means that we are less constrained in trying to choose a round function with good diffusion and confusion properties, and which also satisfies any other design constraints such as code size, software speed or hardware gate count.

#### 5.4.3.1 The Luby-Rackoff result

The key theoretical result on Feistel ciphers was proved by Mike Luby and Charlie Rackoff in 1988. They showed that if  $f_i$  were random functions, then  $\psi(f_1, f_2, f_3)$  was indistinguishable from a random permutation under chosen-plaintext attack, and this result was soon extended to show that  $\psi(f_1, f_2, f_3, f_4)$  was indistinguishable under chosen plaintext/ciphertext attack – in other words, it was a pseudorandom permutation. (I omit a number of technicalities.)

In engineering terms, the effect is that given a really good round function, four rounds of Feistel are enough. So if we have a hash function in which we have confidence, it is straightforward to construct a block cipher from it: use four rounds of keyed hash in a Feistel network.

#### 5.4.3.2 DES

The DES algorithm is widely used in banking and other payment applications. The ‘killer app’ that got it widely deployed was ATM networks; from there

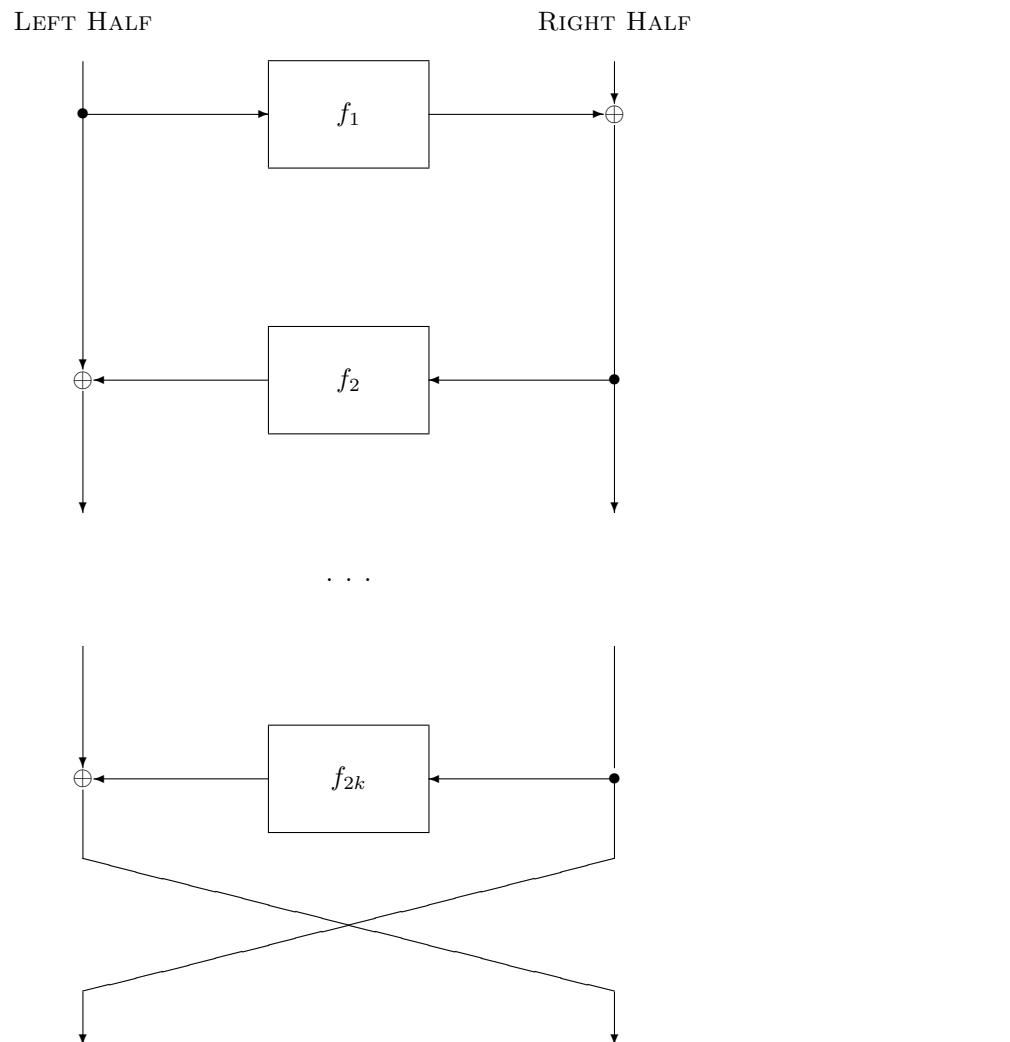


Figure 5.12: – the Feistel cipher structure

it spread to prepayment meters, transport tickets and much else. In its classic form, it is a Feistel cipher, with a 64-bit block and 56-bit key. Its round function operates on 32-bit half blocks and consists of three operations:

- first, the block is expanded from 32 bits to 48;
- next, 48 bits of round key are mixed in using exclusive-or;
- the result is passed through a row of eight S-boxes, each of which takes a six-bit input and provides a four-bit output;
- finally, the bits of the output are permuted according to a fixed pattern.

The effect of the expansion, key mixing and S-boxes is shown in Figure 5.13:

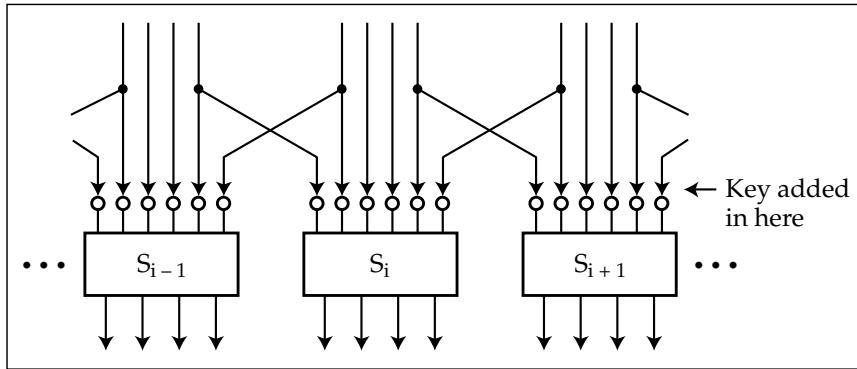


Figure 5.13: – the DES round function

The round keys are derived from the user-supplied key by using each user key bit in twelve different rounds according to a slightly irregular pattern. A full specification of DES is given in [1397].

DES was introduced in 1974 and immediately caused controversy. The most telling criticism was that the key is too short. Someone who wants to find a 56 bit key using brute force, that is by trying all possible keys, will have a *total exhaust time* of  $2^{56}$  encryptions and an *average solution time* of half that, namely  $2^{55}$  encryptions. Whit Diffie and Martin Hellman argued in 1977 that a DES keysearch machine could be built with a million chips, each testing a million keys a second; as a million is about  $2^{20}$ , this would take on average  $2^{15}$  seconds, or a bit over 9 hours, to find the key. They argued that such a machine could be built for \$20 million in 1977 [557]. IBM, whose scientists invented DES, retorted that they would charge the US government \$200 million to build such a machine. (In hindsight, both were right.)

During the 1980's, there were persistent rumors of DES keysearch machines being built by various intelligence agencies, but the first successful public key-search attack took place in 1997. In a distributed effort organised over the net, 14,000 PCs took more than four months to find the key to a challenge. In 1998, the Electronic Frontier Foundation (EFF) built a DES keysearch machine called Deep Crack for under \$250,000, which broke a DES challenge in 3 days. It contained 1,536 chips run at 40MHz, each chip containing 24 search units which

each took 16 cycles to do a test decrypt. The search rate was thus 2.5 million test decryptions per second per search unit, or 60 million keys per second per chip. The design of the cracker is public and can be found at [619]. By 2006, Sandeep Kumar and colleagues at the universities of Bochum and Kiel built a machine using 120 FPGAs and costing \$10,000, which could break DES in 7 days on average [1108]. A modern botnet with 100,000 machines would take a few hours. So the key length of single DES is now inadequate.

Another criticism of DES was that, since IBM kept its design principles secret at the request of the US government, perhaps there was a ‘trapdoor’ which would give them easy access. However, the design principles were published in 1992 after differential cryptanalysis was invented and published [473]. The story was that IBM had discovered these techniques in 1972, and the US National Security Agency (NSA) even earlier. IBM kept the design details secret at the NSA’s request. We’ll discuss the political aspects of all this in 26.2.7.1.

We now have a fairly thorough analysis of DES. The best known *shortcut attack*, that is, a cryptanalytic attack involving less computation than keysearch, is a linear attack using  $2^{42}$  known texts. DES would be secure with more than 20 rounds, but for practical purposes its security is limited by its keylength. I don’t know of any real applications where an attacker might get hold of even  $2^{40}$  known texts. So the known shortcut attacks are not an issue. However, its vulnerability to keysearch makes single DES unusable in most applications. As with AES, there are also attacks based on timing analysis and power analysis.

The usual way of dealing with the DES key length problem is to use the algorithm multiple times with different keys. Banking networks have largely moved to *triple-DES*, a standard since 1999 [1397]. Triple-DES does an encryption, then a decryption, and then a further encryption, all with independent keys. Formally:

$$3DES(k_0, k_1, k_2; M) = DES(k_2; DES^{-1}(k_1; DES(k_0; M)))$$

By setting the three keys equal, you get the same result as a single DES encryption, thus giving a backwards compatibility mode with legacy equipment. (Some banking systems use *two-key triple-DES* which sets  $k_2 = k_0$ ; this gives an intermediate step between single and triple DES.) Most new systems use AES as the default choice, but many banking systems are committed to using block ciphers with an eight-byte block, because of the message formats used in the many protocols by which ATMs, point-of-sale terminals and bank networks talk to each other, and because of the use of block ciphers to generate and protect customer PINs (which I discuss in the chapter on Banking and Bookkeeping). Triple DES is a perfectly serviceable block cipher for such purposes for the foreseeable future.

Another way of preventing keysearch (and making power analysis harder) is *whitening*. In addition to the 56-bit key, say  $k_0$ , we choose two 64-bit whitening keys  $k_1$  and  $k_2$ , xor’ing the first with the plaintext before encryption and the second with the output of the encryption to get the ciphertext afterwards. This composite cipher is known as DESX. Formally,

$$DESX(k_0, k_1, k_2; M) = DES(k_0; M \oplus k_1) \oplus k_2$$

It can be shown that, on reasonable assumptions, DESX has the properties you'd expect; it inherits the differential strength of DES but its resistance to keysearch is increased by the amount of the whitening [1047]. Whitened block ciphers are used in some applications, most specifically in the XTS mode of operation which I discuss below. Nowadays, it's usually used with AES, and AESX is defined similarly, with the whitening keys used to make each block encryption operation unique – as we shall see below in section 5.5.7.

## 5.5 Modes of Operation

A common failure is that cryptographic libraries enable or even encourage developers to use an inappropriate *mode of operation*. This specifies how a block cipher with a fixed block size (8 bytes for DES, 16 for AES) can be extended to process messages of arbitrary length.

There are several standard modes of operation for using a block cipher on multiple blocks [1404]. It is vital to understand them, so you can choose the right one for the job, especially as some common tools provide a weak one by default. This weak mode is electronic code book (ECB) mode, which we discuss next.

### 5.5.1 How not to use a block cipher

In electronic code book mode, we just encrypt each succeeding block of plaintext with our block cipher to get ciphertext, as with the Playfair example above. This is adequate for protocols using single blocks such as challenge-response and some key management tasks; it's also used to encrypt PINs in cash machine systems. But if we use it to encrypt redundant data the patterns will show through, giving an opponent information about the plaintext. For example, figure 5.14 shows what happens to a cartoon image when encrypted using DES in ECB mode. Repeated blocks of plaintext all encrypt to the same ciphertext, leaving the image quite recognisable.

In one popular corporate email system from the last century, the encryption used was DES ECB with the key derived from an eight-character password. If you looked at a ciphertext generated by this system, you saw that a certain block was far more common than the others – the one corresponding to a plaintext of nulls. This gave one of the simplest attacks ever on a fielded DES encryption system: just encrypt a null block with each password in a dictionary and sort the answers. You can now break at sight any ciphertext whose password was one of those in your dictionary.

In addition, using ECB mode to encrypt messages of more than one block length which require authenticity – such as bank payment messages – is particularly foolish, as it opens you to a *cut and splice* attack along the block boundaries. For example, if a bank message said “Please pay account number  $X$  the sum  $Y$ , and their reference number is  $Z$ ” then an attacker might initiate a payment designed so that some of the digits of  $X$  are replaced with some of the digits of  $Z$ .

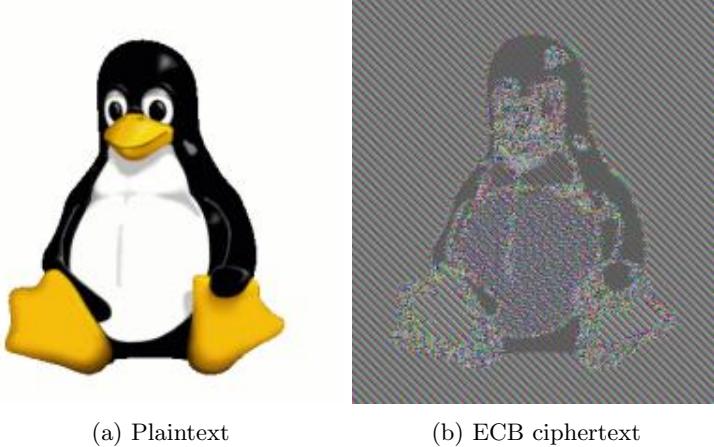


Figure 5.14: The Linux penguin, in clear and ECB encrypted (from Wikipedia, derived from images created by Larry Ewing).

### 5.5.2 Cipher block chaining

Most commercial applications which encrypt more than one block used to use cipher block chaining, or CBC, mode. Like ECB, this was one of the original modes of operation standardised with DES. In it, we exclusive-or the previous block of ciphertext to the current block of plaintext before encryption (see Figure 5.15).

This mode disguises patterns in the plaintext: the encryption of each block depends on all the previous blocks. The input initialisation vector (IV) ensures that stereotyped plaintext message headers won't leak information by encrypting to identical ciphertexts, just as with a stream cipher.

However, an opponent who knows some of the plaintext may be able to cut and splice a message (or parts of several messages encrypted under the same key). In fact, if an error is inserted into the ciphertext, it will affect only two blocks of plaintext on decryption, so if there isn't any integrity protection on the plaintext, an enemy can insert two-block garbles of random data at locations of their choice. For that reason, CBC encryption usually has to be used with a separate authentication code.

More subtle things can go wrong, too; systems have to pad the plaintext to a multiple of the block size, and if a server that decrypts a message and finds incorrect padding signals this fact, whether by returning an ‘invalid padding’ message or just taking longer to respond, then this opens a *padding oracle attack* in which the attacker tweaks input ciphertexts, one byte at a time, watches the error messages, and ends up being able to decrypt whole messages. This was discovered by Serge Vaudenay in 2002; variants of it were used against SSL, IPSEC and TLS as late as 2016 [1949].

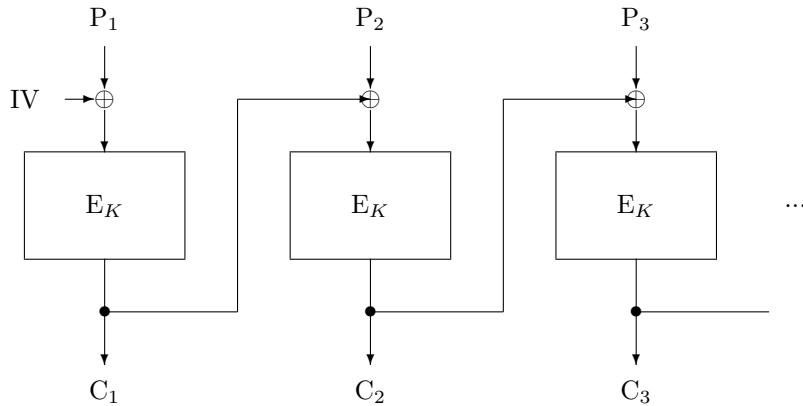


Figure 5.15: – Cipher Block Chaining (CBC) mode

### 5.5.3 Counter encryption

Feedback modes of block cipher encryption are falling from fashion, and not just because of cryptographic issues. They are hard to parallelise. With CBC, a whole block of the cipher must be computed between each block input and each block output. This can be inconvenient in high-speed applications, such as protecting traffic on backbone links. As silicon is cheap, we would rather pipeline our encryption chip, so that it encrypts a new block (or generates a new block of keystream) in as few clock ticks as possible.

The simplest solution is to use AES as a stream cipher. We generate a keystream by encrypting a counter starting at an initialisation vector:  $K_i = \{IV + i\}_K$ , thus expanding the key  $K$  into a long stream of blocks  $K_i$  of keystream, which is typically combined with the blocks of a message  $M_i$  using exclusive-or to give ciphertext  $C_i = M_i \oplus K_i$ .

Additive stream ciphers have two systemic vulnerabilities, as we noted in section 5.2.2 above. The first is an attack in depth: if the same keystream is used twice, then the xor of the two ciphertexts is the xor of the two plaintexts, from which plaintext can often be deduced, as with Venona. The second is that they fail to protect message integrity. Suppose that a stream cipher were used to encipher fund transfer messages. These messages are highly structured; you might know, for example, that bytes 37–42 contain the sum being transferred. You could then cause the data traffic from a local bank to go via your computer, for example by an SS7 exploit. You go into the bank and send \$500 to an accomplice. The ciphertext  $C_i = M_i \oplus K_i$ , duly arrives in your machine. You know  $M_i$  for bytes 37–42, so you can recover  $K_i$  and construct a modified message which instructs the receiving bank to pay not \$500 but \$500,000! This is an example of an *attack in depth*; it is the price not just of the perfect secrecy we get from the one-time pad, but of much more humble stream ciphers, too.

The usual way of dealing with this is to add an authentication code, and the most common standard uses a technique called Galois counter mode, which I describe later.

#### 5.5.4 Legacy stream cipher modes

You may find two old stream-cipher modes of operation, output feedback mode (OFB) and less frequently ciphertext feedback mode (CFB).

Output feedback mode consists of repeatedly encrypting an initial value and using this as a keystream in a stream cipher. Writing IV for the initialization vector, we will have  $K1 = \{IV\}_K$  and  $Ki = \{IV\}_{K(i-1)}$ . However an  $n$ -bit block cipher in OFB mode will typically have a cycle length of  $2^{n/2}$  blocks, after which the birthday theorem will see to it that we loop back to the IV. So we may have a cycle-length problem if we use a 64-bit block cipher such as triple-DES on a high-speed link: once we've called a little over  $2^{32}$  pseudorandom 64-bit values, the odds favour a match. (In CBC mode, too, the birthday theorem ensures that after about  $2^{n/2}$  blocks, we will start to see repeats.) Counter mode encryption, however, has a guaranteed cycle length of  $2^n$  rather than  $2^{n/2}$ , and as we noted above is easy to parallelise. Despite this OFB is still used, as counter mode only became a NIST standard in 2002.

Cipher feedback mode is another kind of stream cipher, designed for use in radio systems that have to resist jamming. It was designed to be self-synchronizing, in that even if we get a burst error and drop a few bits, the system will recover synchronization after one block length. This is achieved by using our block cipher to encrypt the last  $n$  bits of ciphertext, adding the last output bit to the next plaintext bit, and shifting the ciphertext along one bit. But this costs one block cipher operation per bit and has very bad error amplification properties; nowadays people tend to use dedicated link layer protocols for synchronization and error correction rather than trying to combine them with the cryptography at the traffic layer.

#### 5.5.5 Message Authentication Code

Another official mode of operation of a block cipher is not used to encipher data, but to protect its integrity and authenticity. This is the *message authentication code*, or MAC. To compute a MAC on a message using a block cipher, we encrypt it using CBC mode and throw away all the output ciphertext blocks except the last one; this last block is the MAC. (The intermediate results are kept secret in order to prevent splicing attacks.)

This construction makes the MAC depend on all the plaintext blocks as well as on the key. It is secure provided the message length is fixed; Mihir Bellare, Joe Kilian and Philip Rogaway proved that any attack on a MAC under these circumstances would give an attack on the underlying block cipher [211].

If the message length is variable, you have to ensure that a MAC computed on one string can't be used as the IV for computing a MAC on a different string, so that an opponent can't cheat by getting a MAC on the composition of the two

strings. In order to fix this problem, NIST has standardised CMAC, in which a variant of the key is xor-ed in before the last encryption [1405]. (CMAC is based on a proposal by Tetsu Iwata and Kaoru Kurosawa [965].) You may see legacy systems in which the MAC consists of only half of the last output block, with the other half thrown away, or used in other mechanisms.

There are other possible constructions of MACs: the most common one is HMAC, which uses a hash function with a key; we'll describe it in section 5.6.2.

### 5.5.6 Galois Counter Mode

The above modes were all developed for DES in the 1970s and 1980s (although counter mode only became an official US government standard in 2002). They are not efficient for bulk encryption where you need to protect integrity as well as confidentiality; if you use either CBC mode or counter mode to encrypt your data and a CBC-MAC or CMAC to protect its integrity, then you invoke the block cipher twice for each block of data you process, and the operation cannot be parallelised.

The modern approach is to use a mode of operation designed for authenticated encryption. Galois Counter Mode (GCM) has taken over as the default since being approved by NIST in 2007 [1407]. It uses only one invocation of the block cipher per block of text, and it's parallelisable so you can get high throughput on fast data links with low cost and low latency. Encryption is performed in a variant of counter mode; the resulting ciphertexts are also used as coefficients of a polynomial which is evaluated at a key-dependent point over a Galois field of  $2^{128}$  elements to give an authenticator tag. The tag computation is a universal hash function of the kind I described in section 5.2.4 and is provably secure so long as keys are never reused. The supplied key is used along with a random IV to generate both a unique message key and a unique authenticator key. The output is thus a ciphertext of the same length as the plaintext, plus an IV and a tag of typically 128 bits each.

GCM also has an interesting incremental property: a new authenticator and ciphertext can be calculated with an amount of effort proportional to the number of bits that were changed. GCM was invented by David McGrew and John Viega of Cisco; their goal was to create an efficient authenticated encryption mode suitable for use in high-performance network hardware [1268]. GCM is the sensible default for authenticated encryption of bulk content. (There's an earlier composite mode, CCM, which you'll find used in Bluetooth 4.0 and later; this combines counter mode with CBC-MAC, so it costs about twice as much effort to compute, and cannot be parallelised or recomputed incrementally [1406].)

### 5.5.7 XTS

GCM and other authenticated encryption modes expand the plaintext by adding a message key and an authenticator tag. This is very inconvenient in applications such as hard disk encryption, where we prefer a mode of operation that preserves plaintext length. Disk encryption systems used to use CBC with the sector number providing an IV, but since Windows 10, Microsoft has been using a new

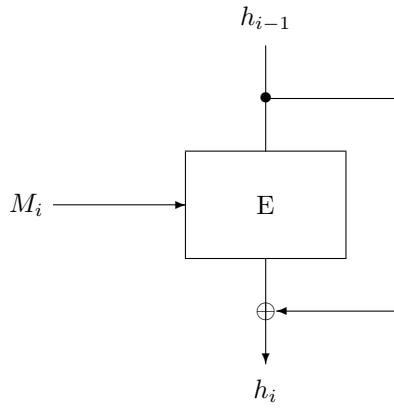


Figure 5.16: – feedforward mode (hash function)

mode of operation, XTS-AES, inspired by GCM and standardised in 2007. This is a codebook mode but with the plaintext whitened by a *tweak key* derived from the disk sector. Formally, the message  $M_i$  encrypted with the key  $K$  at block  $j$  is

$$AESX(KT_j, K, KT_j; M)$$

where the tweak key  $KT_j$  is derived by encrypting the IV using a different key and then multiplying it repeatedly with a suitable constant so as to give a different whitener for each block. This means that if an attacker swaps two encrypted blocks, all 256 bits will decrypt to randomly wrong values. You still need higher-layer mechanisms to detect ciphertext manipulation, but simple checksums will be sufficient.

## 5.6 Hash Functions

In section 5.4.3.1 I showed how the Luby-Rackoff theorem enables us to construct a block cipher from a hash function. It's also possible to construct a hash function from a block cipher<sup>5</sup>. The trick is to feed the message blocks one at a time to the key input of our block cipher, and use it to update a hash value (which starts off at say  $H_0 = 0$ ). In order to make this operation non-invertible, we add feedforward: the  $(i - 1)$ st hash value is exclusive or'd with the output of round  $i$ . This *Davies-Meyer construction* gives our final mode of operation of a block cipher (Figure 5.16).

---

<sup>5</sup>In fact, we can also construct hash functions and block ciphers from stream ciphers – so, subject to some caveats I'll discuss in the next section, given any one of these three primitives we can construct the other two.

The birthday theorem makes another appearance here, in that if a hash function  $h$  is built using an  $n$  bit block cipher, it is possible to find two messages  $M_1 \neq M_2$  with  $h(M_1) = h(M_2)$  with about  $2^{n/2}$  effort (hash slightly more than that many messages  $M_i$  and look for a match). So a 64 bit block cipher is not adequate, as forging a message would cost of the order of  $2^{32}$  messages, which is just too easy. A 128-bit cipher such as AES used to be just about adequate, and in fact the AACPS content protection mechanism in Blu-ray DVDs used ‘AES-H’, the hash function derived from AES in this way.

### 5.6.1 Common hash functions

The hash functions most commonly used through the 1990s and 2000s evolved as variants of a block cipher with a 512 bit key and a block size increasing from 128 to 512 bits. The first two were designed by Ron Rivest and the others by the NSA:

- MD4 has three rounds and a 128 bit hash value, and a collision was found for it in 1998 [568];
- MD5 has four rounds and a 128 bit hash value, and a collision was found for it in 2004 [1979, 1981];
- SHA-1, released in 1995, has five rounds and a 160 bit hash value. A collision was found in 2017 [1828], and a more powerful version of the attack in 2020 [1146];
- SHA-2, which replaced it in 2002, comes in 256-bit and 512-bit versions (called SHA256 and SHA512) plus a number of variants.

The block ciphers underlying these hash functions are similar: their round function is a complicated mixture of the register operations available on 32 bit processors [1667]. Cryptanalysis has advanced steadily. MD4 was broken by Hans Dobbertin in 1998 [568]; MD5 was broken by Xiaoyun Wang and her colleagues in 2004 [1979, 1981]; collisions can now be found easily, even between strings containing meaningful text and adhering to message formats such as those used for digital certificates. Wang seriously dented SHA-1 the following year, providing an algorithm to find collisions in only  $2^{69}$  steps [1980]; it now takes about  $2^{60}$  computations. In February 2017, scientists from Amsterdam and Google published one, to prove the point and help persuade people to move to stronger hash functions such as SHA-2 [1828] (and from earlier versions of TLS to TLS 1.3). In 2020, Gaëtan Leurent and Thomas Peyrin developed an improved attack that computes chosen-prefix collisions, enabling certificate forgery at a cost of several tens of thousands of dollars [1146].

In 2007, the US National Institute of Standards and Technology (NIST) organised a competition to find a replacement hash function family [1409]. The winner, Keccak, has a quite different internal structure, and was standardised as SHA-3 in 2015. So we now have a choice of SHA-2 and SHA-3 as standard hash functions.

A lot of deployed systems still use hash functions such as MD5 for which there's an easy collision-search algorithm. Whether a collision will break any given application can be a complex question. I already mentioned forensic systems, which keep hashes of files on seized computers, to reassure the court that the police didn't tamper with the evidence; a hash collision would merely signal that someone had been trying to tamper, whether the police or the defendant, and trigger a more careful investigation. If bank systems actually took a message composed by a customer saying 'Pay  $X$  the sum  $Y$ ', hashed it and signed it, then a crook could find two messages 'Pay  $X$  the sum  $Y$ ' and 'Pay  $X$  the sum  $Z$ ' that hashed to the same value, get one signed, and swap it for the other. But bank systems don't work like that. They typically use MACs rather than digital signatures on actual transactions, and logs are kept by all the parties to a transaction, so it's not easy to sneak in one of a colliding pair. And in both cases you'd probably have to find a preimage of an existing hash value, which is a much harder cryptanalytic task than finding a collision.

### 5.6.2 Hash function applications – HMAC, commitments and updating

But even though there may be few applications where a collision-finding algorithm could let a bad guy to steal real money today, the existence of a vulnerability can still undermine a system's value. Some people doing forensic work continue to use MD5, as they've used it for years, and its collisions don't give useful attacks. This is probably a mistake. In 2005, a motorist accused of speeding in Sydney, Australia was acquitted after the New South Wales Roads and Traffic Authority failed to find an expert to testify that MD5 was secure in this application. The judge was "not satisfied beyond reasonable doubt that the photograph [had] not been altered since it was taken" and acquitted the motorist; his strange ruling was upheld on appeal the following year [1432]. So even if a vulnerability doesn't present an engineering threat, it can still present a *certificational* threat.

Hash functions have many other uses. One of them is to compute MACs. A naïve method would be to hash the message with a key:  $\text{MAC}_k(M) = h(k, M)$ . However the accepted way of doing this, called HMAC, uses an extra step in which the result of this computation is hashed again. The two hashing operations are done using variants of the key, derived by exclusive-or'ing them with two different constants. Thus  $\text{HMAC}_k(M) = h(k \oplus B, h(k \oplus A, M))$ .  $A$  is constructed by repeating the byte 0x36 as often as necessary, and  $B$  similarly from the byte 0x5C. If a hash function is on the weak side, this construction can make exploitable collisions harder to find [1089]. HMAC is now FIPS 198-1.

Another use of hash functions is to make commitments that are to be revealed later. For example, I might wish to timestamp a digital document in order to establish intellectual priority, but not reveal the contents yet. In that case, I can publish a hash of the document, or send it to a commercial timestamping service, or have it mined into the Bitcoin blockchain. Later, when I reveal the document, the timestamp on its hash establishes that I had written it by then. Again, an algorithm that generates colliding pairs doesn't break this, as you have to have the pair to hand when you do the timestamp.

*Merkle trees* hash a large number of inputs to a single hash output. The inputs are hashed to values that form the leaves of a tree; each non-leaf node contains the hash of all the hashes at its child nodes, so the hash at the root is a hash of all the values at the leaves. This is a fast way to hash a large data structure; it's used in code signing, where you may not want to wait for all of an application's files to have their signatures checked before you open it. It's also widely used in blockchain applications; in fact, a blockchain is just a Merkle tree. It was invented by Ralph Merkle, who first proposed it to calculate a short hash of a large file of public keys [1296], particularly for systems where public keys are used only once. For example, a Lamport digital signature can be constructed from a hash function: you create a private key of 512 random 256-bit values  $k_i$  and publish the verification key  $V$  as their Merkle tree hash. Then to sign  $h = \text{SHA256}(M)$  you would reveal  $k_{2i}$  if the  $i$ -th bit of  $h$  is zero, and otherwise reveal  $k_{2i+1}$ . This is secure if the hash function is, but has the drawback that each key can be used only once. Merkle saw that you could generate a series of private keys by encrypting a counter with a master secret key, and then use a tree to hash the resulting public keys. However, for most purposes, people use signature algorithms based on number theory, which I'll describe in the next section.

One security-protocol use of hash functions is worth a mention: key updating and autokeying. *Key updating* means that two or more principals who share a key pass it through a one-way hash function at agreed times:  $K_i = h(K_{i-1})$ . The point is that if an attacker compromises one of their systems and steals the key, he only gets the current key and is unable to decrypt back traffic. The chain of compromise is broken by the hash function's one-wayness. This property is also known as *backward security*. A variant is *autokeying* where the principals update a key by hashing it with the messages they have exchanged since the last key change:  $K_{i+1} = h(K_i, M_{i1}, M_{i2}, \dots)$ . If an attacker now compromises one of their systems and steals the key, then as soon as they exchange a message which he can't observe or guess, security will be recovered; again, the chain of compromise is broken. This property is known as *forward security*. It was first used in banking in EFT payment terminals in Australia [207, 209]. The use of asymmetric cryptography allows a slightly stronger form of forward security, namely that as soon as a compromised terminal exchanges a message with an uncompromised one which the opponent doesn't control, security can be recovered even if the message is in plain sight. I'll describe how this works next.

## 5.7 Asymmetric crypto primitives

The commonly used building blocks in *asymmetric cryptography*, public-key encryption and digital signature are based on number theory. I'll give a brief overview here, and look in more detail at some of the mechanisms in Part II when I discuss applications.

The basic idea is to make the security of the cipher depend on the difficulty of solving a mathematical problem that's known to be hard, in the sense that a lot of people have tried to solve it and failed. The two problems used in almost

all real systems are factorization and discrete logarithm.

### 5.7.1 Cryptography based on factoring

The *prime numbers* are the positive whole numbers with no proper divisors: the only numbers that divide a prime number are 1 and the number itself. By definition, 1 is not prime; so the primes are {2, 3, 5, 7, 11, ...}. The *fundamental theorem of arithmetic* states that each natural number greater than 1 factors into prime numbers in a way that is unique up to the order of the factors. It is easy to find prime numbers and multiply them together to give a composite number, but much harder to resolve a composite number into its factors. And lots of smart people have tried really hard since we started using cryptography based on factoring. The largest composite product of two large random primes to have been factorized in 2020 was RSA-250, an 829-bit number (250 decimal digits). This took the equivalent of 2700 years' work on a single 2.2GHz core; the previous record, RSA-240 in 2019, had taken the equivalent of 900 years [301]. It is possible for factoring to be done surreptitiously, perhaps using a botnet; in 2001, when the state of the art was factoring 512-bit numbers, such a challenge was set in Simon Singh's 'Code Book' and solved by five Swedish students using several hundred computers to which they had access [43]. As for 1024-bit numbers, I expect the NSA can factor them already, and I noted in the second edition that 'an extrapolation of the history of factoring records suggests the first factorization will be published in 2018.' Moore's law is slowing down, and we're a year late. Anyway, organisations that want keys to remain secure for many years are already using 2048-bit numbers at least.

The algorithm commonly used to do public-key encryption and digital signatures based on factoring is RSA, named after its inventors Ron Rivest, Adi Shamir and Len Adleman. It uses *Fermat's little theorem*, which states that for all primes  $p$  not dividing  $a$ ,  $a^{p-1} \equiv 1 \pmod{p}$  (proof: take the set {1, 2, ...,  $p - 1$ } and multiply each of them modulo  $p$  by  $a$ , then cancel out  $(p - 1)!$  each side). For a general integer  $n$ ,  $a^{\phi(n)} \equiv 1 \pmod{n}$  where Euler's function  $\phi(n)$  is the number of positive integers less than  $n$  with which it has no divisor in common (the proof is similar). So if  $n$  is the product of two primes  $pq$  then  $\phi(n) = (p - 1)(q - 1)$ .

In RSA, the encryption key is a modulus  $N$  which is hard to factor (take  $N = pq$  for two large randomly chosen primes  $p$  and  $q$ , say of 1024 bits each) plus a public exponent  $e$  that has no common factors with either  $p - 1$  or  $q - 1$ . The private key is the factors  $p$  and  $q$ , which are kept secret. Where  $M$  is the message and  $C$  is the ciphertext, encryption is defined by

$$C \equiv M^e \pmod{N}$$

Decryption is the reverse operation:

$$M \equiv \sqrt[e]{C} \pmod{N}$$

Whoever knows the private key – the factors  $p$  and  $q$  of  $N$  – can easily calculate  $\sqrt[e]{C} \pmod{N}$ . As  $\phi(N) = (p - 1)(q - 1)$  and  $e$  has no common

factors with  $\phi(N)$ , the key's owner can find a number  $d$  such that  $de \equiv 1 \pmod{\phi(N)}$  – she finds the value of  $d$  separately modulo  $p - 1$  and  $q - 1$ , and combines the answers.  $\sqrt[N]{C} \pmod{N}$  is now computed as  $C^d \pmod{N}$ , and decryption works because of Fermat's theorem:

$$C^d \equiv \{M^e\}^d \equiv M^{ed} \equiv M^{1+k\phi(N)} \equiv M \cdot M^{k\phi(N)} \equiv M \cdot 1 \equiv M \pmod{N}$$

Similarly, the owner of a private key can operate on a message with it to produce a signature

$$\text{Sig}_d(M) \equiv M^d \pmod{N}$$

and this signature can be verified by raising it to the power  $e \pmod{N}$  (thus, using  $e$  and  $N$  as the public signature verification key) and checking that the message  $M$  is recovered:

$$M \equiv (\text{Sig}_d(M))^e \pmod{N}$$

Neither RSA encryption nor signature is safe to use on its own. The reason is that, as encryption is an algebraic process, it preserves certain algebraic properties. For example, if we have a relation such as  $M_1 M_2 = M_3$  that holds among plaintexts, then the same relationship will hold among ciphertexts  $C_1 C_2 = C_3$  and signatures  $\text{Sig}_1 \text{Sig}_2 = \text{Sig}_3$ . This property is known as a *multiplicative homomorphism*; a homomorphism is a function that preserves some mathematical structure. The homomorphic nature of raw RSA means that it doesn't meet the random oracle model definitions of public key encryption or signature.

Another general problem with public-key encryption is that if the plaintexts are drawn from a small set, such as ‘attack’ or ‘retreat’, and the encryption process is deterministic (as RSA is), then an attacker might just precompute the possible ciphertexts and recognise them when they appear. With RSA, it's also dangerous to use a small exponent  $e$  to encrypt the same message to multiple recipients, as this can lead to an algebraic attack. To stop the guessing attack, the low-exponent attack and attacks based on homomorphism, it's sensible to add in some randomness, and some redundancy, into a plaintext block before encrypting it. Every time we encrypt the same short message, say ‘attack’, we want to get a completely different ciphertext, and for these to be indistinguishable from each other as well as from the ciphertexts for ‘retreat’. And there are good ways and bad ways of doing this.

Crypto theoreticians have wrestled for decades to analyse all the things that can go wrong with asymmetric cryptography, and to find ways to tidy it up. Shafi Goldwasser and Silvio Micali came up with formal models of *probabilistic encryption* in which we add randomness to the encryption process, and *semantic security*, which we mentioned already; in this context it means that an attacker cannot get any information at all about a plaintext  $M$  that was encrypted to a ciphertext  $C$ , even if he is allowed to request the decryption of any other ciphertext  $C'$  not equal to  $C$  [777]. In other words, we want the encryption to resist chosen-ciphertext attack as well as chosen-plaintext attack. There are a

number of constructions that give semantic security, but they tend to be too ungainly for practical use.

The usual real-world solution is *optimal asymmetric encryption padding* (OAEP), where we concatenate the message  $M$  with a random nonce  $N$ , and use a hash function  $h$  to combine them:

$$C_1 = M \oplus h(N)$$

$$C_2 = N \oplus h(C_1)$$

In effect, this is a two-round Feistel cipher that uses  $h$  as its round function. The result, the combination  $C_1, C_2$ , is then encrypted with RSA and sent. The recipient then computes  $N$  as  $C_2 \oplus h(C_1)$  and recovers  $M$  as  $C_1 \oplus h(N)$  [212]. This was eventually proven to be secure. There are a number of public-key cryptography standards; PKCS #1 describes OAEP [993]. These block a whole lot of attacks that were discovered in the 20th century and about which people have mostly forgotten, such as the fact that an opponent can detect if you encrypt the same message with two different RSA keys. In fact, one of the things we learned in the 1990s was that randomisation helps make crypto protocols more robust against all sorts of attacks, and not just the mathematical ones. Side-channel attacks and even physical probing of devices take a lot more work.

With signatures, things are slightly simpler. In general, it's often enough to just hash the message before applying the private key:  $Sig_d = [h(M)]^d \pmod{N}$ ; PKCS #7 describes simple mechanisms for signing a message digest [1008]. However, in some applications one might wish to include further data in the signature block, such as a timestamp, or some randomness to make side-channel attacks harder.

Many of the things that have gone wrong with real implementations have to do with side channels and error handling. One spectacular example was when Daniel Bleichenbacher found a way to break the RSA implementation in SSL v 3.0 by sending suitably chosen ciphertexts to the victim and observing any resulting error messages. If he could learn from the target whether a given  $c$ , when decrypted as  $c^d \pmod{n}$ , corresponds to a PKCS #1 message, then he could use this to decrypt or sign messages [264]. There have been many more side-channel attacks on common public-key implementations, typically via measuring the precise time taken to decrypt. RSA is also mathematically fragile; you can break it using homomorphisms, or if you have the same ciphertext encrypted under too many different small keys, or if the message is too short, or if two messages are related by a known polynomial, or in several other edge cases. Errors in computation can also give a result that's correct modulo one factor of the modulus and wrong modulo the other, enabling the modulus to be factored; errors can be inserted tactically, by interfering with the crypto device, or strategically, for example by the chipmaker arranging for one particular value of a 64-bit multiply to be computed incorrectly. Yet other attacks have involved stack overflows, whether by sending the attack code in as keys, or as padding in poorly-implemented standards.

### 5.7.2 Cryptography based on discrete logarithms

While RSA was the first public-key encryption algorithm deployed in the SSL and SSH protocols, the most popular public-key algorithms now are based on discrete logarithms. There are a number of flavors, some using normal modular arithmetic while others use *elliptic curves*. I'll explain the normal case first.

A *primitive root* modulo  $p$  is a number whose powers generate all the nonzero numbers mod  $p$ ; for example, when working modulo 7 we find that  $5^2 = 25$  which reduces to 4 (modulo 7), then we can compute  $5^3$  as  $5^2 \times 5$  or  $4 \times 5$  which is 20, which reduces to 6 (modulo 7), and so on, as in Figure 5.17:

$$\begin{aligned} 5^1 &= 5 \pmod{7} \\ 5^2 &= 25 \equiv 4 \pmod{7} \\ 5^3 &\equiv 4 \times 5 \equiv 6 \pmod{7} \\ 5^4 &\equiv 6 \times 5 \equiv 2 \pmod{7} \\ 5^5 &\equiv 2 \times 5 \equiv 3 \pmod{7} \\ 5^6 &\equiv 3 \times 5 \equiv 1 \pmod{7} \end{aligned}$$

Figure 5.17 – example of discrete logarithm calculations

Thus 5 is a primitive root modulo 7. This means that given any  $y$ , we can always solve the equation  $y = 5^x \pmod{7}$ ;  $x$  is then called the discrete logarithm of  $y$  modulo 7. Small examples like this can be solved by inspection, but for a large random prime number  $p$ , we do not know how to do this efficiently. So the mapping  $f : x \rightarrow g^x \pmod{p}$  is a one-way function, with the additional properties that  $f(x+y) = f(x)f(y)$  and  $f(nx) = f(x)^n$ . In other words, it is a *one-way homomorphism*. As such, it can be used to construct digital signature and public key encryption algorithms.

#### 5.7.2.1 One-way commutative encryption

Imagine we're back in ancient Rome, that Anthony wants to send a secret to Brutus, and the only communications channel available is an untrustworthy courier (say, a slave belonging to Caesar). Anthony can take the message, put it in a box, padlock it, and get the courier to take it to Brutus. Brutus could then put his own padlock on it too, and have it taken back to Anthony. He in turn would remove his padlock, and have it taken back to Brutus, who would now at last open it.

Exactly the same can be done using a suitable encryption function that commutes, that is, has the property that  $\{\{M\}_{KA}\}_{KB} = \{\{M\}_{KB}\}_{KA}$ . Alice can take the message  $M$  and encrypt it with her key  $KA$  to get  $\{M\}_{KA}$  which she sends to Bob. Bob encrypts it again with his key  $KB$  getting  $\{\{M\}_{KA}\}_{KB}$ . But the commutativity property means that this is just  $\{\{M\}_{KB}\}_{KA}$ , so Alice can decrypt it using her key  $KA$  getting  $\{M\}_{KB}$ . She sends this to Bob and he can decrypt it with  $KB$ , finally recovering the message  $M$ .

How can a suitable commutative encryption be implemented? The one-time pad does indeed commute, but is not suitable here. Suppose Alice chooses a random key  $xA$  and sends Bob  $M \oplus xA$  while Bob returns  $M \oplus xB$  and Alice finally sends him  $M \oplus xA \oplus xB$ , then an attacker can simply exclusive-or these

three messages together; as  $X \oplus X = 0$  for all  $X$ , the two values of  $xA$  and  $xB$  both cancel out, leaving the plaintext  $M$ .

The discrete logarithm problem comes to the rescue. If the discrete log problem based on a primitive root modulo  $p$  is hard, then we can use discrete exponentiation as our encryption function. For example, Alice encodes her message as the primitive root  $g$ , chooses a random number  $xA$ , calculates  $g^{xA}$  modulo  $p$  and sends it, together with  $p$ , to Bob. Bob likewise chooses a random number  $xB$  and forms  $g^{xAxB}$  modulo  $p$ , which he passes back to Alice. Alice can now remove her exponentiation: using Fermat's theorem, she calculates  $g^{xB} = (g^{xAxB})^{(p-xA)} \pmod{p}$  and sends it to Bob. Bob can now remove his exponentiation, too, and so finally gets hold of  $g$ . The security of this scheme depends on the difficulty of the discrete logarithm problem. In practice, it can be tricky to encode a message as a primitive root; but there's a simpler way to achieve the same effect.

### 5.7.2.2 Diffie-Hellman key establishment

The first public-key encryption scheme to be published, by Whitfield Diffie and Martin Hellman in 1976, has a fixed primitive root  $g$  and uses  $g^{xAxB}$  modulo  $p$  as the key to a shared-key encryption system. The values  $xA$  and  $xB$  can be the private keys of the two parties.

Let's walk through this. The prime  $p$  and generator  $g$  are common to all users. Alice chooses a secret random number  $xA$ , calculates  $yA = g^{xA}$  and publishes it opposite her name in the company phone book. Bob does the same, choosing a random number  $xB$  and publishing  $yB = g^{xB}$ . In order to communicate with Bob, Alice fetches  $yB$  from the phone book, forms  $yB^{xA}$  which is just  $g^{xAxB}$ , and uses this to encrypt the message to Bob. On receiving it, Bob looks up Alice's public key  $yA$  and forms  $yA^{xB}$  which is also equal to  $g^{xAxB}$ , so he can decrypt her message.

Alternatively, Alice and Bob can use transient keys, and get a mechanism for providing forward security. As before, let the prime  $p$  and generator  $g$  be common to all users. Alice chooses a random number  $R_A$ , calculates  $g^{R_A}$  and sends it to Bob; Bob does the same, choosing a random number  $R_B$  and sending  $g^{R_B}$  to Alice; they then both form  $g^{R_A R_B}$ , which they use as a session key (see Figure 5.19).

$$\begin{aligned} A \rightarrow B : & \quad g^{R_A} \pmod{p} \\ B \rightarrow A : & \quad g^{R_B} \pmod{p} \\ A \rightarrow B : & \quad \{M\}_{g^{R_A R_B}} \end{aligned}$$

Figure 5.18 – the Diffie-Hellman key exchange protocol

Alice and Bob can now use the session key  $g^{R_A R_B}$  to encrypt a conversation. If they used transient keys, rather than long-lived ones, they have managed to create a shared secret ‘out of nothing’. Even if an opponent had inspected both their machines before this protocol was started, and knew all their stored private keys, then provided some basic conditions were met (e.g., that their random number generators were not predictable and no malware was left behind)

the opponent could still not eavesdrop on their traffic. This is the strong version of the forward security property to which I referred in section 5.6.2. The opponent can't work forward from knowledge of previous keys which he might have obtained. Provided that Alice and Bob both destroy the shared secret after use, they will also have backward security: an opponent who gets access to their equipment subsequently cannot work backward to break their old traffic. In what follows, we may write the Diffie-Hellman key derived from  $R_A$  and  $R_B$  as  $DH(R_A, R_B)$  when we don't have to be explicit about which group we're working in and don't need to write out explicitly which is the private key  $R_A$  and which is the public key  $g^{R_A}$ .

Slightly more work is needed to provide a full solution. Some care is needed when choosing the parameters  $p$  and  $g$ ; we can infer from the Snowden disclosures, for example, that the NSA can solve the discrete logarithm problem for commonly-used 1024-bit prime numbers<sup>6</sup>. And there are several other details which depend on whether we want properties such as forward security.

But this protocol has a small problem: although Alice and Bob end up with a session key, neither of them has any real idea who they share it with.

Suppose that in our padlock protocol Caesar had just ordered his slave to bring the box to him instead, and placed his own padlock on it next to Anthony's. The slave takes the box back to Anthony, who removes his padlock, and brings the box back to Caesar who opens it. Caesar can even run two instances of the protocol, pretending to Anthony that he's Brutus and to Brutus that he's Anthony. One fix is for Anthony and Brutus to apply their seals to their locks.

With the Diffie-Hellman protocol, the same idea leads to a middleperson attack. Charlie intercepts Alice's message to Bob and replies to it; at the same time, he initiates a key exchange with Bob, pretending to be Alice. He ends up with a key  $DH(R_A, R_C)$  which he shares with Alice, and another key  $DH(R_B, R_C)$  which he shares with Bob. So long as he continues to sit in the middle of the network and translate the messages between them, they may have a hard time detecting that their communications are compromised. The usual solution is to authenticate transient keys, and there are various possibilities.

In the STU-2 telephone, which is now obsolete but which you can see in the NSA museum at Fort Meade, the two principals would read out an eight-digit hash of the key they had generated and check that they had the same value before starting to discuss classified matters. Something similar is implemented in Bluetooth versions 4 and later, but is complicated by the many versions that the protocol has evolved to support devices with different user interfaces. The protocol has suffered from multiple attacks, most recently the key negotiation of Bluetooth (KNOB) attack, which allows a middleperson to force one-byte keys that are easily brute forced; all devices produced before 2018 are vulnerable [123]. The standard allows for key lengths between one and sixteen bytes;

---

<sup>6</sup>The likely discrete log algorithm, NFS, involves a large computation for each prime number followed by a smaller computation for each discrete log modulo that prime number. The open record is 795 bits, which took 3,100 core-years in 2019 [?], using a version of NFS that's three times more efficient than ten years ago. There have been persistent rumours of a further NSA improvement and in any case the agency can throw a lot more horsepower at an important calculation.

as the keylength negotiation is performed in the clear, an attacker can force the length to the lower limit. All standards-compliant chips are vulnerable; this may be yet more of the toxic waste from the Crypto Wars, which I discuss in section 26.2.7. Earlier versions of Bluetooth are more like the ‘just-works’ mode of the HomePlug protocol described in section 14.3.3.3 in that they were principally designed to help you set up a pairing key with the right device in a benign environment, rather than defending against a sophisticated attack in a hostile one. The more modern ones appear to be better, but it’s really just theatre.

So many things go wrong: protocols that will generate or accept very weak keys and thus give only the appearance of protection; programs that leak keys via side channels such as the length of time they take to decrypt; and software vulnerabilities leading to stack overflows and other hacks. If you’re implementing public-key cryptography you need to consult up-to-date standards, use properly accredited toolkits, and get someone knowledgeable to evaluate what you’ve done. And please don’t write the actual crypto code on your own – doing it properly requires a lot of different skills, from computational number theory to side-channel analysis and formal methods. Even using good crypto libraries gives you plenty of opportunities to shoot your foot off.

#### 5.7.2.3 ElGamal digital signature and DSA

Suppose that the base  $p$  and the generator  $g$  are public values chosen in some suitable way, and that each user who wishes to sign messages has a private signing key  $X$  with a public signature verification key  $Y = g^X$ . An ElGamal signature scheme works as follows. Choose a message key  $k$  at random, and form  $r = g^k \pmod{p}$ . Now form the signature  $s$  using a linear equation in  $k$ ,  $r$ , the message  $M$  and the private key  $X$ . There are a number of equations that will do; the one that happens to be used in ElGamal signatures is

$$rX + sk = M$$

So  $s$  is computed as  $s = (M - rX)/k$ ; this is done modulo  $\phi(p)$ . When both sides are passed through our one-way homomorphism  $f(x) = g^x \pmod{p}$  we get:

$$g^{rX} g^{sk} \equiv g^M$$

or

$$Y^r r^s \equiv g^M$$

An ElGamal signature on the message  $M$  consists of the values  $r$  and  $s$ , and the recipient can verify it using the above equation.

A few more details need to be fixed up to get a functional digital signature scheme. As before, bad choices of  $p$  and  $g$  can weaken the algorithm. We will also want to hash the message  $M$  using a hash function so that we can sign messages of arbitrary length, and so that an opponent can’t use the algorithm’s algebraic structure to forge signatures on messages that were never signed. Having attended to these details and applied one or two optimisations, we get the

*Digital Signature Algorithm* (DSA) which is a US standard and widely used in government applications.

DSA assumes a prime  $p$  of typically 2048 bits<sup>7</sup>, a prime  $q$  of 256 bits dividing  $(p - 1)$ , an element  $g$  of order  $q$  in the integers modulo  $p$ , a secret signing key  $x$  and a public verification key  $y = g^x$ . The signature on a message  $M$ ,  $\text{Sig}_x(M)$ , is  $(r, s)$  where

$$r \equiv (g^k \pmod p) \pmod q$$

$$s \equiv (h(M) - xr)/k \pmod q$$

The hash function used by default is SHA256<sup>8</sup>.

DSA is the classic example of a randomised digital signature scheme without message recovery. The most commonly-used version nowadays is ECDSA, a variant based on elliptic curves, which we'll discuss now – this is for example the standard for cryptocurrency and increasingly also for certificates in bank smartcards.

### 5.7.3 Elliptic curve cryptography

Discrete logarithms and their analogues exist in many other mathematical structures. *Elliptic curve cryptography* uses discrete logarithms on an elliptic curve – a curve given by an equation like  $y^2 = x^3 + ax + b$ . These curves have the property that you can define an addition operation on them and the resulting *Mordell group* can be used for cryptography. The algebra gets a bit complex and this book isn't the place to set it out<sup>9</sup>. However, elliptic curve cryptosystems are interesting for at least two reasons.

First is performance; they give versions of the familiar primitives such as Diffie-Hellmann key exchange and the Digital Signature Algorithm that use less computation, and also have shorter variables; both are welcome in constrained environments. Elliptic curve cryptography is used in applications from the latest versions of EMV payment cards to Bitcoin.

Second, some elliptic curves have a *bilinear pairing* which Dan Boneh and Matt Franklin used to construct cryptosystems where your public key is your name [286]. Recall that in RSA and Diffie-Hellmann, the user chose his private key and then computed a corresponding public key. In a so-called *identity-based cryptosystem*, you choose your identity then go to a central authority that issues you with a private key corresponding to that identity. There is a global public

<sup>7</sup>In the 1990s  $p$  could be in the range 512–1024 bits and  $q$  160 bits; this was changed to 1023–1024 bits in 2001 [1402] and 1024–3072 bits in 2009, with  $q$  in the range 160–256 bits [1403].

<sup>8</sup>The default sizes of  $p$  are chosen to be 2048 bits and  $q$  256 bits in order to equalise the work factors of the two best known cryptanalytic attacks, namely the number field sieve whose running speed depends on the size of  $p$  and Pollard's rho which depends on the size of  $q$ . Larger sizes can be chosen if you're anxious about Moore's law or about progress in algorithms.

<sup>9</sup>See Katz and Lindell [1022] for an introduction.

key, with which anyone can encrypt a message to your identity; you can decrypt this using your private key. Earlier, Adi Shamir had discovered *identity-based signature schemes* that allow you to sign messages using a private key so that anyone can verify the signature against your name [1704]. In both cases, your private key is computed by the central authority using a system-wide private key known only to itself. Identity-based primitives have been used in a few specialist systems: in Zcash for the payment privacy mechanisms, and in a UK government key-management protocol called Mikey-Sakke. Computing people's private keys from their email addresses or other identifiers may seem a neat hack, but it can be expensive when government departments are reorganised or renamed [115]. Most organisations and applications use ordinary public-key systems with certification of public keys, which I'll discuss next.

#### 5.7.4 Certification authorities

Now that we can do public-key encryption and digital signature, we need some mechanism to bind users to keys. The approach proposed by Diffie and Hellman when they invented digital signatures was to have a directory of the public keys of a system's authorised users, like a phone book. A more common solution, due to Loren Kohnfelder, is for a *certification authority* (CA) to sign the users' public encryption keys or their signature verification keys giving certificates that contain a user's name, one or more of their public keys, and attributes such as authorisations. The CA might be run by the local system administrator; but it is most commonly a third party service such as Verisign whose business is to sign public keys after doing some due diligence about whether they are controlled by the principals named in them.

A certificate might be described symbolically as

$$C_A = \text{Sig}_{K_S}(T_S, L, A, K_A, V_A) \quad (5.1)$$

where  $T_S$  is the certificate's starting date and time,  $L$  is the length of time for which it is valid,  $A$  is the user's name,  $K_A$  is her public encryption key, and  $V_A$  is her public signature verification key. In this way, only the administrator's public signature verification key needs to be communicated to all principals in a trustworthy manner.

Certification is hard, for a whole lot of reasons. Naming is hard, for starters; we discuss this in the following chapter on Distributed Systems. But often names aren't really what the protocol has to establish, as in the real world it's often about authorisation rather than authentication. Government systems are often about establishing not just a user's name or role but their security clearance level. In banking systems, it's about your balance, your available credit and your authority to spend it. In commercial systems, it's often about linking remote users to role-based access control. In user-facing systems, there is a tendency to dump on the customer as many of the compliance costs as possible [524]. There are many other things that can go wrong with certification at the level of systems engineering. At the level of politics, there are hundreds of certification authorities in a typical browser, they are all more or less equally trusted, and

many nation states can coerce at least one of them<sup>10</sup>. The revocation of bad certificates is usually flaky, if it works at all. There will be much more on these topics later. With these warnings, it's time to look at the most commonly used public key protocol, TLS.

### 5.7.5 TLS

I remarked above that a server could publish a public key  $KS$  and any web browser could then send a message  $M$  containing a credit card number to it encrypted using  $KS$ :  $\{M\}_{KS}$ . This is in essence what the TLS protocol (then known as SSL) was designed to do, at the start of e-commerce. It was developed by Paul Kocher and Taher ElGamal in 1995 to support encryption and authentication in both directions, so that both `http` requests and responses can be protected against both eavesdropping and manipulation. It's the protocol that's activated when you see the padlock on your browser toolbar.

Here is a simplified description of the basic version of the protocol in TLS v1:

1. the client sends the server a *client hello* message that contains its name  $C$ , a transaction serial number  $C\#$ , and a random nonce  $N_C$ ;
2. the server replies with a *server hello* message that contains its name  $S$ , a transaction serial number  $S\#$ , a random nonce  $N_S$ , and a certificate  $CS$  containing its public key  $KS$ . The client now checks the certificate  $CS$ , and if need be checks the key that signed it in another certificate, and so on back to a root certificate issued by a company such as Verisign and stored in the browser;
3. the client sends a *key exchange* message containing a *pre-master-secret* key,  $K_0$ , encrypted under the server public key  $KS$ . It also sends a *finished* message with a message authentication code (MAC) computed on all the messages to date. The key for this MAC is the *master-secret*,  $K_1$ . This key is computed by hashing the pre-master-secret key with the nonces sent by the client and server:  $K_1 = h(K_0, N_C, N_S)$ . From this point onward, all the traffic is encrypted; we'll write this as  $\{\dots\}_{KCS}$  in the client-server direction and  $\{\dots\}_{KSC}$  from the server to the client. These keys are generated in turn by hashing the nonces with  $K_1$ .
4. The server also sends a *finished* message with a MAC computed on all the messages to date. It then finally starts sending the data.

---

<sup>10</sup>The few that can't, try to cheat. In 2011 Iran hacked the CA Diginotar, and in 2019 Kazakhstan forced its citizens to add a local police certificate to their browser. In both cases the browser vendors pushed back fast and hard: Diginotar failed after it was blacklisted, while the Kazakh cert was blocked even if its citizens installed it manually. This of course raises issues of sovereignty.

$$\begin{aligned}
 C \rightarrow S : & \quad C, C\#, N_C \\
 S \rightarrow C : & \quad S, S\#, N_S, CS \\
 C \rightarrow S : & \quad \{K_0\}_{KS} \\
 C \rightarrow S : & \quad \{\text{finished}, MAC(K_1, \text{everythingtodate})\}_{KCS} \\
 S \rightarrow C : & \quad \{\text{finished}, MAC(K_1, \text{everythingtodate})\}_{KSC}, \{data\}_{KSC}
 \end{aligned}$$

Once a client and server have established a pre-master-secret, no more public-key operations are needed as further master secrets can be obtained by hashing it with new nonces.

#### 5.7.5.1 TLS uses

The full protocol is more complex than this, and has gone through a number of versions. It has supported a number of different ciphersuites, initially so that export versions of software could be limited to 40 bit keys – a condition of export licensing that was imposed for many years by the US government. This led to downgrade attacks where a middleperson could force the use of weak keys. Other ciphersuites support signed Diffie-Hellman key exchanges for transient keys, to provide forward and backward secrecy. TLS also has options for bidirectional authentication so that if the client also has a certificate, this can be checked by the server. In addition, the working keys  $K_{CS}$  and  $K_{SC}$  can contain separate subkeys for encryption and authentication, as is needed for legacy modes of operation such as CBC plus CBC MAC.

As well as being used to encrypt web traffic, TLS has also been available as an authentication option in Windows from Windows 2000 onwards; you can use it instead of Kerberos for authentication on corporate networks. I will describe its use in more detail in the chapter on network attack and defence.

#### 5.7.5.2 TLS security

Although early versions of SSL had a number of bugs [1973], SSL 3.0 and later appear to be sound; the version after SSL 3.0 was renamed TLS 1.0. It was formally verified by Larry Paulson in 1998, so we know that the idealised version of the protocol doesn't have any bugs [1502].

However, in the more than twenty years since then, there have been over a dozen serious attacks. Even in 1998, Daniel Bleichenbacher came up with the first of a number of attacks based on measuring the time it takes a server to decrypt, or the error messages it returns in response to carefully-crafted protocol responses [264]. TLS 1.1 appeared in 2006 with protection against exploits of CBC encryption and of padding errors; TLS 1.2 followed two years later, upgrading the hash function to SHA256 and supporting authenticated encryption; and meanwhile there were a number of patches dealing with various attacks that had emerged. Many of these patches were rather inelegant because of the difficulty of changing a widely-used protocol; it's difficult to change both the server and client ends at once, as any client still has to interact with millions of servers, many running outdated software, and most websites want to be able to deal with browsers of all ages and on all sorts of devices. This has been dealt with by the big service firms changing their browsers to reject obsolete

ciphersuites, and to add features like *strict transport security* (STS) whereby a website can instruct browsers to only interact with it using https in future (to prevent downgrade attacks). The browser firms have also mandated a number of other supporting measures, from shorter certificate lifetimes to certificate transparency, which we'll discuss in the chapter on network attack and defence.

#### 5.7.5.3 TLS 1.3

The most recent major upgrade to the core protocol, TLS 1.3, was approved by the IETF in January 2019 after two years of discussion. It has dropped backwards compatibility in order to end support for many old ciphers, and made it mandatory to establish end-to-end forward secrecy by means of a Diffie-Hellman key exchange at the start of each session. This has caused controversy with the banking industry, which routinely intercepts encrypted sessions in order to do monitoring for compliance purposes. This will no longer be possible, so banks will have to bear the legal discomfort of using obsolete encryption or the financial cost of redeveloping systems to monitor compliance at endpoints instead<sup>11</sup>.

### 5.7.6 Other public-key protocols

Dozens of other public-key protocols have found wide use, including the following, most of which we'll discuss in detail later. Here I'll briefly mention code signing, PGP and QUIC.

#### 5.7.6.1 Code signing

Code signing was introduced in the 1990s when people started downloading software rather than getting it on diskettes. It is now used very widely to assure the provenance of software. You might think that having a public signature-verification key in your software so that version  $N$  can verify an update to version  $N + 1$  would be a simple application of public-key cryptography but this is far from the case. Many platforms sign their operating-system code, including updates, to prevent persistent malware; the mechanisms often involve trusted hardware such as TPMs and I'll discuss them in the next chapter in section 6.2.5. Some platforms, such as the iPhone, will only run signed code; this not only assures the provenance of software but enables platform owners to monetise apps, as I will discuss in section 22.3.2; games consoles are similar. As some users go to great lengths to jailbreak their devices, such platforms typically have trustworthy hardware to store the verification keys. Where that isn't available, verification may be done using code that is obfuscated to make it harder for malware (or customers) to tamper with it; this is a constant arms race, which I discuss in section 24.3.3. As for the signing key, the developer may keep it in a hardware security module, which is expensive and breaks in subtle ways discussed in section 20.5; there may be a chain of trust going back a commercial CA, but then have to worry about legal coercion by government

---

<sup>11</sup>The COVID-19 pandemic has given some respite: Microsoft had been due to remove support for legacy versions of TLS in spring 2020 but has delayed this.

agencies, which I discuss in section 26.2.7; you might even implement your own CA for peace of mind. In short, code signing isn't quite as easy as it looks, particularly when the user is the enemy.

### 5.7.6.2 PGP/GPG

During the ‘Crypto Wars’ in the 1990s, cyber-activists fought governments for the right to encrypt email, while governments pushed for laws restricting encryption; I'll discuss the history and politics in section 26.2.7. The crypto activist Phil Zimmermann wrote an open-source encryption product *Pretty Good Privacy* (PGP) and circumvented U.S. export controls by publishing the source code in a paper book, which could be posted, scanned and compiled. Along with later compatible products such as GPG, it has become fairly widely used among geeks. For example, sysadmins, Computer Emergency Response Teams (CERTs) and malware researchers use it to share information about attacks and vulnerabilities. It has also been built into customised phones sold to criminal gangs to support messaging; I'll discuss this later in section 25.4.1.

PGP has a number of features but, in its most basic form, each user generates private/public keypairs manually and shares public keys with contacts. There are command-line options to sign a message with your signature key and/or encrypt it using the public key of each of the intended recipients. Manual key management avoids the need for a CA that can be cracked or coerced. Many things were learned from the deployment and use of PGP during the 1990s. As I described in section 3.2.1, Alma Whitten and Doug Tygar wrote the seminal paper on security usability by assessing whether motivated but cryptologically unsophisticated users could understand it well enough to drive the program safely. Only four of twelve subjects were able to correctly send encrypted email to the other subjects, and every subject made at least one significant error.

### 5.7.6.3 QUIC

QUIC is a new UDP-based protocol designed by Google and promoted as an alternative to TLS that allows quicker session establishment and thus cutting latency in the ad auctions that happen as pages load; sessions can persist as people move between access points. This is achieved by a cookie that holds the client's last IP address, encrypted by the server. It appeared in Chrome in 2013 and now has about 7% of Internet traffic; it's acquired a vigorous standardisation community. Google claims it reduces search latency 8% and YouTube buffer time 18%. Independent evaluation suggests that the benefit is mostly on the desktop rather than mobile [1007], and there's a privacy concern as the server can use an individual public key for each client, and use this for tracking. As a general principle, one should be wary of corporate attempts to replace open standards with proprietary ones, whether IBM's EBCDIC coding standard of the 1950s and SNA in the 1970s, or Microsoft's attempts to ‘embrace and extend’ both mail standards and security protocols since the 1990s, or Facebook's promotion of Internet access in Africa that kept users largely within its walled garden. I'll discuss the monopolistic tendencies of our industry at greater length in Chapter 8.

### 5.7.7 Special-purpose primitives

Researchers have invented a large number of public-key and signature primitives with special properties. Two that have so far appeared in real products are threshold cryptography and blind signatures.

*Threshold crypto* is a mechanism whereby a signing key, or a decryption key, can be split up among  $n$  principals so that any  $k$  out of  $n$  can sign a message (or decrypt). For  $k = n$  the construction is easy. With RSA, for example, you can split up the private key  $d$  as  $d = d_1 + d_2 + \dots + d_n$ . For  $k < n$  it's slightly more complex (but not much – you use the Lagrange interpolation formula) [554]. Threshold signatures were first used in systems where a number of servers process transactions independently and vote independently on the outcome; they have more recently been used to implement business rules on cryptocurrency wallets such as ‘a payment must be authorised by any two of the seven company directors’.

*Blind signatures* are a way of making a signature on a message without knowing what the message is. For example, if we are using RSA, I can take a random number  $R$ , form  $R^e M \pmod{n}$ , and give it to the signer who computes  $(R^e M)^d = R \cdot M^d \pmod{n}$ . When he gives this back to me, I can divide out  $R$  to get the signature  $M^d$ . Now you might ask why on earth someone would want to sign a document without knowing its contents, but there are some applications.

The first was in *digital cash*; you might want to be able to issue anonymous payment tokens to customers, and the earliest idea, due to David Chaum, was a way to sign ‘digital coins’ without knowing their serial numbers [411]. A bank might agree to honour for \$10 any string  $M$  with a unique serial number and a specified form of redundancy, bearing a signature that verified as correct using the public key  $(e, n)$ . The blind signature protocol ensures a customer can get a bank to sign a coin without the banker knowing its serial number, and it was used in prototype road toll systems. The effect is that the digital cash can be anonymous for the spender. The main problem with digital cash was to detect people who spend the same coin twice, and this was eventually fixed using blockchains or other ledger mechanisms, as I discuss in section 20.7. Digital cash failed to take off because neither banks nor governments really want payments to be anonymous: anti-money-laundering regulations since 9/11 restrict anonymous payment services to small amounts, while both banks and bitcoin miners like to collect transaction fees.

Anonymous digital credentials are now used in attestation: the TPM chip on your PC motherboard might prove something about the software running on your machine without identifying you. Unfortunately, this led to designs for attestation in SGX (and its AMD equivalent) which mean that a single compromised device breaks the whole ecosystem. Anonymous signatures are also found in prototype systems for conducting electronic elections, to which I will return in section 25.5.

### 5.7.8 How strong are asymmetric cryptographic primitives?

In order to provide the same level of protection as a symmetric block cipher, asymmetric cryptographic primitives generally require at least twice the block length. Elliptic curve systems appear to achieve this bound; a 256-bit elliptic scheme could be about as hard to break as a 128-bit block cipher with a 128-bit key; and the only public-key encryption schemes used in the NSA's Suite B of military algorithms are 384-bit elliptic curve systems. The traditional schemes, based on factoring and discrete log, now require 3072-bit keys to protect material at Top Secret, as there are shortcut attack algorithms such as the number field sieve. As a result, elliptic curve cryptosystems are faster.

When I wrote the first edition of this book in 2000, the number field sieve had been used to attack keys up to 512 bits, a task comparable in difficulty to keysearch on 56-bit DES keys; by the time I rewrote this chapter for the second edition in 2007, 64-bit symmetric keys had been brute-forced, and the 663-bit challenge number RSA-200 had been factored. By the third edition in 2019, bitcoin miners are finding 68-bit hash collisions every ten minutes, RSA-768 has been factored and Ed Snowden has as good as told us that the NSA can do discrete logs for a 1024-bit prime modulus.

There has been much research into *quantum computers* – devices that perform a large number of computations simultaneously using superposed quantum states. Peter Shor has shown that if a sufficiently large quantum computer could be built, then both factoring and discrete logarithm computations will become easy [1725]. So far only very small quantum devices have been built; although there are occasional claims of ‘quantum supremacy’ – of a quantum computer performing a task sufficiently faster than a conventional one to convince us that quantum superposition or entanglement is doing any real work – they seem to lead nowhere. I am sceptical (as are many physicists) about whether the technology will ever threaten real systems. I am even more sceptical about the value of quantum cryptography; it may be able to re-key a line encryption device that uses AES for bulk encryption, but we already know how to do that.

What’s more, I find the security proofs offered for entanglement-based quantum cryptography to be unconvincing [311]. Theoretical physics has been stuck since the early 1970s when Gerard ’t Hooft completed the Standard Model by proving the renormalisability of Yang-Mills. Since then, a series of ideas have come and gone, such as string theory [2032]. Quantum information theory is the latest enthusiasm. Its proponents talk up the mystery of the Bell tests, which are supposed to demonstrate that physics cannot be simultaneously local and causal. But alternative interpretations such as ’t Hooft’s cellular automaton model [916] and Grisha Volovik’s superfluid model [1967] suggest that the Bell tests merely demonstrate the existence of long-range order in the quantum vacuum, like the order parameter of a superfluid. And there are now classical mechanical experiments that demonstrate the quantum-mechanical properties relevant to the Bell tests [1557]. When I point out such loopholes in the proofs claimed for quantum cryptosystems, the proponents’ only answer appears to be personal abuse.

Personally I think it more likely that a major challenge to public-key cryp-

tography would come in the form of a better algorithm for computing discrete logarithms on elliptic curves. These curves have a lot of structure; they are studied intensively by some of the world's smartest pure mathematicians; better discrete-log algorithms for curves of small characteristic were discovered in 2013 [168]; and the NSA is apparently moving away from using elliptic-curve crypto.

If quantum computers ever work, we have other ‘post-quantum’ algorithms ready to go for which quantum computers give no obvious advantage. In 2020, NIST began the third round of public review of submissions for the Post-Quantum Cryptography Standardization Process. The 65 initial submissions have been cut to 15 through two rounds of review<sup>12</sup>. One or more algorithms will now be chosen and standardised, so ciphersuites using them could be dropped into protocols such as TLS as upgrades. Many protocols in use could even be redesigned to use variants on Kerberos. If elliptic logarithms become easy, we have these resources and can also fall back to discrete logs in prime fields, or to RSA. But if elliptic logs become easy, bitcoins will become trivial to forge, and the cryptocurrency ecosystem would probably collapse, putting an end to the immensely wasteful mining operations I describe in section 20.7. So mathematicians who care about the future of the planet might do worse than to think about the elliptic logarithm problem.

### 5.7.9 What else goes wrong

Very few attacks on systems nowadays involve cryptanalysis in the sense of a mathematical attack on the encryption algorithm or key. There have indeed been attacks on systems designed in the 20th century, mostly involving keys that were kept too short by export-control rules, clueless designs or both. I already discussed in section 4.3.1 how weak crypto has facilitated a wave of car theft, as all the devices used for remote key entry were defeated one after another in 2005–15. In later chapters, I give examples of how the crypto wars and their export control rules resulted in attacks on door locks (section 13.2.5), mobile phones (section 22.2.1) and copyright enforcement (section 24.2.5).

Most attacks nowadays exploit the implementation. In chapter 2, I mentioned the scandal of NIST standardising a complicated random number generator based on elliptic curves that turned out to contain an NSA backdoor; see section 2.2.1.5. Poor random number generators have led to many other failures: RSA keys with common factors [1140], predictable seeds for discrete logs [1676], etc. These vulnerabilities have continued; thanks to the Internet of Things, the proportion of RSA certs one can find out there on the Internet that share a common factor with other RSA keys has actually risen between 2012 and 2020; 1 in 172 IoT certs are trivially vulnerable [1046].

Many of the practical attacks on cryptographic implementations that have forced significant changes over the past 20 years have exploited side channels such as timing and power analysis; I devote Chapter 19 to these.

---

<sup>12</sup>One of them, the McEliece cryptosystem, has been around since 1978; we’ve had digital signatures based on hash functions for about as long, and some of us used them in the 1990s to avoid paying patent royalties on RSA.

In Chapter 20, I'll discuss a number of systems that use public-key mechanisms in intricate ways to get interesting emergent properties, including the Signal messaging protocol, the TOR anonymity system, and cryptocurrencies. I'll also look at the crypto aspects of SGX enclaves. These also have interesting failure modes, some but not all of them relating to side channels.

In Chapter 21, I'll discuss protocols used in network infrastructure such as DKIM, DNSSec versus DNS over HTTP, and SSH.

## 5.8 Summary

Many ciphers fail because they're used badly, so the security engineer needs a clear idea of what different types of cipher do. This can be tackled at different levels; one is at the level of crypto theory, where we can talk about the random oracle model, the concrete model and the semantic security model, and hopefully avoid using weak modes of operation and other constructions. The next level is that of the design of individual ciphers, such as AES, or the number-theoretic mechanisms that underlie public-key cryptosystems and digital signature mechanisms. These also have their own specialised fields of mathematics, namely block cipher cryptanalysis and computational number theory. The next level involves implementation badness, which is much more intractable and messy. This involves dealing with timing, error handling, power consumption and all sorts of other grubby details, and is where modern cryptosystems tend to break in practice.

Peering under the hood of real systems, we've discussed how block ciphers for symmetric key applications can be constructed by the careful combination of substitutions and permutations; for asymmetric applications such as public key encryption and digital signature one uses number theory. In both cases, there is quite a large body of mathematics. Other kinds of ciphers – stream ciphers and hash functions – can be constructed from block ciphers by using them in suitable modes of operation. These have different error propagation, pattern concealment and integrity protection properties. A lot of systems fail because popular crypto libraries encourage programmers to use inappropriate modes of operation by exposing unsafe defaults. Never use ECB mode unless you really understand what you're doing.

There are many other things that can go wrong, from side channel attacks to poor random number generators. In particular, it is surprisingly hard to build systems that are robust even when components fail (or are encouraged to) and where the cryptographic mechanisms are well integrated with other measures such as access control and physical security. I'll return to this repeatedly in later chapters.

The moral is: Don't roll your own! Don't design your own protocols, or your own ciphers; and don't write your own crypto code unless you absolutely have to. If you do, then you not only need to read this book (and then read it again, carefully); you need to read up the relevant specialist material, speak to experts, and have capable motivated people try to break it. At the very least, you need to get your work peer-reviewed. Designing crypto is a bit like juggling

chainsaws; it's just too easy to make fatal errors.

## Research Problems

There are many active threads in cryptography research. Many of them are where crypto meets a particular branch of mathematics (number theory, algebraic geometry, complexity theory, combinatorics, graph theory, and information theory). The empirical end of the business is concerned with designing primitives for encryption, signature and composite operations, and which perform reasonably well on available platforms. The two meet in the study of subjects ranging from cryptanalysis, to the search for primitives that combine provable security properties with decent performance.

The best way to get a flavor of what's going on at the theoretical end of things is to read the last few years' proceedings of research conferences such as Crypto, Eurocrypt and Asiacrypt; work on cipher design appears at Fast Software Encryption; attacks on implementations often appear at CHES; while attacks on how crypto gets used in systems can be found in the top systems security conferences such as IEEE Security and Privacy, CCS and Usenix.

## Further Reading

The classic papers by Whit Diffie and Martin Hellman [556] and by Ron Rivest, Adi Shamir and Len Adleman [1607] are the closest to required reading in this subject. Bruce Schneier's *Applied Cryptography* [1667] covers a lot of ground at a level a non-mathematician can understand, and got crypto code out there in the 1990s despite US export control laws, but is now slightly dated. Alfred Menezes, Paul van Oorschot and Scott Vanstone's *Handbook of Applied Cryptography* [1289] is one reference book on the mathematical detail. Katz and Lindell is the book we get our students to read for the math. It gives an introduction to the standard crypto theory plus the number theory you need for public-key crypto (including elliptic curves and index calculus) but is also dated: they don't mention GCM, for example [1022].

There are many more specialist books. The bible on differential cryptanalysis is by its inventors Eli Biham and Adi Shamir [245], while a good short tutorial on linear and differential cryptanalysis was written by Howard Heys [895]. Doug Stinson's textbook has another detailed explanation of linear cryptanalysis [1829]; and the modern theory of block ciphers can be traced through the papers in the *Fast Software Encryption* conference series. The original book on modes of operation is by Carl Meyer and Steve Matyas [1301]. Neal Koblitz has a good basic introduction to the mathematics behind public key cryptography [1060]; and the number field sieve is described by Arjen and Henrik Lenstra [1141]. For the practical attacks on TLS over the past twenty years, see the survey paper by Christopher Meyer and Joerg Schwenk [1302] as well as the chapter on Side Channels later in this book.

If you want to work through the mathematical detail of theoretical cryptology, there's an recent graduate textbook by Dan Boneh and Victor Shoup [287].

## 5.8. SUMMARY

---

A less thorough but more readable introduction to randomness and algorithms is in [835]. Research at the theoretical end of cryptology is found at the FOCS, STOC, Crypto, Eurocrypt and Asiacrypt conferences.

The history of cryptology is fascinating, and so many old problems keep on recurring. The standard work is Kahn [1001]; there are also compilations of historical articles from *Cryptologia* [531, 529, 530] as well as several books on the history of cryptology in World War II by Kahn, Marks, Welchman and others [438, 1002, 1224, 2007]. The NSA Museum at Fort George Meade, Md., is also worth a visit, but perhaps the best is the museum at Bletchley Park in England.

Finally, no chapter that introduces public key encryption would be complete without a mention that, under the name of ‘non-secret encryption,’ it was first discovered by James Ellis in about 1969. However, as Ellis worked for GCHQ, his work remained classified. The RSA algorithm was then invented by Clifford Cocks, and also kept secret. This story is told in [625]. One effect of the secrecy was that their work was not used: although it was motivated by the expense of Army key distribution, Britain’s Ministry of Defence did not start building electronic key distribution systems for its main networks until 1992. And the classified community did not pre-invent digital signatures; they remain the achievement of Whit Diffie and Martin Hellman.

# Chapter 6

# Access Control

Microsoft could have incorporated effective security measures as standard, but good sense prevailed. Security systems have a nasty habit of backfiring and there is no doubt they would cause enormous problems.  
– RICK MAYBURY

Optimisation consists of taking something that works and replacing it with something that almost works but is cheaper.  
– ROGER NEEDHAM

## 6.1 Introduction

I first learned to program on an IBM mainframe whose input was punched cards and whose output was a printer. You queued up with a deck of cards, ran the job, and went away with printout. All security was physical. Then along came machines that would run more than one program at once, and the *protection problem* of preventing one program from interfering with another. You don't want a virus to steal the passwords from your browser, or patch a banking application so as to steal your money. And many reliability problems stem from applications misunderstanding each other, or fighting with each other. But it's tricky to separate applications when the customer wants them to share data. It would make phishing much harder if your email client and browser ran on separate machines, so you were unable to just click on URLs in emails, but that would make life too hard.

From the 1970s, access control became the centre of gravity of computer security. It's where security engineering meets computer science. Its function is to control which principals (persons, processes, machines, . . .) have access to which resources in the system – which files they can read, which programs they can execute, how they share data with other principals, and so on. It's become horrendously complex. If you start out by leafing through the 7000-plus pages of Arm's architecture reference manual or the equally complex arrangements for

## **6.1. INTRODUCTION**

---

Windows at the O/S level, your first reaction might be ‘I wish I’d studied music instead!’ In this chapter I try to help you make sense of it all.

Access control works at a number of different levels, including at least:

1. Access controls at the application level may express a very rich, domain-specific security policy. The call centre staff in a bank are typically not allowed to see your account details until you have answered a couple of security questions; this not only stops outsiders impersonating you, but also stops the bank staff looking up the accounts of celebrities, or their neighbours. Some transactions might also require approval from a supervisor. And that’s nothing compared with the complexity of the access controls on a modern social networking site, which will have a thicket of rules about who can see, copy, and search what data from whom, and privacy options that users can set to modify these rules.
2. The applications may be written on top of middleware, such as a web browser, a bank’s bookkeeping system or a social network’s database management system. These enforce a number of protection properties. For example, bookkeeping systems ensure that a transaction that debits one account must credit another, with the debits and credits balancing so that money cannot be created or destroyed; they must also allow the system’s state to be reconstructed later.
3. As the operating system constructs resources such as files and communications ports from lower level components, it has to provide ways to control access to them. Your Android phone treats apps written by different companies as different users and protects their data from each other. The same happens when a shared server separates the VMs, containers or other resources belonging to different users.
4. Finally, the operating system relies on hardware protection provided by the processor and its associated memory-management hardware, which control which memory addresses a given process or thread can access.

As we work up from the hardware through the operating system and middleware to the application layer, the controls become progressively more complex and less reliable. And we find the same access-control functions being implemented at multiple layers. For example, the separation between different phone apps that is provided by Android is mirrored in your browser which separates web page material according to the domain name it came from (though this separation is often less thorough). And the access controls built at the application layer or the middleware layer may largely duplicate access controls in the underlying operating system or hardware. It can get very messy, and to make sense of it we need to understand the underlying principles, the common architectures, and how they have evolved.

I will start off by discussing operating-system protection mechanisms that support the isolation of multiple processes. These came first historically – being invented along with the first time-sharing systems in the 1960s – and they remain the foundation on which many higher-layer mechanisms are built, as well as inspiring similar mechanisms at higher layers. They are often described as

*discretionary access control* (DAC) mechanisms, which leave protection to the machine operator, or *mandatory access control* (MAC) mechanisms which are typically under the control of the vendor and protect the operating system itself from being modified by malware. I'll give an introduction to software attacks and techniques for defending against them – MAC, ASLR, sandboxing, virtualisation and what can be done with hardware. Modern hardware not only provides CPU support for virtualisation and capabilities, but also hardware support such as TPM chips for trusted boot to stop malware being persistent. These help us tackle the toxic legacy of the old single-user PC operating systems such as DOS and Win95/98 which let any process modify any data, and constrain the many applications that won't run unless you trick them into thinking that they are running with administrator privileges.

## 6.2 Operating system access controls

The access controls provided with an operating system typically authenticate principals using a mechanism such as passwords or fingerprints in the case of phones, or passwords or security protocols in the case of servers, then authorise access to files, communications ports and other system resources.

Access controls can often be modeled as a matrix of access permissions, with columns for files and rows for users. We'll write r for permission to read, w for permission to write, x for permission to execute a program, and – for no access at all, as shown in Figure 6.1.

	Operating System	Accounts Program	Accounting Data	Audit Trail
Sam	rwx	rwx	rw	r
Alice	x	x	rw	–
Bob	rx	r	r	r

Fig. 6.1 – naive access control matrix

In this simplified example, Sam is the system administrator and has universal access (except to the audit trail, which even he should only be able to read). Alice, the manager, needs to execute the operating system and application, but only through the approved interfaces – she mustn't have the ability to tamper with them. She also needs to read and write the data. Bob, the auditor, can read everything.

This is often enough, but in the specific case of a bookkeeping system it's not quite what we need. We want to ensure that transactions are well-formed – that each debit is balanced by credits somewhere else – so we don't want Alice to have uninhibited write access to the account file. We would also rather that Sam didn't have this access. So we would prefer that write access to the accounting data file be possible only via the accounting program. The access permissions might now look like in Figure 6.2:

## 6.2. OPERATING SYSTEM ACCESS CONTROLS

---

User	Operating System	Accounts Program	Accounting Data	Audit Trail
Sam	rwx	rwx	r	r
Alice	rx	x	-	-
Accounts program	rx	rx	rw	w
Bob	rx	r	r	r

Fig. 6.2 – access control matrix for bookkeeping

Another way of expressing a policy of this type would be with *access triples* of (*user, program, file*). In the general case, our concern isn't with a program so much as a *protection domain* which is a set of processes or threads that share access to the same resources.

Access control matrices (whether in two or three dimensions) can be used to implement protection mechanisms as well as just model them. But they don't scale well: a bank with 50,000 staff and 300 applications would have a matrix of 15,000,000 entries, which might not only impose a performance overhead but also be vulnerable to administrators' mistakes. We will need a better way of storing and managing this information, and the two main options are to compress the users and to compress the rights. With the first, we can use groups or roles to manage large sets of users simultaneously, while with the second we may store the access control matrix either by columns (access control lists) or rows (capabilities, also known as 'tickets' to protocol engineers and 'permissions' on mobile phones) [1639, 2020].

### 6.2.1 Groups and Roles

When we look at large organisations, we usually find that most staff fit into one of a small number of categories. A bank might have 40 or 50: teller, call centre operator, loan officer and so on. Only a few dozen people (security manager, chief foreign exchange dealer, ...) will need personally customised access rights.

So we need to design a set of groups, or functional roles, to which staff can be assigned. Some vendors (such as Microsoft) use the words *group* and *role* almost interchangeably, but a more careful definition is that a group is a list of principals, while a role is a fixed set of access permissions that one or more principals may assume for a period of time. The classic example of a role is the officer of the watch on a ship. There is exactly one watchkeeper at any one time, and there is a formal procedure whereby one officer relieves another when the watch changes. In most government and business applications, it's the role that matters rather than the individual.

Groups and roles can be combined. *The officers of the watch of all ships currently at sea* is a group of roles. In banking, the manager of the Cambridge branch might have their privileges expressed by membership of the group *manager* and assumption of the role *acting manager of Cambridge branch*. The group *manager* might express a rank in the organisation (and perhaps even a salary band) while the role *acting manager* might include an assistant accountant standing in while the manager, deputy manager, and branch accountant are all off sick.

Whether we need to be careful about this distinction is a matter for the application. In a warship, even an ordinary seaman may stand watch if everyone more senior has been killed. In a bank, we might have a policy that “transfers over \$10m must be approved by two staff, one with rank at least manager and one with rank at least assistant accountant”. If the branch manager is sick, then the assistant accountant acting as manager might have to get the regional head office to provide the second signature on a large transfer.

### 6.2.2 Access control lists

The traditional way to simplify the management of access rights is to store the access control matrix a column at a time, along with the resource to which the column refers. This is called an *access control list* or ACL (pronounced ‘ackle’). In the first of our above examples, the ACL for file 3 (the account file) might look as shown here in Figure 6.3.

User	Accounting Data
Sam	rw
Alice	rw
Bob	r

Fig. 6.3 – access control list (ACL)

ACLs have a number of advantages and disadvantages as a means of managing security state. They are a natural choice in environments where users manage their own file security, and became widespread in Unix systems from the 1970s. They are the basic access control mechanism in Unix-based systems such as Linux and Apple’s macOS, as well as in derivatives such as Android and iOS. The access controls in Windows were also based on ACLs, but have become more complex over time. Where access control policy is set centrally, ACLs are suited to environments where protection is data-oriented; they are less suited where the user population is large and constantly changing, or where users want to be able to delegate their authority to run a particular program to another user for some set period of time. ACLs are simple to implement, but are not efficient for security checking at runtime, as the typical operating system knows which user is running a particular program, rather than what files it has been authorized to access since it was invoked. The operating system must either check the ACL at each file access, or keep track of the active access rights in some other way.

Finally, distributing the access rules into ACLs makes it tedious to find all the files to which a user has access. Verifying that no files have been left world-readable or even world-writable could involve checking ACLs on millions of user files; this is a real issue for large complex firms. Although you can write a script to check whether any file on a server has ACLs that breach a security policy, you can be tripped up by technology changes; the move to containers has led to many corporate data exposures as admins forgot to check the containers’ ACLs too. (The containers themselves are often dreadful as it’s a new technology being sold by dozens of clueless startups.) And revoking the access of an employee

who has just been fired will usually have to be done by cancelling their password or authentication token.

Let's look at an important example of ACLs – their implementation in Unix (plus its derivatives Android, MacOS and iOS).

### 6.2.3 Unix operating system security

In traditional Unix systems, files are not allowed to have arbitrary access control lists, but simply **rwx** attributes that allow the file to be read, written and executed. The access control list as normally displayed has a flag to show whether the file is a directory, then flags r, w and x for owner, group and world respectively; it then has the owner's name and the group name. A directory with all flags set would have the ACL:

```
drwxrwxrwx Alice Accounts
```

In our first example in Figure 6.1, the ACL of file 3 would be:

```
-rw-r----- Alice Accounts
```

This records that the file is simply a file rather than a directory; that the file owner can read and write it; that group members (including Bob) can read it but not write it; that non-group members have no access at all; that the file owner is Alice; and that the group is Accounts.

The program that gets control when the machine is booted (the operating system kernel) runs as the supervisor, and has unrestricted access to the whole machine. All other programs run as users and have their access mediated by the supervisor. Access decisions are made on the basis of the userid associated with the program. However if this is zero (**root**), then the access control decision is 'yes'. So root can do what it likes – access any file, become any user, or whatever. What's more, there are certain things that only root can do, such as starting certain communication processes. The root userid is typically made available to the system administrator in systems with discretionary access control.

This means that the system administrator can do anything, so we have difficulty implementing an audit trail as a file that they cannot modify. In our example, Sam could tinker with the accounts, and have difficulty defending himself if he were falsely accused of tinkering; what's more, a hacker who managed to become the administrator could remove all evidence of his intrusion. The traditional, and still the most common, way to protect logs against root compromise is to keep them separate. In the old days that meant sending the system log to a printer in a locked room; nowadays, it means sending it to another machine, or even to a third-party service. Increasingly, it may also involve mandatory access control, as we discuss later.

Second, ACLs only contain the names of users, not of programs; so there is no straightforward way to implement access triples of (user, program, file). Instead, Unix provides an indirect method: the *set-user-id* (**suid**) file attribute. The owner of a program can mark the file representing that program as **suid**, which enables it to run with the privilege of its owner rather than the privilege of the user who has invoked it. So in order to achieve the functionality needed by our second example above, we could create a user '**account-package**' to own

## 6.2. OPERATING SYSTEM ACCESS CONTROLS

---

file 2 (the accounts package), make the file **suid** and place it in a directory to which Alice has access. This special user can then be given the access that the accounts program needs.

But when you take an access control problem that has three dimensions – (user, program, data) – and implement it using two-dimensional mechanisms, the outcome is much less intuitive than triples and people are liable to make mistakes. Programmers are often lazy or facing tight deadlines; so they just make the application **suid root**, so it can do anything. This practice leads to some shocking security holes. The responsibility for making access control decisions is moved from the operating system environment to the application program, and most programmers are insufficiently experienced to check everything they should. (It's hard to know what to check, as the person invoking a **suid root** program controls its environment and could manipulate this in unexpected ways.)

Third, ACLs are not very good at expressing mutable state. Suppose we want a transaction to be authorised by a manager and an accountant before it's acted on; we can either do this at the application level (say, by having queues of transactions awaiting a second signature) or by doing something fancy with **suid**. Managing stateful access rules is difficult; they can complicate the revocation of users who have just been fired, as it can be hard to track down the files they've opened, and stuff can get stuck.

Fourth, the Unix ACL only names one user. If a resource will be used by more than one of them, and you want to do access control at the OS level, you have a couple of options. With older systems you had to use groups; newer systems implement the Posix system of extended ACLs, which may contain any number of named user and named group entities. In theory, the ACL and **suid** mechanisms can often be used to achieve the desired effect. In practice, programmers are often in too much of a hurry to figure out how to do this, and security interfaces are usually way too fiddly to use. So people design their code to require much more privilege than it strictly ought to have, as that seems to be the only way to get the job done.

### 6.2.4 Capabilities

The next way to manage the access control matrix is to store it by rows. These are called *capabilities*, and in our example in Figure 6.1 above, Bob's capabilities would be as in Figure 6.4 here:

User	Operating System	Accounts Program	Accounting Data	Audit Trail
Bob	rx	r	r	r

Fig. 6.4 – a capability

The strengths and weaknesses of capabilities are roughly the opposite of ACLs. Runtime security checking is more efficient, and we can delegate a right without much difficulty: Bob could create a certificate saying 'Here is my capability and I hereby delegate to David the right to read file 4 from 9am to 1pm,

signed Bob'. On the other hand, changing a file's status becomes more tricky as it can be hard to find out which users have access. This can be tiresome when we have to investigate an incident or prepare evidence. In fact, scalable systems end up using de-facto capabilities internally, as instant system-wide revocation is just too expensive; in Unix, file descriptors are really capabilities, and continue to grant access for some time even after ACL permissions or even file owners change. In a distributed Unix, access may persist for the lifetime of Kerberos tickets.

Could we do away with ACLs entirely then? People built experimental machines in the 1970s that used capabilities throughout [2020]; the first commercial product was the Plessey System 250, a telephone-switch controller [1575]. The IBM AS/400 series systems brought capability-based protection to the mainstream computing market in 1988, and enjoyed some commercial success. The public key certificates used in cryptography are in effect capabilities, and became mainstream from the mid-1990s. Capabilities have started to supplement ACLs in operating systems, including more recent versions of Windows, FreeBSD and iOS, as I will describe later.

In some applications, they can be the natural way to express security policy. For example, a hospital may have access rules like 'a nurse shall have access to all the patients who are on his or her ward, or who have been there in the last 90 days'. In early systems based on traditional ACLs, each access control decision required a reference to administrative systems to find out which nurses and which patients were on which ward, when – but this made both the HR system and the patient administration system safety-critical, which hammered reliability. Matters were fixed by giving nurses ID cards with certificates that entitle them to access the files associated with a number of wards or hospital departments [535, 536]. If you can make the trust relationships in systems mirror the trust relationships in that part of the world you're trying to automate, you should. Working with the grain can bring advantages at all levels in the stack, making things more usable, supporting safer defaults, cutting errors, reducing engineering effort and saving money too.

### 6.2.5 DAC and MAC

In the old days, anyone with physical access to a computer controlled all of it: you could load whatever software you liked, inspect everything in memory or on disk and change anything you wanted to. This is the model behind *discretionary access control* (DAC): you start your computer in supervisor mode and then, as the administrator, you can make less-privileged accounts available for less-trusted tasks – such as running apps written by companies you don't entirely trust, or giving remote logon access to others. But this can make things hard to manage at scale, and in the 1970s the US military started a huge computer-security research program whose goal was to protect classified information: to ensure that a file marked 'Top Secret' would never be made available to a user with only a 'Secret' clearance, regardless of the actions of any ordinary user or even of the supervisor. In such a *multilevel secure* (MLS) system, the sysadmin is no longer the boss: ultimate control rests with a remote government authority that sets security policy. The mechanisms started to be described as *mandatory*

## 6.2. OPERATING SYSTEM ACCESS CONTROLS

---

*access control* (MAC). The supervisor, or root access if you will, is under remote control. This drove development of technology for mandatory access control – a fascinating story, which I tell in Part 2 of the book.

From the 1980s, safety engineers also worked on the idea of *safety integrity levels*; roughly, that a more dependable system must not rely on a less dependable one. They started to realise they needed something similar to multilevel security, but for safety. Military system people also came to realise that the tamper-resistance of the protection mechanisms themselves was of central importance. In the 1990s, as computers and networks became fast enough to handle audio and video, the creative industries lobbied for *digital rights management* (DRM) in the hope of preventing people undermining their business models by sharing music and video. This is also a form of mandatory access control – stopping a subscriber sharing a song with a non-subscriber is in many ways like stopping a Top Secret user sharing an intelligence report with a Secret user.

In the early 2000s, these ideas came together as a number of operating-system vendors started to incorporate ideas and mechanisms from the MAC research programme into their products. The catalyst was an initiative by Microsoft and Intel to introduce cryptography into the PC platform to support DRM. Intel believed the business market for PCs was saturated, so growth would come from home sales where, they believed, DRM would be a requirement. Microsoft started with DRM and then realised that offering rights management for documents too might be a way of locking customers tightly into Windows and Office. They set up an industry alliance, now called the Trusted Computing Group, to introduce cryptography and MAC mechanisms into the PC platform. To do this, the operating system had to be made tamper-resistant, and this is achieved by means of a separate processor, the Trusted Platform Module (TPM), basically a smartcard chip mounted on the PC motherboard to support trusted boot and hard disk encryption. The TPM monitors the boot process, and at each stage a hash of everything loaded so far is needed to retrieve the key needed to decrypt the next stage. The real supervisor on the system is now no longer you, the machine owner – it's the operating-system vendor.

MAC, based on TPMs and trusted boot, was used in Windows 6 (Vista) from 2006 as a defence against persistent malware<sup>1</sup>. The TPM standards and architecture were adapted by other operating-system vendors and device OEMs, and there is now even a project for an open-source TPM chip, OpenTitan, based on Google's product. However the main purpose of such a design, whether its own design is open or closed, is to lock a hardware device to using specific software.

---

<sup>1</sup>Microsoft had had more ambitious plans; its project Palladium would have provided a new, more trusted world for rights-management apps, alongside the normal one for legacy software. They launched Information Rights Management – DRM for documents – in 2003 but corporates didn't buy it, seeing it as a lock-in play. A two-world implementation turned out to be too complex for Vista and after two separate development efforts it was abandoned; but the vision persisted from 2004 in Arm's TrustZone, which I discuss below.

### 6.2.6 Apple's macOS

Apple's macOS operating system (formerly called OS/X or Mac OS X) is based on the FreeBSD version of Unix running on top of the Mach kernel. The BSD layer provides memory protection; applications cannot access system memory (or each others') unless running with advanced permissions. This means, for example, that you can kill a wedged application using the 'Force Quit' command without having to reboot the system. On top of this Unix core are a number of graphics components, including OpenGL, Quartz, Quicktime and Carbon, while at the surface the Aqua user interface provides an elegant and coherent view to the user.

At the file system level, macOS is almost a standard Unix. The default installation has the root account disabled, but users who may administer the system are in a group 'wheel' that allows them to su to root. If you are such a user, you can install programs (you are asked for the root password when you do so). Since version 10.5 (Leopard), it has been based on TrustedBSD, a variant of BSD that incorporates mandatory access control mechanisms, which are used to protect core system components against tampering by malware.

### 6.2.7 iOS

Since 2008, Apple has led the smartphone revolution with the iPhone, which (along with other devices like the iPad) uses the iOS operating system – which is now (in 2020) the second-most popular. iOS is based on Unix; Apple took the Mach kernel from CMU and fused it with the FreeBSD version of Unix, making a number of changes for performance and robustness. For example, in vanilla Unix a filename can have multiple pathnames that lead to an inode representing a file object, which is what the operating system sees; in iOS, this has been simplified so that files have unique pathnames, which in turn are the subject of the file-level access controls. Again, there is a MAC component, where mechanisms from Domain and Type Enforcement (DTE) are used to tamper-proof core system components (we'll discuss DTE in more detail in chapter 9). Apple introduced this because they were worried that apps would brick the iPhone, leading to warranty claims.

Apps also have *permissions*, which are capabilities; they request a capability to access device services such as the mobile network, the phone, SMSes, the camera, and the first time the app attempts to use such a service. This is granted if the user consents<sup>2</sup>. The many device services open up possible side-channel attacks; for example, an app that's denied access to the keyboard could deduce keypresses using the accelerometer and gyro. We'll discuss side channels in Part 2, in the chapter on that subject.

The Apple ecosystem is closed in the sense that an iPhone will only run apps

---

<sup>2</sup>The trust-on-first-use model goes back to the 1990s with the Java standard J2ME, popularised by Symbian, and the Resurrecting Duckling model from about the same time. J2ME also supported trust-on-install and more besides. When Apple and Android came along, they initially made different choices. In each case, having an app store was a key innovation; Nokia failed to realise that this was important to get a two-sided market going. The app store does some of the access control by deciding what apps can run. This is hard power in Apple's case, and soft power in Android's; we'll discuss this in the chapter on phones.

## 6.2. OPERATING SYSTEM ACCESS CONTROLS

---

that Apple has signed<sup>3</sup>. This enables the company to extract a share of app revenue, and also to screen apps for malware or other undesirable behaviour, such as the exploitation of side channels to defeat access controls.

The iPhone 5S introduced a fingerprint biometric and payments, adding a *secure enclave* (SE) to the A7 processor to give them separate protection. Apple decided to trust neither iOS nor TrustZone with such sensitive data, since vulnerabilities give transient access until they're patched. Its engineers also worried that an unpatchable exploit might be found in the ROM (this eventually happened, with Checkm8). While iOS has access to the system partition, the user's personal data are encrypted, with the keys managed by the SE. Key management is bootstrapped by a unique 256-bit AES key burned into fusible links on the system-on-chip. When the device is powered up, the user has ten tries to enter a passcode; only then are file keys derived from the master key and made available<sup>4</sup>. When the device is locked, some keys are still usable so that iOS can work out who sent an incoming message and notify you; the price of this convenience is that forensic equipment can get some access to user data. The SE also manages upgrades and prevents rollbacks. Such public information as there is can be found in the iOS Security white paper [128].

The security of mobile devices is a rather complex issue, involving not just access controls and tamper resistance, but the whole ecosystem – from the provision of SIM cards through the operation of app stores to the culture of how people use devices, how businesses try to manipulate them and how government agencies spy on them. I will discuss this in detail in the chapter on phones in Part 2.

### 6.2.8 Android

Android is the world's most widely used operating system, with 2.5 billion active Android devices in May 2019, according to Google's figures. Android is based on Linux; apps from different vendors run under different userids. The Linux mechanisms control access at the file level, preventing one app from reading another's data and exhausting shared resources such as memory and CPU. As in iOS, apps have *permissions*, which are in effect capabilities: they grant access to device services such as SMSes, the camera and the address book.

Apps come in signed packages, as .apk files, and while iOS apps are signed by Apple, the verification keys for Android come in self-signed certificates and function as the developer's name. This supports integrity of updates while maintaining an open ecosystem. Each package contains a manifest that demands a set of permissions, and users have to approve the 'dangerous' ones – roughly, those that can spend money or compromise personal data. In early versions of Android, the user would have to approve the lot on installation or not run the app. But experience showed that most users would just click on anything to get through the installation process, and you found even flashlight apps demanding access to your address book, as they could sell it for money. So Android 6 moved

---

<sup>3</sup>There are a few exceptions: corporates can get signing keys for internal apps, but these can be blacklisted if abused.

<sup>4</sup>I'll discuss fusible links in the chapter on tamper resistance, and iPhone PIN retry defeats in the chapter on surveillance and privacy.

to the Apple model of trust on first use; apps compiled for earlier versions still demand capabilities on installation.

Since Android 5, SELinux has been used to harden the operating system with mandatory access controls, so as not only to protect core system functions from attack but also to separate processes strongly and log violations. SELinux was developed by the NSA to support MAC in government systems; we'll discuss it further in chapter 9. The philosophy is actions require the consent of three parties: the user, the developer and the platform.

As with iOS (and indeed Windows), the security of Android is a matter of the whole ecosystem, not just of the access control mechanisms. The new phone ecosystem is sufficiently different from the old PC ecosystem, but inherits enough of the characteristics of the old wireline phone system, that it merits a separate discussion in the chapter on Phones in Part II. We'll consider other aspects in the chapters on Side Channels and Surveillance.

### 6.2.9 Windows

The current version of Windows (Windows 10) appears to be the third-most popular operating system, having achieved a billion monthly active devices in March 2020 (until 2016, Windows was the leader). Windows has a scarily complex access control system, and a quick canter through its evolution may make it easier to understand what's going on.

Early versions of Windows had no access control. A break came with Windows 4 (NT), which was very much like Unix, and was inspired by it, but with some extensions. First, rather than just *read*, *write* and *execute* there were separate attributes for *take ownership*, *change permissions* and *delete*, to support more flexible delegation. These attributes apply to groups as well as users, and group permissions allow you to achieve much the same effect as **suid** programs in Unix. Attributes are not simply on or off, as in Unix, but have multiple values: you can set *AccessDenied*, *AccessAllowed* or *SystemAudit*. These are parsed in that order: if an *AccessDenied* is encountered in an ACL for the relevant user or group, then no access is permitted regardless of any conflicting *AccessAllowed* flags. The richer syntax lets you arrange matters so that everyday configuration tasks, such as installing printers, don't have to require full administrator privileges.

Second, users and resources can be partitioned into domains with distinct administrators, and trust can be inherited between domains in one direction or both. In a typical large company, you might put all the users into a personnel domain administered by HR, while assets such as servers and printers may be in resource domains under departmental control; individual workstations may even be administered by their users. Things can be arranged so that the departmental resource domains trust the user domain, but not vice versa – so a hacked or careless departmental administrator can't do too much external damage. The individual workstations would in turn trust the department (but not vice versa) so that users can perform tasks that require local privilege (such as installing software packages). Limiting the damage a hacked administrator can do still needs careful organisation. The data structure used to manage all this, and hide

## 6.2. OPERATING SYSTEM ACCESS CONTROLS

the ACL details from the user interface, is called the *Registry*. Its core used to be the *Active Directory* which managed remote authentication – using either a Kerberos variant or TLS, encapsulated behind the *Security Support Provider Interface* (SSPI) which enables administrators to plug in other authentication services. Active Directory is essentially a database that organises users, groups, machines, and organisational units within a domain in a hierarchical namespace. It lurked behind Exchange, but is now being phased out as Microsoft becomes a cloud-based company and moves its users to Office365.

Windows has added capabilities in two ways which can override or complement ACLs. First, users or groups can be either allowed or denied access by means of profiles. Security policy is set by groups rather than for the system as a whole; group policy overrides individual profiles, and can be associated with sites, domains or organisational units, so it can start to tackle complex problems. Policies can be created using standard tools or custom coded.

The second way in which capabilities insinuate their way into Windows is that in many applications, people use TLS for authentication, and TLS certificates provide another, capability-oriented, layer of access control outside the purview of the Active Directory.

I already mentioned that Windows Vista introduced trusted boot to make the operating system itself tamper-resistant, in the sense that it always boots into a known state, limiting the persistence of malware. It added three further protection mechanisms to get away from the previous default of all software running as root. First, the kernel was closed off to developers; second, the graphics subsystem and most drivers were removed from the kernel; and third, *User Account Control* (UAC) replaced the default administrator privilege with user defaults instead. Previously, so many routine tasks needed administrative privilege that many enterprises made all their users administrators, which made it difficult to contain malware; and many developers wrote their software on the assumption that it would have access to everything (for a hall of shame, see [?]). According to Microsoft engineers, this was a major reason for Windows' lack of robustness: applications monkey with system resources in incompatible ways. So they added an Application Information Service that launches applications which require elevated privilege and uses virtualisation to contain them: if they modify the registry, for example, they don't modify the 'real' registry but simply the version of it that they can see.

Since Vista, the desktop acts as the parent process for later user processes, so even administrators browse the web as normal users, and malware they download can't overwrite system files unless given later authorisation. When a task requires admin privilege, the user gets an *elevation prompt* asking them for an admin password. (Apple's macOS is similar although the details under the hood differ somewhat.) As admin users are often tricked into installing malicious software, Vista added mandatory access controls in the form of file integrity levels. The basic idea is that low-integrity processes (such as code you download from the Internet) should not be able to modify high-integrity data (such as system files) in the absence of some trusted process (such as verification of a signature by Microsoft on the code in question).

In 2012, Windows 8 added *dynamic access control* which lets you control

user access by context, such as their work PC versus their home PC and their phone; this is done via account attributes in Active Directory, which appear as claims about a user, or in Kerberos tickets as claims about a domain. In 2016, Windows 8.1 added a cleaner abstraction with *principals*, which can be a user, computer, process or thread running in a security context or a group to which such a principal belongs, and *security identifiers* (SIDs) which represent such principals. When a user signs in, they get tickets with the SIDs to which they belong. Windows 8.1 also prepared for the move to cloud computing by adding *Microsoft accounts* (formerly LiveID), whereby a user signs in to a Microsoft cloud service rather than to a local server. Where credentials are stored locally, it protects them using virtualisation. Finally, Windows 10 added a number of features to support the move to cloud computing with a diversity of client devices, ranging from certificate pinning (which we'll discuss in the chapter on Network Security) to the abolition of the old secure attention sequence ctrl-alt-del (which is hard to do on touch-screen devices and which users didn't understand anyway).

To sum up, Windows evolved to provide a richer and more flexible set of access control tools than any system previously sold in mass markets. It was driven by corporate customers who need to manage tens of thousands of staff performing hundreds of different job roles across hundreds of different sites, providing internal controls to limit the damage that can be done by small numbers of dishonest staff or infected machines. (How such controls are actually designed will be our topic in the chapter on Banking and Bookkeeping.) The driver for this development was the fact that Microsoft made over half of its revenue from firms that licensed more than 25,000 seats; but the cost of the flexibility that corporate customers demanded is complexity. Setting up access control for a big Windows shop is a highly skilled job.

### 6.2.10 Middleware

Doing access control at the level of files and programs was fine in the early days of computing, when these were the resources that mattered. Since the 1980s, growing scale and complexity has led to access control being done at other levels instead of (or as well as) at the operating system level. For example, bookkeeping systems often run on top of a database product such as Oracle, which looks to the operating system as one large file. So most of the access control has to be done in the database; all the operating system supplies may be an authenticated ID for each user who logs on. And since the 1990s, a lot of the work at the client end has been done by the web browser.

#### 6.2.10.1 Database access controls

Before people started using websites for shopping, database security was largely a back-room concern. But enterprises now have critical databases to handle inventory, dispatch and e-commerce, fronted by web servers that pass transactions to the databases directly. These databases now contain much of the data that matter to our lives – bank accounts, vehicle registrations and employment records – and failures sometimes expose them to random online users.

Database products, such as Oracle, DB2 and MySQL, have their own access control mechanisms, which are modelled on operating-system mechanisms, with privileges typically available for both users and objects (so the mechanisms are a mixture of access control lists and capabilities). However, the typical database access control architecture is comparable in complexity with Windows; modern databases are intrinsically complex, as are the things they support – typically business processes involving higher levels of abstraction than files or domains. There may be access controls aimed at preventing any user learning too much about too many customers; these tend to be stateful, and may deal with possible statistical inference rather than simple yes-no access rules. I devote a whole chapter in Part 2 to exploring the topic of Inference Control.

Ease of administration is often a bottleneck. In companies I've advised, the operating-system and database access controls have been managed by different departments, which don't talk to each other; and often IT departments have to put in crude hacks to make the various access control systems seem to work as one, but which open up serious holes.

Some products let developers bypass operating-system controls. For example, Oracle has both operating system accounts (whose users must be authenticated externally by the platform) and database accounts (whose users are authenticated directly by the Oracle software). It is often convenient to use the latter, to save the effort of synchronising with what other departments are doing. In many installations, the database is accessible directly from the outside; and even where it's shielded by a web service front-end, this often contains loopholes that let SQL code be inserted into the database.

Database security failures can thus cause problems directly. The Slammer worm in 2003 propagated itself using a stack-overflow exploit against Microsoft SQL Server 2000 and created large amounts of traffic as compromised machines sent floods of attack packets to random IP addresses.

Just as Windows is tricky to configure securely, because it's so complicated, the same goes for the typical database system. If you ever have to lock one down – or even just understand what's going on – you had better read a specialist textbook, such as [1174], or get in an expert.

### 6.2.10.2 Browsers

The web browser is another middleware platform on which we rely for access control and whose complexity often lets us down. The main access control rule is the *same-origin policy* whereby JavaScript or other active content on a web page is only allowed to communicate with the IP address that it originally came from; such code is run in a *sandbox* to prevent it altering the host system, as I'll describe in the next section. But many things can go wrong.

In previous editions of this book, we considered web security to be a matter of how the servers were configured, and whether this led to cross-site vulnerabilities. For example a malicious website can include links or form buttons aimed at creating a particular side-effect:

```
https://mybank.com/transfer.cgi?amount=10000USD&recipient=thief
```

The idea is that if a user clicks on this who is logged into `mybank.com`, there may be a risk that the transaction will be executed, as there's a valid session cookie. So payment websites deploy countermeasures such as using short-lived sessions and an anti-CSRF token (an invisible MAC of the session cookie), and checking the `Referer:` header. There are also issues around web authentication mechanisms; I described OAuth briefly in section 4.7.4. If you design web pages for a living you had better understand the mechanics of all this in rather more detail (see for example [119]); but many developers don't take enough care. For example, as I write in 2020, Amazon Alexa has just turned out to have a misconfigured policy on cross-origin resource sharing, which meant that anyone who compromised another Amazon subdomain could replace the skills on a target Alexa with malicious ones [1481].

By now there's a realisation that we should probably have treated browsers as access control devices all along. After all, the browser is the place on your laptop where you run code written by people you don't want to trust and who will occasionally be malicious; as we discussed earlier, mobile-phone operating systems run different apps as different users to give even more robust protection. Even in the absence of malice, you don't want to have to reboot your browser if it hangs because of a script in one of the tabs. (Chrome tries to ensure this by running each tab in a separate operating-system process.)

Bugs in browsers are exploited in *drive-by download* attacks, where visiting an attack web page can infect your machine, and even without this the modern web environment is extremely difficult to control. Many web pages are full of trackers and other bad things, supplied by multiple ad networks and data brokers, which make a mockery of the intent behind the same-origin policy. Malicious actors can even use web services to launder origin: for example, the attacker makes a mash-up of the target site plus some evil scripts of his own, and then gets the victim to view it through a proxy such as Google Translate [1854]. A prudent person will go to their bank website by typing in the URL directly, or using a bookmark; unfortunately, the marketing industry trains everyone to click on links in emails.

### 6.2.11 Sandboxing

The late 1990s saw the emergence of yet another type of access control: the software *sandbox*, introduced by Sun with its Java programming language. The model is that a user wants to run some code that she has downloaded as an applet, but is concerned that the applet might do something nasty, such as stealing her address book and mailing it off to a marketing company, or just hogging the CPU and running down the battery.

The designers of Java tackled this problem by providing a 'sandbox' – a restricted environment in which the code has no access to the local hard disk (or at most only temporary access to a restricted directory), and is only allowed to communicate with the host it came from (the *same-origin policy*). This is enforced by having the code executed by an interpreter – the Java Virtual Machine (JVM) – with only limited access rights [783]. This idea was adapted to JavaScript, the main scripting language used in web pages, though it's actually a different language; and other active content too. A version of Java is also used

on smartcards so they can support applets written by different firms.

### 6.2.12 Virtualisation

Virtualisation is what powers cloud computing; it enables a single machine to emulate a number of machines independently, so that you can rent a *virtual machine* (VM) in a data centre for a few tens of dollars a month rather than having to pay maybe a hundred for a whole server. Virtualisation was invented in the 1960s by IBM [496]; a single machine could be partitioned using VM/370 into multiple virtual machines. Initially this was about enabling a new mainframe to run legacy apps from several old machine architectures; it soon became normal for a company that bought two computers to use one for its production environment and the other as a series of logically separate machines for development, testing, and minor applications. It's not enough to run a virtual machine monitor (VMM) on top of a host operating system, and then run other operating systems on top; you have to deal with sensitive instructions that reveal processor state such as absolute addresses and the processor clock. Working VMMs appeared for Intel platforms with VMware ESX Server in 2003 and (especially) Xen in 2003, which accounted for resource usage well enough to enable AWS and the cloud computing revolution. Things can be done more cleanly with processor support, which Intel has provided since 2006 with VT-x, and whose details I'll discuss below. VM security claims rest to some extent on the argument that a VMM hypervisor's code can be much smaller than an operating system and thus easier to code-review and secure; whether there are actually fewer vulnerabilities is of course an empirical question [1575].

At the client end, virtualisation allows people to run a guest operating system on top of a host (for example, Windows on top of macOS), which offers not just flexibility but the prospect of better containment. For example, an employee might have two copies of Windows running on their laptop – a locked-down version with the office environment, and another for use at home. Samsung offers Knox, which creates a virtual machine on a mobile phone that an employer can lock down and manage remotely, while the user enjoys a normal Android as well on the same device.

But using virtualisation to separate security domains on clients is harder than it looks. People need to share data between multiple VMs and if they use ad-hoc mechanisms, such as USB sticks and webmail accounts, this undermines the separation. Safe data sharing is far from trivial. For example, Bromium<sup>5</sup> offers VMs tailored to specific apps on corporate PCs, so you have one VM for Office, one for Acrobat reader, one for your browser and so on. This enables firms to work reasonably securely with old, unsupported software. So how do you download an Office document? Well, the browser exports the file from its VM to the host hard disc, marking it ‘untrusted’, so when the user tries to open it they’re given a new VM which holds that document plus Office and nothing else. When they then email this untrusted document, there’s an Outlook plugin that stops it being rendered in the ‘sent mail’ pane. Things get even more messy with network services integrated into apps; the rules on what sites can access which cookies are complicated, and it’s hard to deal with single signon and workflows

---

<sup>5</sup>Now owned by HP

### 6.3. HARDWARE PROTECTION

---

that cross multiple domains. The clipboard also needs a lot more rules to control it. Many of the rules change from time to time, and are heuristics rather than hard, verifiable access logic. In short, using VMs for separation at the client requires deep integration with the OS and apps if it's to appear transparent to the user, and there are plenty of tradeoffs made between security and usability. In effect, you're retrofitting virtualisation on to an existing OS and apps that were not built for it.

*Containers* have been the hot new topic in the late 2010s. They evolved as a lightweight alternative to virtualisation in cloud computing and are often confused with it, especially by the marketing people. My definition is that while a VM has a complete operating system, insulated from the hardware by a hypervisor, a container is an isolated guest process that shares a kernel with other containers. Container implementations separate groups of processes by virtualising a subset of operating-system mechanisms, including process identifiers, interprocess communication, and namespaces; they also use techniques such as sandboxing and system call filtering. The business incentive is to minimise the guests' size, their interaction complexity and the costs of managing them, so they are deployed along with orchestration tools. Like any other new technology, there are many startups with more enthusiasm than experience. A 2019 survey by Jerry Gamblin disclosed that of the top 1000 containers available to developers on Docker Hub, 194 were setting up blank root passwords [743]. If you're going to use cloud systems, you need to pay serious attention to your choice of tools, and also learn yet another set of access control mechanisms – those offered by the service provider, such as the Amazon AWS Identity and Access Management (IAM). This adds another layer of complexity, which people can get wrong. For example, in 2019 a security firm providing biometric identification services to banks and the police left its entire database unprotected; two researchers found it using Elasticsearch and discovered millions of people's photos, fingerprints, passwords and security clearance levels on a database that they could not only read but write [1864].

But even if you tie down a cloud system properly, there are hardware limits on what the separation mechanisms can achieve. In 2018, two classes of powerful side-channel attacks were published: Meltdown and Spectre, which I discuss in the following section and at greater length in the chapter on side channels. Those banks that use containers to deploy payment processing rely, at least implicitly, on their containers being difficult to target in a cloud the size of Amazon's or Google's. For a comprehensive survey of the evolution of virtualisation and containers, see Randal [1575].

## 6.3 Hardware Protection

Most access control systems set out not just to control what users can do, but to limit what programs can do as well. In many systems, users can either write programs, or download and install them, and these programs may be buggy or even malicious.

Preventing one process from interfering with another is the *protection problem*. The *confinement problem* is that of preventing programs communicating

### **6.3. HARDWARE PROTECTION**

---

outward other than through authorized channels. There are several flavours of each. The goal may be to prevent active interference, such as memory overwriting, or to stop one process reading another's memory directly. This is what commercial operating systems set out to do. Military systems may also try to protect *metadata* – data about other data, or subjects, or processes – so that, for example, a user can't find out what other users are logged on to the system or what processes they're running.

Unless one uses sandboxing techniques (which are too restrictive for general programming environments), solving the protection problem on a single processor means, at the very least, having a mechanism that will stop one program from overwriting another's code or data. There may be areas of memory that are shared to allow interprocess communication; but programs must be protected from accidental or deliberate modification, and must have access to memory that is similarly protected.

This usually means that hardware access control must be integrated with the processor's memory management functions. A classic mechanism is *segment addressing*. Memory is addressed by two registers, a segment register that points to a segment of memory, and an address register that points to a location within that segment. The segment registers are controlled by the operating system, often by a component of it called the *reference monitor* which links the access control mechanisms with the hardware.

The implementation has become more complex as processors themselves have. Early IBM mainframes had a two-state CPU: the machine was either in authorized state or it was not. In the latter case, the program was restricted to a memory segment allocated by the operating system; in the former, it could write to segment registers at will. An authorized program was one that was loaded from an authorized library.

Any desired access control policy can be implemented on top of this, given suitable authorized libraries, but this is not always efficient; and system security depended on keeping bad code (whether malicious or buggy) out of the authorized libraries. So later processors offered more complex hardware mechanisms. Multics, an operating system developed at MIT in the 1960s and which inspired Unix, introduced *rings of protection* which express differing levels of privilege: ring 0 programs had complete access to disk, supervisor states ran in ring 2, and user code at various less privileged levels [1684]. Many of its features have been adopted in more recent processors.

There are a number of general problems with interfacing hardware and software security mechanisms. For example, it often happens that a less privileged process such as application code needs to invoke a more privileged process (e.g. a device driver). The mechanisms for doing this need to be designed with care, or security bugs can be expected. Also, performance may depend quite drastically on whether routines at different privilege levels are called by reference or by value [1684].

### 6.3.1 Intel processors

The Intel 8088/8086 processors used in early PCs had no distinction between system and user mode, and thus any running program controlled the whole machine<sup>6</sup>. The 80286 added protected segment addressing and rings, so for the first time a PC could run proper operating systems. The 80386 had built-in virtual memory, and large enough memory segments (4 Gb) that they could be ignored and the machine treated as a 32-bit flat address machine. The 486 and Pentium series chips added more performance (caches, out of order execution and additional instructions such as MMX).

The rings of protection are supported by a number of mechanisms. The current privilege level can only be changed by a process in ring 0 (the kernel). Procedures cannot access objects in lower-level rings directly but there are *gates* that allow execution of code at a different privilege level and manage the supporting infrastructure, such as multiple stack segments.

From 2006, Intel added hardware support for x86 virtualisation, known as Intel VT, which helped drive the adoption of cloud computing. Some processor architectures such as S/370 and PowerPC are easy to virtualise, and the theoretical requirements for this had been established in 1974 by Gerald Popek and Robert Goldberg [1532]; they include that all sensitive instructions that expose raw processor state be privileged instructions. The native Intel instruction set, however, has sensitive user-mode instructions, requiring messy workarounds such as application code rewriting and patches to hosted operating systems. Adding VMM support in hardware means that you can run an operating system in ring 0 as it was designed; the VMM has its own copy of the memory architecture underneath. You still have to trap sensitive opcodes, but system calls don't automatically require VMM intervention, you can run unmodified operating systems, things go faster and systems are generally more robust. Modern Intel CPUs now have nine rings: ring 0–3 for normal code, under which is a further set of ring 0–3 VMM root mode for the hypervisor, and at the bottom is *system management mode* (SMM) for the BIOS. In practice, the four levels that are used are SMM, ring 0 of VMX root mode, the normal ring 0 for the operating system, and ring 3 above that for applications.

In 2015, Intel released Software Guard eXtensions (SGX), which lets trusted code run in an *enclave* – an encrypted section of the memory – while the rest of the code is executed as usual. The company had worked on such architectures in the early years of the Trusted Computing initiative, but let things slide until it needed an enclave architecture to compete with TrustZone, which I discuss in the next section. The encryption is performed by a Memory Encryption Engine (MEE), while SGX also introduces new instructions and memory-access checks to ensure non-enclave processes cannot access enclave memory (not even root processes). SGX has been promoted for DRM and securing cloud VMs, particularly those containing crypto keys, credentials or sensitive personal information; this is under threat from Spectre and similar attacks, which I discuss in detail in the chapter on side channels. Since SGX's security perimeter is the CPU, its software is encrypted in main memory, which imposes real penalties

---

<sup>6</sup>They had been developed on a crash programme to save market share following the advent of RISC processors and the market failure of the iAPX432.

in both time and space. Another drawback used to be that SGX code had to be signed by Intel. The company has now delegated signing (so bad people can get code signed) and from SGXv2 will open up the root of trust to others. So people are experimenting with SGX malware, which can remain undetectable by anti-virus software. As SGX apps cannot issue syscalls, it had been hoped that enclave malware couldn't do much harm, yet Michael Schwarz, Samuel Weiser and Daniel Gruss have now worked out how to mount stealthy return-oriented programming (ROP) attacks from an enclave on a host app; they argue that the problem is a lack of clarity about what enclaves are supposed to do, and that any reasonable threat model must include untrusted enclaves [1688]. This simple point may force a rethink of enclave architectures; Intel says 'In the future, Intel's control-flow enforcement technology (CET) should help address this threat inside SGX'<sup>7</sup>. As for what comes next, AMD released full system memory encryption in 2016, and Intel announced a competitor. This aimed to deal with cold-boot and DMA attacks, and protect code against an untrusted hypervisor; it might also lift space and performance limits on next-generation enclaves. However, Jan Werner and colleagues found multiple inference and data-injection attacks on AMD's offering when it's used in a virtual environment. [2010]. There's clearly some way to go.

As well as the access-control vulnerabilities, there are crypto issues, which I'll discuss in the chapter on Advanced Cryptographic Engineering.

#### 6.3.2 Arm processors

The Arm is the processor core most commonly used in phones, tablets and IoT devices; billions have been used in mobile phones alone, with a high-end device having several dozen Arm cores of various sizes in its chipset. The original Arm (which stood for *Acorn Risc Machine*) was the first commercial RISC design; it was released in 1985, just before MIPS. In 1991, Arm became a separate firm which, unlike Intel, does not own or operate any fabs: it licenses a range of processor cores, which chip designers include in their products. Early cores had a 32-bit datapath and contained fifteen registers, of which seven were shadowed by banked registers for system processes to cut the cost of switching context on interrupt. There are multiple supervisor modes, dealing with fast and normal interrupts, the system mode entered on reset, and various kinds of exception handling. The core initially contained no memory management, so Arm-based designs could have their hardware protection extensively customized; there are now variants with *memory protection units* (MPUs), and others with *memory management units* (MMUs) that handle virtual memory as well.

In 2011, Arm launched version 8, which supports 64-bit processing and enables multiple 32-bit operating systems to be virtualised. Hypervisor support added yet another supervisor mode. The cores come in all sizes, from large 64-bit superscalar processors with pipelines over a dozen stages deep, to tiny ones for cheap embedded devices.

TrustZone is a security extension that supports the 'two worlds' model men-

---

<sup>7</sup>The best defence against ROP attacks in 2019 appears to be Apple's mechanism, in the iPhone X3 and later, for signing pointers with a key that's kept in a register; this stops ROP attacks as the attacker can't guess the signatures.

tioned above; it was made available to mobile phone makers in 2004 [44]. Phones were the ‘killer app’ for enclaves as operators wanted to lock subsidised phones and regulators wanted to make the baseband software that controls the RF functions tamper-resistant [1239]. TrustZone supports an open world for a normal operating system and general-purpose applications, plus a closed enclave to handle sensitive operations such as cryptography and critical I/O (in a mobile phone, this can include the SIM card and the fingerprint reader). Whether the processor is in a secure or non-secure state is orthogonal to whether it’s in user mode or a supervisor mode (though it must choose between secure and hypervisor mode). The closed world hosts a single *trusted execution environment* (TEE) with separate stacks, a simplified operating system, and typically runs only trusted code signed by the OEM – although Samsung’s Knox, which sets out to provide ‘home’ and ‘work’ environments on your mobile phone, allows regular rich apps to execute in the secure environment.

Although TrustZone was released in 2004, it was kept closed until 2015; OEMs used it to protect their own interests and didn’t open it up to app developers, except occasionally under NDA. As with Intel SGX, there appears to be no way yet to deal with malicious enclave apps, which might come bundled as DRM with gaming apps or be mandated by authoritarian states; and, as with Intel SGX, enclave apps created with TrustZone can raise issues of transparency and control, which can spill over into auditability, privacy and much else. Again, company insiders mutter ‘wait and see’; no doubt we shall.

Arm’s latest offering is CHERI<sup>8</sup> which adds fine-grained capability support to Arm CPUs. At present, browsers such as Chrome put tabs in different processes, so that one webpage can’t slow down the other tabs if its scripts run slowly. It would be great if each object in each web page could be sandboxed separately, but this isn’t possible because of the large cost, in terms of CPU cycles, of each inter-process context switch. CHERI enables a process spawning a subthread to allocate it read and write accesses to specific ranges of memory, so that multiple sandboxes can run in the same process. This was announced as a product in 2018 and we expect to see first silicon in 2021. The long-term promise of this technology is that, if it were used thoroughly in operating systems such as Windows, Android and iOS, it would have prevented most of the zero-day exploits of recent years. Incorporating a new protection technology at scale costs real money, just like the switch from 32-bit to 64-bit CPUs, but it could save the cost of lots of patches.

## 6.4 What Goes Wrong

Popular operating systems such as Android, Linux and Windows are very large and complex, with their features tested daily by billions of users under very diverse circumstances. Many bugs are found, some of which give rise to vulnerabilities, which have a typical lifecycle. After discovery, a bug is reported to a CERT or to the vendor; a patch is shipped; the patch is reverse-engineered, and an exploit may be produced; and people who did not apply the patch in time

---

<sup>8</sup>Full disclosure: this was developed by a team of my colleagues at Cambridge and elsewhere, led by Robert Watson.

#### 6.4. WHAT GOES WRONG

---

may find that their machines have been compromised. In a minority of cases, the vulnerability is exploited at once rather than reported – called a *zero-day* exploit as attacks happen from day zero of the vulnerability’s known existence. The economics, and the ecology, of the vulnerability lifecycle are the subject of intensive study by security economists; I’ll discuss this in Part III.

The traditional goal of an attacker was to get a normal account on the system and then become the system administrator, so they could take over the system completely. The first step might have involved guessing, or social-engineering, a password, and then using an operating-system bug to escalate from user to root [1129].

The user/root distinction became less important in the twenty-first century for two reasons. First, Windows PCs were the most common online devices (until 2017 when Android overtook them) so they were the most common attack targets; and as they ran many applications as administrator, any application that could be compromised gave administrator access. Second, attackers come in two basic types: targeted attackers, who want to spy on a specific individual and whose goal is typically to acquire access to that person’s accounts; and scale attackers, whose goal is typically to compromise large numbers of PCs, which they can organise into a botnet in order to make money. This, too, doesn’t require administrator access. Even if your mail client does not run as administrator, it can still be useful to a spammer who takes control.

However, botnet herders do prefer to install *rootkits* which, as their name suggests, run as root; they are also known as *remote access trojans* or RATs. The user/root distinction does still matter in business environments, where you do not want such a kit installed as an *advanced persistent threat* by a hostile intelligence agency, or corporate espionage firm, or by a crime gang doing reconnaissance to set you up for a large fraud.

A separate distinction is whether an exploit is *wormable* – whether it can be used to spread malware quickly online from one machine to another without human intervention. The Morris worm was the first large-scale case of this, and there have been many since. I mentioned WannaCry and NotPetya in chapter 2; these used a vulnerability developed by the NSA and then leaked to other state actors. Operating system vendors react quickly to wormable exploits, typically releasing out-of-sequence patches, because of the scale of the damage they can do. The most troublesome wormable exploits at the time of writing are variants of Mirai, a worm used to take over IoT devices that use known root passwords. This appeared in October 2016 to exploit CCTV cameras, and hundreds of versions have been produced since, adapted to take over different vulnerable devices and recruit them into botnets. Wormable exploits often use root access but don’t have to; it is sufficient that the exploit be capable of automatic onward transmission<sup>9</sup>.

In any case, the basic types of technical attack have not changed hugely in a generation and I’ll now consider them briefly.

---

<sup>9</sup>In rare cases even human transmission can make malware spread quickly: an example was the ILOVEYOU worm which spread itself in 2000 via an email with that subject line, which caused enough people to open it, running a script that caused it to be sent to everyone in the new victim’s address book.

### 6.4.1 Smashing the stack

The classic software exploit is the memory overwriting attack, colloquially known as ‘smashing the stack’, as used by the Morris worm in 1988; this infected so many Unix machines that it disrupted the Internet and brought malware forcefully to the attention of the mass media [1806]. Attacks involving violations of memory safety accounted for well over half the exploits against operating systems in the late 1990s and early 2000s [487] but the proportion has been dropping slowly since then.

Programmers are often careless about checking the size of arguments, so an attacker who passes a long argument to a program may find that some of it gets treated as code rather than data. The classic example, used in the Morris worm, was a vulnerability in the Unix `finger` command. A common implementation of this would accept an argument of any length, although only 256 bytes had been allocated for this argument by the program. When an attacker used the command with a longer argument, the trailing bytes of the argument ended up overwriting the stack and being executed by the system.

The usual exploit technique was to arrange for the trailing bytes of the argument to have a *landing pad* – a long space of *no-operation* (NOP) commands, or other register commands that didn’t change the control flow, and whose task was to catch the processor if it executed any of them. The landing pad delivered the processor to the attack code which will do something like creating a shell with administrative privilege directly (see Figure 6.5).

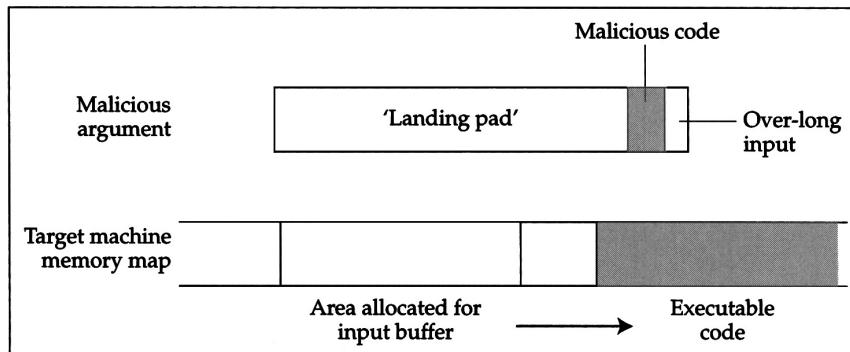


Figure 6.5: – stack smashing attack

Stack-overwriting attacks were around long before 1988. Most of the early 1960s time-sharing systems suffered from this vulnerability, and fixed it [804]. Penetration testing in the early ’70s showed that one of the most frequently-used attack strategies was still “unexpected parameters” [1165]. Intel’s 80286 processor introduced explicit parameter checking instructions – verify read, verify write, and verify length – in 1982, but they were avoided by most software designers to prevent architecture dependencies. Stack overwriting attacks have been found against all sorts of programmable devices – even against things like smartcards and hardware security modules, whose designers really should have known better.

### 6.4.2 Other technical attacks

Many vulnerabilities are variations on the same general theme, in that they occur when data in grammar A is interpreted as being code in grammar B. A stack overflow is when data are accepted as input (e.g. a URL) and end up being executed as machine code. These are failures of *type safety*. In fact, a stack overflow can be seen either as a memory safety failure or as a failure to sanitise user input, but there are purer examples of each type.

The *use after free* type of safety failure is now the most common cause of remote execution vulnerabilities and has provided a lot of attacks on browsers in recent years. It can happen when a chunk of memory is freed and then still used, perhaps because of confusion over which part of a program is responsible for freeing it. If a malicious chunk is now allocated, it may end up taking its place on the heap, and when an old innocuous function is called a new, malicious function may be invoked instead. There are many other variants on the memory safety theme; buffer overflows can be induced by improper string termination, passing an inadequately sized buffer to a path manipulation function, and many other subtle errors. See Gary McGraw's book '*Software Security*' [1266] for a taxonomy.

*SQL injection attacks* are the most common attack based on failure to sanitise input, and arise when a careless web developer passes user input to a back-end database without checking to see whether it contains SQL code. The game is often given away by error messages, from which a capable and motivated user may infer enough to mount an attack. There are similar command-injection problems afflicting other languages used by web developers, such as PHP. The usual remedy is to treat all user input as suspicious and validate it. But this can be harder than it looks, as it's difficult to anticipate all possible attacks and the filters written for one shell may fail to be aware of extensions present in another. Where possible, one should only act on user input in a safe context, by designing such attacks out; where it's necessary to blacklist specific exploits, the mechanism needs to be competently maintained.

Once such type-safety and input-sanitisation attacks are dealt with, *race conditions* are probably next. These occur when a transaction is carried out in two or more stages, where access rights are verified at the first stage and something sensitive is done at the second. If someone can alter the state in between the two stages, this can lead to an attack. A classic example arose in early versions of Unix, where the command to create a directory, '`mkdir`', used to work in two steps: the storage was allocated, and then ownership was transferred to the user. Since these steps were separate, a user could initiate a '`mkdir`' in background, and if this completed only the first step before being suspended, a second process could be used to replace the newly created directory with a link to the password file. Then the original process would resume, and change ownership of the password file to the user.

A more modern example arises with the wrappers used in containers to intercept system calls made by applications to the operating system, parse them, and modify them if need be. These wrappers execute in the kernel's address space, inspect the enter and exit state on all system calls, and encapsulate only security logic. They generally assume that system calls are atomic, but modern

operating system kernels are highly concurrent. System calls are not atomic with respect to each other; there are many possibilities for two system calls to race each other for access to shared memory, which gives rise to *time-of-check-to-time-of-use* (TOCTTOU) attacks. An early (2007) example calls a path whose name spills over a page boundary by one byte, causing the kernel to sleep while the page is fetched; it then replaces the path in memory [1992]. There have been others since, and as more processors ship in each CPU chip as time passes, and containers become an ever more common way of deploying applications, this sort of attack may become more and more of a problem. Some operating systems have features specifically to deal with concurrency attacks, but this field is still in flux.

A different type of timing attack can come from backup and recovery systems. It's convenient if you can let users recover their own files, rather than having to call a sysadmin – but how do you protect information assets from a time traveller? People can reacquire access rights that were revoked, and play even more subtle tricks.

One attack that has attracted a lot of research effort recently is *return-oriented programming* (ROP) [1708]. Many modern systems try to prevent type safety attacks by *data execution prevention* – marking memory as either code or data, a measure that goes back to the Burroughs 5000; and if all the code is signed, surely you'd think that unauthorised code cannot be executed? Wrong! An attacker can look for *gadgets* – sequences of instructions with some useful effect, ending in a return. By collecting enough gadgets, it's possible to assemble a machine that's Turing powerful, and implement our attack code as a chain of ROP gadgets. Then all one has to do is seize control of the call stack. This evolved from the *return-to-libc attack* which uses the common shared library `libc` to provide well-understood gadgets; many variants have been developed since, including an attack that enables malware in an SGX enclave to mount stealthy attacks on host apps [1688]. The latest attack variant, *block-oriented programming* (BOP), can often generate attacks automatically from crashes discovered by program fuzzing, defeating current control-flow integrity controls [964]. This coevolution of attack and defence will no doubt continue.

Finally there are *side channels*. The most recent major innovation in attack technology targets CPU pipeline behaviour. In early 2018, two game-changing attacks pioneered the genre: *Meltdown*, which exploits side-channels created by out-of-order execution on Intel processors [1172], and *Spectre*, which exploits speculative execution on Intel, AMD and Arm processors [1068]. The basic idea is that large modern CPUs' pipelines are so long and complex that they look ahead and anticipate the next dozen instructions, even if these are instructions that the current process wouldn't be allowed to execute (imagine the access check is two instructions in the future and the read operation it will forbid is two instructions after that). The path not taken can still load information into a cache and thus leak information in the form of delays. With some cunning, one process can arrange things to read the memory of another. I will discuss Spectre and Meltdown in more detail in the chapter on side channels in the second part of this book. Although mitigations have been published, further attacks of the same general kind keep on being discovered, and it may take several years and a new generation of processors before they are brought entirely under control.

It all reminds me of the saying by Roger Needham at the head of this chapter. Optimisation consists of replacing something that works with something that almost works, but is cheaper; and modern CPUs are so heavily optimised that we're bound to see more variants on the Spectre theme. Such attacks limit the protection that can be offered not just by containers and VMs, but also by enclave mechanisms such as TrustZone and SGX. In particular, they may stop careful firms from entrusting high-value cryptographic keys to enclaves and prolong the service life of old-fashioned hardware cryptography.

#### 6.4.3 User interface failures

A common way to attack a fortress is to trick the guards into helping you, and operating systems are no exception. One of the earliest attacks was the *Trojan Horse*, a program the administrator is invited to run but which contains a nasty surprise. People would write games that checked whether the player was the system administrator, and if so would create another administrator account with a known password. A variant was to write a program with the same name as a common system utility, such as the `ls` command which lists all the files in a Unix directory, and design it to abuse the administrator privilege (if any) before invoking the genuine utility. You then complain to the administrator that something's wrong with the directory. When they enter the directory and type `ls` to see what's there, the damage is done. This is an example of the *confused deputy* problem: if A does some task on behalf of B, and its authority comes from both A and B, and A's authority exceeds B, things can go wrong. The fix in this particular case was simple: an administrator's 'PATH' variable (the list of directories to be searched for a suitably-named program when a command is invoked) should not contain '.' (the symbol for the current directory). Modern Unix versions ship with this as a default. But it's still an example of how you have to get lots of little details right for access control to be robust, and these details aren't always obvious in advance.

Perhaps the most serious example of user interface failure, in terms of the number of systems historically attacked, consists of two facts: first, Windows is forever popping up confirmation dialogues, which trained people to click boxes away to get their work done; and second, that until 2006 a user needed to be the administrator to install anything. The idea was that restricting software installation to admins enabled Microsoft's big corporate customers, such as banks and government departments, to lock down their systems so that staff couldn't run games or other unauthorised software. But in most environments, ordinary people need to install software to get their work done. So hundreds of millions of people had administrator privileges who shouldn't have needed them, and installed malicious code when a website simply popped up a box telling them to do something. This was compounded by the many application developers who insisted that their code run as root, either out of laziness or because they wanted to collect data that they really shouldn't have had. Windows Vista started to move away from this, but a malware ecosystem is now well established in the PC world, and one is starting to take root in the Android ecosystem as businesses pressure people to install apps rather than using websites, and the apps demand access to all sorts of data and services that they really shouldn't have. We'll

discuss this later in the chapter on phones.

#### 6.4.4 Remedies

Software security is not all doom and gloom; things got substantially better during the 2000s. At the turn of the century, 90% of vulnerabilities were buffer overflows; by the time the second edition of this book came out in 2008, it was just under half, and now it's even less. Several things made a difference.

1. The first consists of specific defences. *Stack canaries* are a random number inserted by the compiler next to the return address on the stack. If the stack is overwritten, then with high probability the canary will change [487]. *Data execution prevention* (DEP) marks all memory as either data or code, and prevents the former being executed; it appeared in 2003 with Windows XP. *Address space layout randomisation* (ASLR) arrived at the same time; by making the memory layout different in each instance of a system, it makes it harder for an attacker to predict target addresses. This is particularly important now that there are toolkits to do ROP attacks, which bypass DEP. *Control flow integrity* mechanisms involve analysing the possible control-flow graph at compile time and enforcing this at runtime by validating indirect control-flow transfers; this appeared in 2005 and was incorporated in various products over the following decade [348]. However the analysis is not precise, and block-oriented programming attacks are among the tricks that have evolved to exploit the gaps [964].
2. The second consists of better general-purpose tools. Static-analysis programs such as Coverity can find large numbers of potential software bugs and highlight ways in which code deviates from best practice; if used from the start of a project, they can make a big difference. (If added later, they can throw up thousands of alerts that are a pain to deal with.) The radical solution is to use a better language; my colleagues increasingly write systems code in Rust rather than in C or C++<sup>10</sup>.
3. The third is better training. In 2002, Microsoft announced a security initiative that involved every programmer being trained in how to write secure code. (The book they produced for this, '*Writing Secure Code*' [927], is still worth a read.) Other companies followed suit.
4. The latest approach is DevSecOps, which I discuss in Part 3. Agile development methodology is extended to allow very rapid deployment of patches and response to incidents; it may enable the effort put into design, coding and testing to be aimed at the most urgent problems.

Architecture matters; having clean interfaces that evolve in a controlled way, under the eagle eye of someone experienced who has a long-term stake in the security of the product, can make a huge difference. Programs should only have

---

<sup>10</sup>Rust emerged from Mozilla research in 2010 and has been used to redevelop Firefox; it's been voted the favourite language in the Stack Overflow annual survey from 2016–2019.

as much privilege as they need: the *principle of least privilege* [1639]. Software should also be designed so that the default configuration, and in general, the easiest way of doing something, should be safe. Sound architecture is critical in achieving safe defaults and using least privilege. However, many systems are shipped with dangerous defaults and messy code, exposing all sorts of interfaces to attacks like SQL injection that just shouldn't happen. These involve failures of incentives, personal and corporate, as well as inadequate education and the poor usability of security tools.

#### 6.4.5 Environmental creep

Many security failures result when environmental change undermines a security model. Mechanisms that worked adequately in an initial environment often fail in a wider one.

Access control mechanisms are no exception. Unix, for example, was originally designed as a ‘single user Multics’ (hence the name). It then became an operating system to be used by a number of skilled and trustworthy people in a laboratory who were sharing a single machine. In this environment the function of the security mechanisms is mostly to contain mistakes; to prevent one user’s typing errors or program crashes from deleting or overwriting another user’s files. The original security mechanisms were quite adequate for this purpose.

But Unix security became a classic ‘success disaster’. Over the 50 years since Ken Thomson started work on it at Bell Labs in 1969, Unix was repeatedly extended without proper consideration being given to how the protection mechanisms also needed to be extended. The Berkeley versions assumed an extension from a single machine to a network of machines that were all on one LAN and all under one management. The Internet mechanisms (telnet, ftp, DNS, SMTP) were originally written for mainframes on a secure network. Mainframes were autonomous, the network was outside the security protocols, and there was no transfer of authorisation. So remote authentication, which the Berkeley model really needed, was simply not supported. The Sun extensions such as NFS added to the party, assuming a single firm with multiple trusted LANs. We’ve had to retrofit protocols like Kerberos, TLS and SSH as duct tape to hold the world together. The arrival of billions of phones, which communicate sometimes by wifi and sometimes by a mobile network, and which run apps from millions of authors (most of them selfish, some of them actively malicious), has left security engineers running ever faster to catch up.

Mixing many different models of computation together has been a factor in the present chaos. Some of their initial assumptions still apply partially, but none of them apply globally any more. The Internet now has billions of phones, billions of IoT devices, maybe a billion PCs, and millions of organisations whose managers not only fail to cooperate but may be in conflict. There are companies that compete; political groups that despise each other, and nation states that are at war with each other. Users, instead of being trustworthy but occasionally incompetent, are now largely unskilled – but some are both capable and hostile. Code used to be simply buggy – but now there is a lot of malicious code out there. Attacks on communications used to be the purview of intelligence agencies – now they can be done by youngsters who’ve downloaded attack tools from the

net and launched them without any real idea of how they work.

## **6.5 Summary**

Access control mechanisms operate at a number of levels in a system, from the hardware up through the operating system and middleware like browsers to the applications. Higher-level mechanisms can be more expressive, but also tend to be more vulnerable to attack for a variety of reasons ranging from intrinsic complexity to implementer skill.

The main function of access control is to limit the damage that can be done by particular groups, users, and programs whether through error or malice. The most widely fielded examples are Android and Windows at the client end and Linux at the server end; they have a common lineage and many architectural similarities. The basic mechanisms (and their problems) are pervasive. Most attacks involve the opportunistic exploitation of bugs; products that are complex, widely used, or both are particularly likely to have vulnerabilities found and turned into exploits. Many techniques have been developed to push back on the number of implementation errors, to make it less likely that the resulting bugs give rise to vulnerabilities, and harder to turn the vulnerabilities into exploits; but the overall dependability of large software systems improves only slowly.

## **Research Problems**

Most of the issues in access control were identified by the 1960s or early 1970s and were worked out on experimental systems such as Multics [1684] and the CAP [2020]. Much of the research in access control systems since then has involved reworking the basic themes in new contexts, such as mobile phones.

Recent threads of research include enclaves, and the CHERI mechanisms for adding finer-grained access control. Another question is: how will developers use such tools effectively?

In the second edition I predicted that ‘a useful research topic for the next few years will be how to engineer access control mechanisms that are not just robust but also usable – by both programmers and end users.’ Recent work by Yasemin Acar and others has picked that up and developed it into one of the most rapidly-growing fields of security research [11]. Many if not most technical security failures are due at least in part to the poor usability of the protection mechanisms that developers are expected to use. I already mention in the chapter on cryptography how crypto APIs often induce people to use really unsafe defaults, such as encrypting long messages with ECB mode; access control is just as bad, as anyone coming cold to the access control mechanisms in a Windows system or either an Intel or Arm CPU will find.

As a teaser, here’s a new problem. Can we extend what we know about access control at the technical level – whether hardware, OS or app – to the organisational level? In the 20th century, there were a number of security policies proposed, from Bell-LaPadula to Clark-Wilson, which we discuss at greater

length in Part 2. Is it time to revisit this for a world of deep outsourcing and virtual organisations, now that we have interesting technical analogues?

## Further Reading

There's a history of virtualisation and containers by Allison Randal at [1575]; a discussion of how mandatory access controls were adapted to operating systems such as OS X and iOS by Robert Watson in [1993]; and a reference book for Java security written by its architect Li Gong [783]. The Cloud Native Security Foundation is trying to move people towards better open-source practices around containers and other technologies for deploying and managing cloud-native software. Going back a bit, the classic descriptions of Unix security are by Fred Grampp and Robert Morris in 1984 [805] and by Simson Garfinkel and Eugene Spafford in 1996 [753], while the classic on Internet security by Bill Cheswick and Steve Bellovin [221] gives many examples of network attacks on Unix systems.

Carl Landwehr gives a useful reference to many of the flaws found in operating systems in the 1960s through the 1980s [1129]. One of the earliest reports on the subject (and indeed on computer security in general) is by Willis Ware in 1970 [1986]; Butler Lampson's seminal paper on the confinement problem appeared in 1970s [1125] and three years later, another influential early paper was written by Jerry Saltzer and Mike Schroeder [1639]. The textbook we get our students to read on access control issues is Dieter Gollmann's '*Computer Security*' [779]. The standard reference on Intel's SGX and indeed its CPU security architecture is by Victor Costan and Srinivas Devadas [479].

The field of software security is fast-moving; the attacks change significantly (at least in their details) from one year to the next. The classic starting point is Gary McGraw's 2006 book [1266]. Since then we've had ROP attacks, Spectre and much else; a short but useful update is Matthias Payer's *Software Security* [1504]. But to really keep up, it's not enough to just read textbooks; you need to follow security conferences such as Usenix and CCS as well as the security blogs such as Bruce Schneier, Brian Krebs and – dare I say it – our own [lightbluetouchpaper.org](http://lightbluetouchpaper.org). The most detail on the current attacks is probably in Google's Project Zero blog; see for example their analysis of attacks on iPhones found in the wild for an insight into what's involved in hacking modern operating systems with mandatory access control components [204].

Google is a second-price auction tweaked to optimise revenue. Bidders offer to pay prices  $b_i$ , the platform estimates their ad quality as  $e_i$ , based on the ad's relevance and clickthrough rate. It then calculates 'ad rank' as  $a_i = b_i e_i$ . The idea is that if my ad is five times as likely to be clicked on as yours, then my bid of 10c is just as good as your bid of 50c. This is therefore a second-price auction, but based on ranking  $a_i$  rather than  $b_i$ . Thus if I have five times your ad quality, I bid 10c and you bid 40c, then I get the ad and pay 8c. It can be shown that under reasonable assumptions, this maximises platform revenue.

There's one catch, though. Once media become social, then ad quality can easily segue into virality. If your ads are good clickbait and people click on them, you pay less. One outcome was that in the 2016 US Presidential Election, Hilary Clinton paid a lot more per ad than Donald Trump did [1234]. Both auction theory and empirical data show how the drive to optimise platform revenue may lead to ever more extreme content: in addition to virality effects at the auction step, Facebook's delivery algorithms put ads in front of the people most likely to click on them, strengthening the effect of filter bubbles, and that this is not all due to user actions [40]. Some people feel this 'delivery optimisation' should be prohibited by electoral law; certainly it's one more example of mechanisms with structural tension between efficiency and fairness. In fact, in the UK, election ads aren't permitted on TV, along with some other categories such as tobacco. Maybe the cleanest solution in such jurisdictions is to ban them online too, just like tobacco. And ad pricing is not the only way social media promote extreme content; as former Googler Tristan Harris has explained, the platforms' recommender algorithms are also optimised to maximise the time people spend on site, which means not just scrolling feeds and followers, but a bias towards anxiety and outrage. What's more, ad delivery can be skewed by factors such as gender and race by market effects, as advertisers compete for more 'valuable' demographics, and by content effects because of the appeal of ad headlines or images; this can be deliberate or accidental, and can affect a broad range of ads including employment and housing [39]. This all raises thorny political issues at the boundary between economics and psychology, but economic tools such as auction theory can often be used to unpick them.

## **8.6 The economics of security and dependability**

Economists used to see a simple interaction between economics and security: richer nations could afford bigger armies. But after 1945, nuclear weapons were thought to decouple national survival from economic power, and the fields of economics and strategic studies drifted apart [1238]. It has been left to the information security world to re-establish the connection.

Round about 2000, a number of us noticed persistent security failures that appeared at first sight to be irrational, but which we started to understand once we looked more carefully at the incentives facing the various actors. I observed odd patterns of investment by banks in information security measures [54, 55]. Hal Varian looked into why people were not spending as much money on anti-virus software as the vendors hoped [1943]. When the two of us got to discussing these cases in 2001, we suddenly realised that there was an interesting and im-

portant research topic here, so we contacted other people with similar interests and organised a workshop for the following year. I was writing the first edition of this book at the time, and found that describing many of the problems as incentive problems made the explanations much more compelling; so I distilled what I learned from the book's final edit into a paper 'Why Information Security is Hard – An Economic Perspective'. This paper, plus the first edition of this book, got people talking [72]. By the time they came out, the 9/11 attacks had taken place and people were searching for new perspectives on security.

We rapidly found many other examples of security failure associated with institutional incentives, such as hospital systems bought by medical directors and administrators that support their interests but don't protect patient privacy. (Later, we found that patient safety failures often had similar roots.) Jean Camp had been writing about markets for vulnerabilities, and two startups had set up early vulnerability markets. Networking researchers were starting to use auction theory to design strategy-proof routing protocols. The Department of Defense had been mulling over its failure to get vendors to sell them secure systems, as you can see in the second quote at the head of this chapter. Microsoft was thinking about the economics of standards. All these ideas came together at the Workshop on the Economics of Information Security at Berkeley in June 2002, which launched security economics as a new field of study. The picture that started to emerge was of system security failing because the people guarding a system were not the people who suffered the costs of failure. Sometimes, security mechanisms are used to dump risks on others, and if you are one of those others you'd be better off with an insecure system. Put differently, security is often a power relationship; the principals who control what it means in a given system often use it to advance their own interests.

This was the initial insight, and the story of the birth of security economics is told in [78]. But once we started studying the subject seriously, we found that there's a lot more to it than that.

### 8.6.1 Why is Windows so insecure?

The hot topic in 2002, when security economics got going, was this. Why is Windows so insecure, despite Microsoft's dominant market position? It's possible to write much better software, and there are fields such as defense and healthcare where a serious effort is made to produce dependable systems. Why do we not see a comparable effort made with commodity platforms, especially since Microsoft has no real competitors?

By then, we understood the basics of information economics: the combination of high fixed and low marginal costs, network effects and technical lock-in makes platform markets particularly likely to be dominated by single vendors, who stand to gain vast fortunes if they can win the race to dominate the market. In such a race, the Microsoft philosophy of the 1990s – 'ship it Tuesday and get it right by version 3' – is perfectly rational behaviour. In such a race, the platform vendor must appeal not just to users but also to complementers – to the software companies who decide whether to write applications for its platform or for someone else's. Security gets in the way of applications, and it tends to be a lemons market anyway. So the rational vendor engaged in a

race for platform dominance will enable all applications to run as root on his platform<sup>2</sup>, until his position is secure. Then he may add more security – but will be tempted to engineer it in such a way as to maximise customer lock-in, or to appeal to complementers in new markets such as digital media.

The same pattern was also seen in other platform products, from the old IBM mainframe operating systems through telephone exchange switches to the early Symbian operating system for mobile phones. Products are insecure at first, and although they improve over time, many of the new security features are for the vendor’s benefit as much as the user’s. And this is exactly what we saw with Microsoft’s product lines. DOS had no protection at all and kick-started the malware market; Windows 3 and Windows 95 were dreadful; Windows 98 was only slightly better; and security problems eventually so annoyed Microsoft’s customers that finally in 2003 Bill Gates decided to halt development until all its engineers had been on a secure coding course. This was followed by investment in better testing, static analysis tools, and regular patching. The number and lifetime of exploitable vulnerabilities continued to fall through later releases of Windows. But the attackers got better too, and the protection in Windows isn’t all for the user’s benefit. As Peter Gutmann points out, much more effort went into protecting premium video content than into protecting users’ credit card numbers [842].

From the viewpoint of the consumer, markets with lock-in are often ‘bargains then rip-offs’. You buy a nice new printer for \$39.95, then find to your disgust after just a few months that you need two new printer cartridges for \$19.95 each. You wonder whether you’d not be better off just buying a new printer. From the viewpoint of the application developer, markets with standards races based on lock-in look a bit like this. At first it’s really easy to write code for them; later on, once you’re committed, there are many more hoops to jump through. From the viewpoint of the poor consumer, they could be described as ‘poor security, then security for someone else’.

The same pattern can be seen with externalities from security management costs to infrastructure decisions that the industry takes collectively. When racing to establish a dominant position, vendors are tempted to engineer products so that most of the cost of managing security is dumped on the user. A classic example is SSL/TLS encryption. This was adopted in the mid-1990s as Microsoft and Netscape battled for dominance of the browser market. As we discussed in Chapter 5, SSL leaves it up to the user to assess the certificate offered by a web site and decide whether to trust it; and this led to all kinds of phishing and other attacks. Yet dumping the compliance costs on the user made perfect sense at the time; competing protocols such as SET would have saddled banks with the cost of issuing certificates to every customer who wanted to buy stuff online, and that would just have cost too much [524]. The world ended up with an insecure system of credit card payments on the Internet, and with most of the stakeholders trying to dump liability on others in ways that block progress towards a better system.

There are also network effects for bads, and well as for goods. Most malware writers targeted Windows rather than Mac or Linux through the 2000s and

---

<sup>2</sup>To make coding easier, and enable app developers to steal the user’s other data for sale in secondary markets.

2010s as there are simply more Windows machines to infect – leading to an odd equilibrium in which people who were prepared to pay more for their laptop could have a more secure one, albeit one that didn’t run as much software. This model replicated itself when smartphones took over the world in the 2010s; since Android took over from Windows as the world’s most popular operating system, we’re starting to see a lot of bad apps for Android, while people who pay more for an iPhone get better security but less choice. (There, the more stringent policies of Apple’s app store are more important now than market share.)

### 8.6.2 Managing the patching cycle

The second big debate in security economics was about how to manage the patching cycle. If you discover a vulnerability, should you just publish it, which may force the vendor to patch it but may leave people exposed for months until they do so? Or should you report it privately to the vendor – and risk getting a lawyer’s letter threatening an expensive lawsuit if you tell anyone else, after which the vendor just doesn’t bother to patch it?

This debate goes back a long way; as we noted in the preface, the Victorians agonised over whether it was socially responsible to publish books about lockpicking, and eventually concluded that it was [1895]. People have worried more recently about whether the online availability of the US Army Improvised Munitions Handbook [1924] helps terrorists; in some countries it’s a crime to possess a copy.

Security economics provides both a theoretical and a quantitative framework for discussing some issues of this kind. We started in 2002 with simple models in which bugs were independent, identically distributed and discovered at random; these have nice statistical properties, as attackers and defenders are on an equal footing, and the dependability of a system is a function only of the initial code quality and the total amount of time spent testing it [74]. But is the real world actually like that? Or is it skewed by correlated bugs, or by the vendor’s inside knowledge? This led to a big policy debate. Eric Rescorla argued that software is close enough to the ideal that removing one bug makes little difference to the likelihood of an attacker finding another one later, so frequent disclosure and patching were an unnecessary expense unless the same vulnerabilities were likely to be rediscovered [1596]. Ashish Arora and others responded with data showing that public disclosure made vendors fix bugs more quickly; attacks increased to begin with, but reported vulnerabilities declined over time [133]. In 2006, Andy Ozment and Stuart Schechter found that the rate at which unique vulnerabilities were disclosed for the core OpenBSD operating system decreased over a six-year period [1488]. In short, in the right circumstances, software can be more like wine than like milk – it improves with age. (Sustainability is a holy grail, and I discuss it in more detail in Part 3.)

Several further institutional factors helped settle the debate in favour of *responsible disclosure*, also known as *coordinated disclosure*, whereby people report bugs to vendors or to third parties that keep them confidential for a period until patches are available, then let the reporters get credit for their discoveries. One was the political settlement at the end of Crypto War I whereby bugs would be reported to CERT which would share them with the NSA during the bug-

fixing process, as I will discuss later in section 26.2.7.3. This got governments on board. The second was the emergence of commercial vulnerability markets such as those set up by iDefense and TippingPoint, where security researchers could sell bugs; these firms would then disclose each bug responsibly to the vendor, and also work out indicators of compromise that could be sold to firms operating firewall or intrusion-detection services. Third, smart software firms started their own bug-bounty programs, so that security researchers could sell their bugs directly, cutting out middlemen such as CERT and iDefense.

This marketplace sharpened considerably after Stuxnet drove governments to stockpile vulnerabilities. We've seen the emergence of firms like Zerodium that buy bugs and sell them to state actors, and to cyberweapons suppliers that also sell to states; zero-day exploits for platforms such as the iPhone can now sell for a million dollars or more. This had knock-on effects on the supply chain. For example, in 2012 we came across the first case of a volunteer deliberately contributing vulnerable code to an open-source project<sup>3</sup>, no doubt in the hope of a six-figure payoff if it had found its way into widely-used platforms. Already in 2010, Sam Ransbotham had shown that although open-source and proprietary software are equally secure in an ideal model, bugs get turned into exploits faster in the open source world, so attackers target it more [1579]. In 2014, Abdullah Algarni and Yashwant Malaiya surveyed vulnerability markets and interviewed some of the more prolific researchers; a combination of curiosity and economic incentives draw in many able young men, many from less developed countries, some disclose responsibly, some use vulnerability markets to get both money and recognition, while others sell for more money to the black hats; some will offer bugs to the vendor, but if not treated properly will offer them to the bad guys instead. Vendors have responded with comparable offers: at Black Hat 2019, Apple announced a bug bounty schedule that goes up to \$1m for exploits that allow zero-click remote command execution on iOS. Oh, and many of the bug hunters retire after a few years [38]. Like it or not, volunteers running open-source projects now find themselves some capable motivated opponents if their projects get anywhere, and even if they can't match Apple's pocket, it's a good idea to keep as many of the researchers onside as possible.

The lifecycle of a vulnerability now involves not just its discovery, but perhaps some covert use by an intelligence agency or other black-hat actor; then its rediscovery, perhaps by other black hats but eventually by a white hat; the shipment of a patch; and then further exploitation against users who didn't apply the patch. There are tensions between vendors and their customers over the frequency and timing of patch release, as well as with complementers and secondary users over trust. A vulnerability in Linux doesn't just affect the server in your lab and your kid's Raspberry Pi. Linux is embedded everywhere: in your air-conditioner, your smart TV and even your car. This is why responsible disclosure is being rebranded as coordinated disclosure. There may be simply too many firms using a platform for the core developers to trust them all about a forthcoming patch release. There are also thousands of vulnerabilities, of which dozens appear each year in the exploit kits used by criminals (and some no doubt used only once against high-value targets, so they never become known to defense systems). We have to study multiple overlapping ecosystems – of the

---

<sup>3</sup>Webkit, which is used in mobile phone browsers

vulnerabilities indexed by their CVE numbers; of the Indicators of Compromise (IoCs) that get fed to intrusion detection systems; of disclosure to vendors directly, via markets, via CERTs and via ISACs; of the various botnets, crime gangs and state actors; and of the various recorded crime patterns. We have partial correlations between these ecosystems, but the data are generally noisy. I'll come back to all this in Part III.

### **8.6.3 Structural models of attack and defence**

The late Jack Hirshleifer, the founder of conflict theory, told the story of Anarchia, an island whose flood defences were constructed by individual families each of whom maintained a section of the flood wall. The island's flood defence thus depended on the weakest link, that is, the laziest family. He compared this with a city whose defences against missile attack depend on the single best defensive shot [906]. Another example of best-shot is medieval warfare, where there could be a single combat between the two armies' champions. This can lead to different political systems. Medieval Venice, the best example of weakest-link defence because of the risk of flooding, had strong central government, with the merchant families electing a Doge with near-dictatorial powers over flood defence. In much of the rest of late medieval Europe, kings or chieftains led their own armies to kill enemies and seize land; the strongest king built the biggest empire, and this led to a feudal system that optimised the number of men at arms.

Hal Varian extended this model to the dependability of information systems – where performance can depend on the weakest link, the best effort, or the sum-of-efforts [1945]. This last case, the sum-of-efforts, is the modern model for warfare: we pay our taxes and the government hires soldiers. It's more efficient than best-shot (where most people will free-ride behind the heroes), which in turn is more efficient than weakest-link (where everyone will be vulnerable via the laziest). Information security is an interesting mix of all three modes. Program correctness can depend on the weakest link (the most careless programmer introducing a vulnerability) while software vulnerability testing may depend on the sum of everyone's efforts. Security may also depend on the best effort – the actions taken by an individual champion such as a security architect. As more agents are added, systems become more reliable in the sum-of-efforts case but less reliable in the weakest-link case. So as software companies get bigger, they end up hiring more testers and fewer (but more competent) programmers; Microsoft found by the early 2000s that they had more test engineers than software engineers.

Other models of attack and defence include epidemic models of malware spread, which were important back when computer viruses spread from machine to machine via floppy disks, but are of less interest now that we see relatively few wormable exploits; and models of security games that hinge on timing, notably the game of FlipIt by Ron Rivest and colleagues [559]; indeed, there's a whole conference (Gamesec) devoted to game theory and information security. There are also models of social networks. For example, most social networks owe their connectivity to a relatively small number of nodes that have a relatively high number of links to other nodes [1994]. Knocking out these nodes can

rapidly disconnect things; William the Conqueror consolidated England after 1066 by killing the Anglo-Saxon nobility and replacing them with Normans, while Stalin killed the richer peasants. US and British forces similarly targeted highly-connected people in counterinsurgency operations during the Iraq war (and the resulting social breakdown in Sunni areas helped the emergence of ISIS). Such models also suggest that for insurgents to form into cells is the natural and most effective response to repeated decapitation attacks [1373].

George Danezis and I also showed that where solidarity is needed for defence, smaller and more homogeneous groups will be more effective [511]. Rainer Böhme and Tyler Moore studied what happens where it isn't – if people use defense mechanisms that bring only private benefit, then the weakest-link model becomes one of low-hanging fruit. Examples include spammers who simply guess enough weak passwords to replenish their stock of compromised email accounts, and card-not-present fraud against e-commerce websites [276].

In short, the technology of conflict in any age can have deep and subtle effects on politics, as it conditions the kind of institutions that can survive and thrive. These institutions in turn shape the security landscape. Tyler Moore, Allan Friedman and Ariel Procaccia studied whether a national agency such as the NSA with both defensive and offensive missions would disclose vulnerabilities so they could be fixed, or stockpile them; they concluded that if it could ignore the social costs that fall on others, it would stockpile [1338]. However the biggest institutions in the security ecosystem are probably not government agencies but the dominant firms.

### **8.6.4 The economics of lock-in, tying and DRM**

Technical lock-in is one of the factors that lead to dominant-firm markets, and software firms have spent billions over more than thirty years on mechanisms that make it hard for their customers to leave but easy for their competitors to defect. The 1980s saw file format wars where companies tried to stop anyone else accessing the word-processing files or spreadsheets their software generated. By the 1990s, the fight had shifted to network compatibility as Microsoft tried to exclude other operating systems from LANs, until SAMBA created interoperability with Apple; in the wake of a 1993 anti-trust suit, Microsoft held back from using the Windows contract to block it. Adversarial interoperability emerged as a kind of judo to fight network effects [570]. Similar mechanisms are used to control markets in neighbouring or complementary goods and services, examples being tying ink cartridges to printers, and digital rights management (DRM) systems that lock music and videos to a specific machine or family of machines, by preventing users from simply copying them as files. In an early security-economics paper, Hal Varian pointed out in 2002 that their unfettered use could damage competition [1944].

In 2003, Microsoft, Intel and others launched a 'Trusted Computing' initiative that extended rights management to other types of file, and Windows Server 2003 offered 'Information Rights Management' (IRM) whereby I could email you a Word document that you could only read on screen, not print, and only till the end of the month. There was obvious potential for competitive abuse; by transferring control of user data from the owner of the machine on

which it is stored to the creator of the file in which it is stored, the potential for lock-in is hugely increased [73]. Think of the example in section 8.3.2 above, in which a firm has 100 staff, each with a PC on which they install Office for \$150. The \$15,000 they pay Microsoft is roughly equal to the total costs of switching to (say) LibreOffice, including training, converting files and so on. However, if control of the files moves to its thousands of customers, and the firm now has to contact each customer and request a digital certificate in order to migrate the file, then clearly the switching costs have increased – so you could expect the cost of Office to increase too. Now IRM failed to take off at the time: corporate America quickly understood that it was a lock-in play, European governments objected to the fact that the Trusted Computing initiative excluded small firms, and Microsoft couldn't get the mechanisms to work properly with Vista. However, now that email has moved to the cloud, both Microsoft and Google are offering restricted email services of just the type that was proposed, and objected to, back in 2003.

Another aspect concerns DRM and music. In the late 1990s and early 2000s, Hollywood and the music industry lobbied hard for mandatory DRM in consumer electronics equipment, and we still pay the costs of that in various ways; for example, when you switch your presentation from a VGA adapter to HDMI and you lose the audio. Hollywood's claim that unlicensed peer-to-peer file-sharing would destroy the creative industries was always shaky; a 2004 study showed that downloads didn't harm music industry revenues overall [1457] while a later one suggested that downloaders actually bought more CDs [50]. However the real issue was explained in 2005 by Google's chief economist [1946]: that a stronger link between the tech industry and music would help tech firms more than the music industry, because tech was more concentrated (with only three serious music platforms then – Microsoft, Sony and Apple). The content industry scoffed, but by the end of that year music publishers were protesting that Apple was getting too large a share of the cash from online music sales. Power in the supply chain moved from the music majors to the platforms, so the platforms (now Apple, Google, Amazon and Spotify) got most of the money and the residual power in the music industry shifted from the majors to the independents – just as airline deregulation favoured aircraft makers and low-cost airlines. This is a striking demonstration of the predictive power of economic analysis. By fighting a non-existent threat, the record industry helped the computer industry eat its lunch. I discuss this in more detail in section 24.5.

DRM had become much less of an issue by 2020; the move from removable media to streaming services means that few people copy music or movies any more; the question is whether you pay a subscription to avoid the ads. Similarly, the move to cloud-based services means that few people steal software. As a result, crimes involving copyright infringement have dropped sharply [91].

However, the move to the cloud is making lock-in a more complex matter, operating at the level of ecosystems as well as of individual products. We discussed above how competition from Google Docs cut the price of Office, and so Microsoft responded with a move to Office365; and how the total cost of ownership of either that service or G-suite is greater than a standalone productivity product. So where is the lock-in? Well, if you opt for the Google ecosystem, you'll probably be using not just Gmail and Google Docs but a Google calendar,

maps and much else. Although you can always download all your data, re-installing it on a different platform (such as Microsoft’s or Apple’s) will be a lot of bother, so you’ll probably just grit your teeth and pay for more storage when the free quota runs out. Similarly, if you start using tools like Slack or Splunk in an IT company, you’ll end up customising them in all sorts of ways that make it difficult to migrate. Again, this is nothing new; my own university’s dreadful accounting system has been a heavily customised version of Oracle Financials for about 20 years. Now everyone’s playing the lock-in game by inducing customers to buy or build complementary assets, or even to outsource whole functions. Salesforce has taken over many companies’ sales admin, Palantir has locked in many US police forces, and the big academic publishers are usurping the functions of university libraries. Where there’s no viable competition – as in the second of these cases – there’s a real policy issue. The depth of Microsoft lock-in on public-sector IT is illustrated by the brave attempts made by the city of Munich to break away and use Linux in public administration: this was eventually reverted after 15 years, several visits of Bill Gates, and a new mayor [759].

The control of whole ecosystems by cartels is nothing new; Joshua Specht tells the history of how the big food companies like Cargill and Armour grabbed control of the two-sided markets opened up by the railroads, consolidated their power by buying infrastructure such as grain elevators, dumped climate risk on small farmers, ran union organisers out of town and even got the politicians to pass ‘ag-gag’ laws that define animal-rights activism as terrorism [1808]. There are interesting echoes of this in the way the big IT service firms have built out their market power, controlling everything from the ad ecosystem through operating systems to datacentres. In fact, the whole global economy has become more monopolistic over the past couple of decades, and IT appears to account for much of the growth in industry concentration[234]. It isn’t the only factor – other industries (such as defence contracting) have their own dynamic, while the regulators of natural monopolies such as utilities tend to be captured over time by lobbying. There is a growing literature on *moats* – structural barriers to competition, of which network effects and technical lock-in are merely two examples; others range from patents and regulatory capture to customer insight derived from control of data [1431]. The dynamics of the information industries compound many of these existing problems and can make effective competition even harder. Competition law scholars, led by Lina Khan of Harvard, have been arguing for several years that American law needs to take a broader view of competition abuse than just consumer surplus (as is already the case in Europe) [1044], while Chicago-school economists such as Carl Shapiro denounce antitrust populism and argue that remedies should be targeted at specific harms, as antitrust law is ill-suited to tackle the political power that large corporations wield [1716]. Carl does however concede that US antitrust law has been excessively narrowed by the Supreme Court in the last 40 years; that the consumer-welfare test is inadequate; that dominant firms’ exclusionary conduct and labour-market practices both need to be tackled, and that the USA needs to control horizontal mergers better [1717].

European competition law has for many years forbidden firms from using a dominant position in one market to establish one in another, and we’ve seen a whole series of judgements against the big tech firms. As for the likely future direction, a 2019 report for the European Commission’s Directorate-General

of Competition by Jacques Crémer, Yves-Alexandre de Montjoye and Heike Schweitzer highlights not just the tech majors' network externalities and extreme returns to scale, but also the fact that they control more and more of the data thanks to the move to online services and cloud computing [497]. As a result they have economies of scope: succeeding in one business makes it easier to succeed in another. It concludes that the EU's competition-law framework is basically sound but needs some tuning: regulators need to protect both competition for the market and competition in the market, such as on dominant platforms, which have a responsibility not to distort competition there. In this environment, regulators must pay attention to multihoming, switching, interoperability, data portability and the effect on aftermarkets.

Tying spare parts is also regulated in Europe, with specific laws in some sectors requiring vendors to let other firms make compatible spare parts, and in others requiring that they make spares available for a certain period of time. Some very specific policy issues can arise if you use security mechanisms to tie products to each other. This links in with laws on planned obsolescence, which is reinforced for goods with digital components when the vendors limit the time period for which software updates are made available. The rules have recently been upgraded in the European Union by a new Sales of Goods Directive (2019/771) that from January 2022 requires firms selling goods with digital components – whether embedded software, cloud services or associated phone apps – to maintain this software for at least two years after the good are sold, and for longer if this is the reasonable expectation of the customer (for cars and white goods it's likely to mean ten years). Such regulations will become more of an issue now we have software in durable goods such as cars and medical devices; I'll discuss sustainability in the last chapter of this book.

### **8.6.5 Perversely motivated guards**

“There’s nae sae blind as them that will na see”, goes an old Scots proverb, and security engineering throws up lots of examples.

- There’s very little police action against cybercrime, as they found it simpler to deter people from reporting it. As we noted in section 2.3, this enabled them to claim that crime was falling for many years even though it was just moving online like everything else.
- Governments have imposed a duty on banks to spot money laundering, especially since 9/11. However no banker really wants to know that one of his customers is a Mafioso. So banks lobby for risk reduction to be formalised as due diligence; they press for detailed regulations that specify the forms of ID they need for new account opening, and the processing to be done to identify suspicious transactions.
- When it comes to fraud, spotting a rare bank fraud pattern means a payment service provider should now carry the loss rather than just telling the customer she must be mistaken or lying. So they’re tempted to wait and learn about new fraud types from industry or from academics, rather than doing serious research of their own.

- Click fraud is similar. Spotting a pattern of ‘inorganic clicks’ from a botnet means you can’t charge the advertisers for those clicks any more. You have to do some work to mitigate the worst of it, but if you have a dominant market position then the harder you work at fighting click fraud, the less revenue you earn.
- Finding bugs in your own code is another example. Of course you have to tweak the obvious bugs that stop it working, but what about the more subtle bugs that can be exploited by attackers? The more time you spend looking for them, the more time you have to spend fixing them. You can always go and buy static analysis tools, but then you’ll find thousands more bugs and your ship date will slip by months. So firms tend to do that only if their customers demand it, and it’s only cheap if you do it from the start of a project (but in that case you could just as well write the code in Rust rather than in C).

There are more subtle examples, such as when it’s not politically acceptable to tell the truth about threats. In the old days, it was hard to talk to a board of directors about the insider threat, as directors mostly preferred to believe the best about their company; so a typical security manager would make chilling presentations about ‘evil hackers’ in order to get the budget to build internal controls. Nowadays, the security-policy space in many companies has been captured by the big four accountancy firms, whose consensus on internal controls is tied to their thought leadership on governance, which a cynic might say is optimised for the welfare not of their ostensible client, the shareholders, but for their real client, the CEO. Executive frauds are rarely spotted unless they bring the company down; the effort goes instead into the annoying and irrelevant, such as changing passwords every month and insisting on original paper receipts. I discuss all this in detail in section 12.2.2.

Or consider the 2009 parliamentary expenses scandal in the UK described in section 2.3.6. Perhaps the officers of the Houses of Parliament didn’t defend the expenses system more vigorously because they have to think of MPs and peers as ‘honourable members’ in the context of a government that was pushing harsh surveillance legislation with a slogan of ‘If you’ve nothing to hide you have nothing to fear’. The author of that slogan, then Home Secretary Jacqui Smith, may have had nothing to hide, but her husband did: he was watching porn and charging it to her parliamentary expenses. Jacqui lost her job, and her seat in Parliament too. Had officers known that the information on the expenses server could cost a cabinet minister her job, they probably ought to have classified it Top Secret and kept it in a vault. But how could the extra costs have been justified to the Treasury? On that cheerful note, let’s go on to privacy.

### **8.6.6 Economics of privacy**

The privacy paradox is that people say that they value privacy, yet act otherwise. If you stop people in the street and ask them their views, about a third say they are privacy fundamentalists and will never hand over their personal information to marketers or anyone else; about a third say they don’t care; and about a third are in the middle, saying they’d take a pragmatic view of the risks and benefits

of any disclosure. However, their shopping behavior – both online and offline – is quite different; the great majority of people pay little heed to privacy, and will give away the most sensitive information for little benefit. Privacy-enhancing technologies have been offered for sale by various firms, yet most have failed in the marketplace. Why should this be?

Privacy is one aspect of information security that interested economists before 2000. In 1978, Richard Posner defined privacy in terms of secrecy [1536], and the following year extended it to seclusion [1537]. In 1980, Jack Hirshleifer published a seminal paper in which he argued that rather than being about withdrawing from society, privacy was a means of organising society, arising from evolved territorial behavior; internalised respect for property supports autonomy. In 1996, Hal Varian analysed privacy in terms of information markets [1940]. Consumers want to not be annoyed by irrelevant marketing calls while marketers do not want to waste effort; yet both are frustrated, because of search costs, externalities and other factors. Varian suggested giving consumers rights in information about themselves, and letting contracts sort it out.

However, as we've seen, the information industries are prone to market failures leading to monopoly, and the proliferation of dominant, information-intensive business models demands a different approach. Andrew Odlyzko argued in 2003 that these monopolies simultaneously increase both the incentives and the opportunities for price discrimination [1462]. Companies mine online interactions for data revealing individuals' willingness to pay, and while the differential pricing we see in many markets from airline yield-management systems to telecommunications prices may be economically efficient, it is increasingly resented. Peter Swire argued that we should measure the externalities of privacy intrusion [1852]. If a telesales operator calls 100 prospects, sells three of them insurance, and annoys 80, then the conventional economic analysis considers only the benefit to the three and to the insurer. But persistent annoyance causes millions of people to go ex-directory, screen calls through an answering machine, or just not have a landline at all. The long-run societal costs of robocalls can be considerable. Empirical studies of people's privacy valuations have supported this.

The privacy paradox has generated a significant literature, and is compounded by at least three factors. First, there are many different types of privacy harm, from discrimination in employment, credit and insurance, through the kind of cybercrime that presents as payment fraud, to personal crimes such as stalking and non-consensual intimate imagery.

Second, the behavioral factors we discussed in section 3.2.5 play a large role. Leslie John and colleagues demonstrated the power of context with a neat experiment. She devised a 'privacy meter' in the form of a list of embarrassing questions; the score was how many questions a subject would answer before they balked. She tried this on three groups of students: a control group in a neutral university setting, a privacy treatment group who were given strong assurances that their data would be encrypted, their IP addresses not stored, and so on; and a gamer treatment group that was taken to an external website ([howbadareyou.com](http://howbadareyou.com) with a logo of a smiling devil). You might think that the privacy treatment group would disclose more, but in fact they disclosed less – as privacy had been made salient to them. As for the gamer group, they happily

disclosed twice as much as the control group [987].

Third, the industry understands this, and goes out of its way to make privacy risks less salient. Privacy policies are usually not on the front page, but are easily findable by concerned users; policies typically start with anodyne text and leave the unpleasant stuff to the end, so they don't alarm the casual viewer, but the vigilant minority can quickly find a reason not to use the site, so they also don't stop the other users clicking on the ads. The cookie warnings mandated in Europe are mostly anodyne, though some firms give users fine-grained control; as noted in section 3.2.5, the illusion of control is enough to reassure many.

So what's the overall effect? In the 2000s and early 2010s there was evidence that the public were gradually learning what we engineers already understood about the risks; we could see this for example in the steadily rising proportion of Facebook users who opt to use privacy controls to narrow that system's very open defaults.

In 2015, almost two years after the Snowden revelations, two surveys conducted by Pew Research disclosed a growing sense of learned helplessness among the US public. 93% of adults said that being in control of who can get information about them is important, and 90% that controlling what information is collected about them is important; 88% said it's important that no-one watch or listen to them without their permission. Yet just 6% of adults said they were 'very confident' that government agencies could keep their records private and secure, while another 25% said they were 'somewhat confident.' The figures for phone companies and credit card companies were similar while those for advertisers, social media and search engines were significantly worse. Yet few respondents had done anything significant, beyond occasionally clearing their browser history or refusing particularly inappropriate demands for personal information [1204].

These tensions have been growing since the 1960s, and have led to complex privacy regulation that differs significantly between the US and Europe. I'll discuss this in much more detail in section 26.6.

### **8.6.7 Organisations and human behaviour**

Organisations often act in apparently irrational ways. We frequently see firms and even governments becoming so complacent that they're unable to react to a threat until it's a crisis, when they panic. The erosion of health service resilience and pandemic preparedness in Europe and North America in the century since the 1918–19 Spanish flu is merely the most salient of many examples. As another example, it seems that there's always one phone company, and one bank, that the bad guys are picking on. A low rate of fraud makes people complacent, until the bad guys notice. The rising tide of abuse is ignored, or blamed on customers, for as long as possible. Then it gets in the news and executives panic. Loads of money get spent for a year or two, stuff gets fixed, and the bad guys move on to the next victim.

So the security engineer needs to anticipate the ways in which human frailties express themselves through organizational behaviour.

There's a substantial literature on institutional economics going back to Thorstein Veblen. One distinguished practitioner, Herb Simon, was also a computing pioneer and founded computer science at CMU. In a classic book on administrative behaviour, he explained that the decisions taken by managers are not just about efficiency but also organisational loyalty and authority, and the interaction between the organisation's goals and the incentives facing individual employees; there are messy hierarchies of purpose, while values and facts are mixed up [1754]. A more modern analysis of these problems typically sees them as principal-agency issues in the framework of microeconomics; this is a typical approach of professors of accountancy. We will discuss the failures of the actual practice of accountancy later, in section 12.2. Another approach is public-choice economics, which applies microeconomic methods to study the behaviour of politicians, civil servants and people in public-sector organisations generally. I summarise public choice in section 26.3.3; the principles are illustrated well in the TV sitcom "Yes Minister" which explores the behaviour of British civil servants. Cynics note that bureaucracies seem to evolve in such a way as to minimise the likelihood of blame.

My own observation, having worked in banks, tech companies big and small and in the university sector too, is that competition is more important than whether an enterprise is publicly or privately owned. University professors compete hard with each other; our customer isn't our Vice-Chancellor but the Nobel Prize committee or equivalent. But as university administrators work in a hierarchy with the VC at the top, they face the same incentives as civil servants and display many of the same strengths and weaknesses. Meanwhile, some private firms have such market power that internally they behave just like government (though with much better pay at the top).

### **8.6.8 Economics of cybercrime**

If you're going to protect systems from attack, it's a good idea to know who the attackers are, how many they are, where they come from, how they learn their jobs and how they're motivated. This brings us to the economics of cybercrime. In section 2.3 we gave an overview of the cybercrime ecosystem, and there are many tools we can use to study it in more detail. At the Cambridge Cybercrime Centre we collect and curate the data needed to do this, and make it available to over a hundred researchers worldwide. As in other economic disciplines, there's an iterative process of working out what the interesting questions are and collecting the data to answer them. The people with the questions are not just economists but engineers, psychologists, lawyers, law enforcement and, increasingly, criminologists.

One approach to crime is that of Chicago-school economists such as Gary Becker, who in 1968 analysed crime in terms of rewards and punishments [200]. This approach gives many valuable insights but isn't the whole story. Why is crime clustered in bad neighbourhoods? Why do some kids from these neighbourhoods become prolific and persistent offenders? Traditional criminologists study questions like these, and find explanations of value in crime prevention: the worst offenders often suffer multiple deprivation, with poor parenting, with substance and alcohol abuse, and get drawn into cycles of offending. The earlier

they start in their teens, the longer they'll persist before they give up. Critical criminologists point out that laws are made by the powerful, who maintain their power by oppressing the poor, and that bad neighbourhoods are more likely to be over-policed and stigmatised than the nice suburbs where the rich white people live.

Drilling down further, we can look at the bad neighbourhoods, the psychology of offenders, and the pathways they take into crime. Since the 1960s there has been a substantial amount of research into using environmental design to suppress crime, initially in low-cost housing and then everywhere. For example, courtyards are better than parks, as residents are more likely to identify and challenge intruders; many of these ideas for *situational crime prevention* go across from criminology into systems design. In section 13.2.2 we'll discuss this in more detail.

Second, psychologically normal people don't like harming others; people who do so tend to have low empathy, perhaps because of childhood abuse, or (more often) to have minimisation strategies to justify their actions. Bank robbers see bankers as the real exploiters; soldiers dehumanise the enemy as 'gooks' or 'terrs'; and most common murderers see their crimes as a matter of honour. "She cheated on me" and "He disrespected me" are typical triggers; we discussed the mechanisms in section 3.2.4. These mechanisms go across to the world of online and electronic fraud. Hackers on the wrong side of the law tend to feel their actions are justified anyway: hacktivists are political activists after all, while cyber-crooks use a variety of minimisation strategies to avoid feeling guilty. Some Russian cybercrooks take the view that the USA screwed Russian over after 1989, so they're just getting their own back (and they're supported in this by their own government's attitudes and policies). As for bankers who dump fraud risks on customers, they talk internally about 'the avalanche of fraudulent risks of fraud' they'd face if they owned up to security holes.

Third, it's important to understand the pathways to crime, the organisation of criminal gangs, and the diffusion of skills. Steve Levitt studied the organisation and finances of Chicago crime gangs, finding that the street-level dealers were earning less than minimum wage [1151]. They were prepared to stand in the rain and be shot at for a chance to make it to the next level up, where the neighbourhood boss drove around in a BMW with three girls. Arresting the boss won't make any difference as there are dozens of youngsters who'll fight to replace him. To get a result, the police should target the choke point, such as the importer's system administrator. These ideas also go across. Many cyber-criminals start off as gamers, then cheat on games, then deal in game cheats, then learn how to code game cheats, and within a few years the more talented have become malware devs. So one policy intervention is to try to stop kids crossing the line between legal and illegal game cheating. As I mentioned in section 3.2.4, the UK National Crime Agency bought Google ads which warned people in Britain searching for DDoS-for-hire services that the use of such services was illegal. Ben Collier and colleagues used our Cybercrime Centre data to show that this halted the growth of DDoS attacks in the UK, compared with the USA where they continued to grow [454].

We discussed the overall costs of cybercrime in section 2.3, noting that the ecosystem has been remarkably stable over the past decade, despite the fact that

the technology has changed; we now go online from phones more than laptops, use social networks, and keep everything in the cloud. Most acquisitive crime is now online; in 2019 we expect that about a million UK households suffered a burglary or car theft, while over two million suffered a fraud or scam, almost always online. (In 2020 the difference will be even more pronounced; burglary has fallen still further with people staying at home through the lockdown.) Yet policy responses lag almost everywhere. Studies of specific crimes are reported at various places in this book.

The effects of cybercrime are also studied via the effects of breach disclosures. Alessandro Acquisti and colleagues have studied the effects on the stock price of companies of reporting a security or privacy breach [15]; a single breach tends to cause a small dip that dissipates after a week or so, but a double breach can impair investor confidence over the longer term. Breach disclosure laws have made breaches into insurable events; if TJX loses 47m records and has to pay \$5 to mail each customer, that's a claim; we'll discuss cyber-insurance later in section 28.2.9.

Overall, though, measurement is tricky. Most of the relevant publications come from organisations with an incentive to talk up the losses, from police agencies to anti-virus vendors; our preferred methodology is to count the losses by modus operandi and by sector, as presented in section 2.3.

## 8.7 Summary

Many systems fail because the incentives are wrong, rather than because of some technical design mistake. As a result, the security engineer needs to understand basic economics as well as the basics of crypto, protocols, access controls and psychology. Security economics has grown rapidly to explain many of the things that we used to consider just ‘bad weather’. It constantly throws up fascinating new insights into all sorts of questions from how to optimise the patching cycle through whether people really care about privacy.

## Research problems

So far, three areas of economics have been explored for their relevance to security, namely microeconomics, game theory and behavioural economics. But economics is a vast subject. What other ideas might it give us?

In the history paper I wrote on the origins of security economics, I suggested a new research student might follow the following heuristics to select a research topic. First, think of security and  $X$  for other subfields  $X$  of economics. Second, think about the security economics of  $Y$  for different applications  $Y$ ; there have already been some papers on topics like payments, pornography, gaming, and censorship, but these aren't the only things computers are used for. Third, where you find gold, keep digging (e.g. behavioral privacy) [78]. Since then I would add the following.

Fourth, there is a lot of scope for data-driven research now that we're starting

## 8.7. SUMMARY

---

to make large datasets available to academics (via the Cambridge Cybercrime Centre) and many students are keen to develop skills in data science. A related problem is how to gather more data that might be useful in exploring other fields, from the productivity of individual security staff to how security works within institutions, particularly large complex institutions such as governments and healthcare systems. Is there any good way of measuring the quality of a security culture? Fifth, now we're starting to put software and online connectivity in durable safety-critical things like cars and medical devices, we need to know a lot more about the interaction between security and safety, and about how we can keep such systems patched and running for decades. This opens up all sorts of new topics in dependability and sustainability.

The current research in security economics is published mostly at the Workshop on the Economics of Information Security (WEIS), which has been held annually since 2002 [76]. There are liveblogs of all but one of the workshops, consisting of a summary of each paper and a link to it, which you can get on my blog or linked directly from my Economics and Security Resource Page at <http://www.cl.cam.ac.uk/~rja14/econsec.html>.

## Further reading

The classic introduction to information economics is Shapiro and Varian's '*Information Rules*' which remains remarkably fresh for a book written twenty years ago [1718]. This is still on our student reading list. The most up-to-date summary is probably Jacques Crémer, Yves-Alexandre de Montjoye and Heike Schweitzer's 2019 report for the European Commission's Directorate-General of Competition, which analyses what goes wrong with markets in which information plays a significant role [497]; I would read also Carl Shapiro's 2019 review of the state of competition policy in the USA[1717].

Tim Wu's "The Master Switch" discusses monopoly in telecomms and the information industries generally from the viewpoint of ten years ago [2049]. If you plan to do research in the subject and your degree wasn't in economics, you might work through a standard textbook such as Varian [1941] or the Core Economics website. Adam Smith's classic '*An inquiry into the nature and causes of the wealth of nations*' is still worth a look, while Dick Thaler's '*Misbehaving*' tells the story of behavioural economics.

The early story of security economics is told in [78]; there's an early (2007) survey of the field that I wrote with Tyler Moore at [110], and a more comprehensive 2011 survey, also with Tyler, at [111]. For privacy economics, see Alessandro Acquisti's online bibliography, and the survey paper he wrote with George Loewenstein and Laura Brandimarte [16]; there's also a survey of the literature on the privacy paradox by Spiros Kokolakis [1076]. Then, to dive into the research literature, I'd suggest the WEIS conference papers and liveblogs.

A number of economists study related areas. I mentioned Jack Hirshleifer's conflict theory [907]; another important strand is the economics of crime, which was kick-started by Gary Becker [200], and has been popularised by Steve Levitt and Stephen Dubner's "Freakonomics" [1151]. Diego Gambetta is probably the

## **8.7. SUMMARY**

---

leading scholar of organised crime; his '*Codes of the Underworld: How Criminals Communicate*' is a classic [742]. Finally, there is a growing research community and literature on cyber-criminology, for which the website of our Cambridge Cybercrime Centre might be a reasonable starting point.

# Chapter 9

## Multilevel security

Most high assurance work has been done in the area of kinetic devices and infernal machines that are controlled by stupid robots. As information processing technology becomes more important to society, these concerns spread to areas previously thought inherently harmless, like operating systems.  
— EARL BOEBERT

The password on the government phone always seemed to drop, and I couldn't get into it  
– US diplomat and former CIA officer KURT VOLKER, explaining why he texted from his personal phone

I brief;  
you leak;  
he/she commits a criminal offence  
by divulging classified information.  
— BRITISH CIVIL SERVICE VERB

### 9.1 Introduction

In the next few chapters I'm going to explore the concept of a security policy using case studies. A security policy is a succinct description of what we're trying to achieve; it's driven by an understanding of the bad outcomes we wish to avoid and in turn drives the engineering. After I've fleshed out these ideas a little, I'll spend the rest of this chapter exploring the *multilevel security* (MLS) policy model used in many military and intelligence systems, which hold information at different levels of classification (Confidential, Secret, Top Secret, ...), and have to ensure that data can be read only by a principal whose clearance level is at least as high. Such policies are increasingly also known as *information flow control* (IFC).

## **9.2. WHAT IS A SECURITY POLICY MODEL?**

---

They are important for a number of reasons, even if you're never planning to work for a government contractor:

1. from about 1980 to about 2005, the US Department of Defense spent several billion dollars funding research into multilevel security. So the model was worked out in great detail, and we got to understand the second-order effects of pursuing a single policy goal with great zeal;
2. the *mandatory access control* (MAC) systems used to implement it have now appeared in all major operating systems such as Android, iOS and Windows to protect core components against tampering by malware, as I described in chapter 6;
3. although multilevel security concepts were originally developed to support confidentiality in military systems, many commercial systems now use multilevel integrity policies. For example, safety-critical systems use a number of safety integrity levels<sup>1</sup>.

The poet Archilochus famously noted that a fox knows many little things, while a hedgehog knows one big thing. Security engineering is usually in fox territory, but multilevel security is an example of the hedgehog approach.

## **9.2 What is a Security Policy Model?**

Where a top-down approach to security engineering is possible, it will typically take the form of *threat model – security policy – security mechanisms*. The critical, and often neglected, part of this process is the security policy.

By a security policy, we mean a document that expresses clearly and concisely what the protection mechanisms are to achieve. It is driven by our understanding of threats, and in turn drives our system design. It will often take the form of statements about which users may access which data. It plays the same role in specifying the system's protection requirements, and evaluating whether they have been met, that the system specification does for functionality and the safety case for safety. Like the specification, its primary function is to communicate.

Many organizations use the phrase ‘security policy’ to mean a collection of vapid statements, as in Figure 9.1:

---

<sup>1</sup>Beware though that terminology varies between different safety-engineering disciplines. The safety integrity levels in electricity generation are similar to Biba, while automotive safety integrity levels are set in ISO 26262 as a hazard/risk metric that depends on the likelihood that a fault will cause an accident, together with the expected severity and controllability

## 9.2. WHAT IS A SECURITY POLICY MODEL?

---

### Megacorp Inc security policy

1. This policy is approved by Management.
2. All staff shall obey this security policy.
3. Data shall be available only to those with a “need-to-know”.
4. All breaches of this policy shall be reported at once to Security.

Figure 9.1 – typical corporate policy language

This sort of language is common, but useless – at least to the security engineer. It dodges the central issue, namely ‘Who determines “need-to-know” and how?’ Second, it mixes statements at different levels (organizational approval of a policy should logically not be part of the policy itself). Third, there is a mechanism but it’s implied rather than explicit: ‘staff shall obey’ – but what does this mean they actually have to do? Must the obedience be enforced by the system, or are users ‘on their honour’? Fourth, how are breaches to be detected and who has a specific duty to report them?

When you think about it, this is political language. A politician’s job is to resolve the tensions in society, and this often requires vague language on which different factions can project their own wishes; corporate executives are often operating politically, to balance different factions within a company<sup>2</sup>.

Because the term ‘security policy’ is often abused to mean using security for politics, more precise terms have come into use by security engineers.

A *security policy model* is a succinct statement of the protection properties that a system must have. Its key points can typically be written down in a page or less. It is the document in which the protection goals of the system are agreed with an entire community, or with the top management of a customer. It may also be the basis of formal mathematical analysis.

A *security target* is a more detailed description of the protection mechanisms that a specific implementation provides, and how they relate to a list of control objectives (some but not all of which are typically derived from the policy model). The security target forms the basis for testing and evaluation of a product.

A *protection profile* is like a security target but expressed in an implementation-independent way to enable comparable evaluations across products and versions. This can involve the use of a semi-formal language, or at least of suitable security jargon. A protection profile is a requirement for products that are to be evaluated under the *Common Criteria* [1396]. (I discuss the Common Criteria in Part III; they are used by many governments for mutual recognition of security evaluations of defense information systems.)

When I don’t have to be so precise, I may use the phrase ‘security policy’ to refer to either a security policy model or a security target. I will never use it to refer to a collection of platitudes.

---

<sup>2</sup>Big projects often fail in companies when the specification becomes political, and they fail even more often when run by governments – issues I’ll discuss further in Part 3.

### 9.3. MULTILEVEL SECURITY POLICY

---

Sometimes, we're confronted with a completely new application and have to design a security policy model from scratch. More commonly, there already exists a model; we just have to choose the right one, and develop it into a security target. Neither of these steps is easy. In this section of the book, I provide a number of security policy models, describe them in the context of real systems, and examine the engineering mechanisms (and associated constraints) which a security target can use to meet them.

## 9.3 Multilevel Security Policy

On March 22, 1940, President Roosevelt signed Executive Order 8381, enabling certain types of information to be classified Restricted, Confidential or Secret [978]. President Truman later added a higher level of Top Secret. This developed into a common protective marking scheme for the sensitivity of documents, and was adopted by NATO governments too in the Cold War. *Classifications* are labels, which run upwards from *Unclassified* through *Confidential*, *Secret* and *Top Secret* (see Figure 9.2). The original idea was that information whose compromise could cost lives was marked ‘Secret’ while information whose compromise could cost many lives was ‘Top Secret’. Government employees and contractors have *clearances* depending on the care with which they’ve been vetted; in the USA, for example, a ‘Secret’ clearance involves checking FBI fingerprint files, while ‘Top Secret’ also involves background checks for the previous five to fifteen years’ employment plus an interview and often a polygraph test [548]. Candidates have to disclose all their sexual partners in recent years and all material that might be used to blackmail them, such as teenage drug use or gay affairs<sup>3</sup>.

The access control policy was simple: you can read a document only if your clearance is at least as high as the document’s classification. So an official cleared to ‘Top Secret’ could read a ‘Secret’ document, but not vice versa. So information may only flow upwards, from confidential to secret to top secret, but never downwards – unless an authorized person takes a deliberate decision to declassify it.

TOP SECRET
SECRET
CONFIDENTIAL
UNCLASSIFIED

Figure 9.2 – multilevel security

The system rapidly became more complicated. The damage criteria for classifying documents were expanded from possible military consequences to

<sup>3</sup>In June 2015, the clearance review data of about 20m Americans was stolen from the Office of Personnel Management by the Chinese intelligence services. By then, about a million Americans had a Top Secret clearance; the OPM data also covered former employees and job applicants, as well as their relatives and sexual partners. With hindsight, collecting all the dirt on all the citizens with a sensitive job may not have been a great idea.

### 9.3. MULTILEVEL SECURITY POLICY

---

economic harm and even political embarrassment. Information that is neither classified nor public is known as ‘Controlled Unclassified Information’ (CUI) in the USA while Britain uses ‘Official’<sup>4</sup>.

There is also a system of codewords whereby information, especially at Secret and above, can be restricted further. For example, information that might reveal intelligence sources or methods – such as the identities of agents or decryption capabilities – is typically classified ‘Top Secret Special Compartmented Intelligence’ or TS/SCI, which means that so-called *need to know* restrictions are imposed as well, with one or more codewords attached to a file. Some codewords relate to a particular military operation or intelligence source and are available only to a group of named users. To read a document, a user must have all the codewords that are attached to it. A classification label, plus a set of codewords, makes up a *security category* or (if there’s at least one codeword) a *compartment*, which is a set of records with the same access control policy. Compartmentation is typically implemented nowadays using discretionary access control mechanisms; I’ll discuss it in the next chapter.

There are also *descriptors*, *caveats* and *IDO markings*. Descriptors are words such as ‘Management’, ‘Budget’, and ‘Appointments’: they do not invoke any special handling requirements, so we can deal with a file marked ‘Confidential – Management’ as if it were simply marked ‘Confidential’. Caveats are warnings such as “UK Eyes Only”, or the US equivalent, “NOFORN”; they do create restrictions. There are also *International Defence Organisation* markings such as *NATO*<sup>5</sup>. The lack of obvious differences between codewords, descriptors, caveats and IDO marking helps make the system confusing. A more detailed explanation can be found in [1562].

#### 9.3.1 The Anderson report

In the 1960s, when computers started being widely used, the classification system caused serious friction. Paul Karger, who worked for the USAF then, described having to log off from a Confidential system, walk across the yard to a different hut, show a pass to an armed guard, then go in and log on to a Secret system – over a dozen times a day. People soon realised they needed a way to deal with information at different levels at the same desk, but how could this be done without secrets leaking? As soon as one operating system bug was fixed, some other vulnerability would be discovered. The NSA hired an eminent computer scientist, Willis Ware, to its scientific advisory board, and in 1967 he brought the extent of the computer security problem to official and public attention [1985]. There was the constant worry that even unskilled users would

<sup>4</sup>Prior to adopting the CUI system, the United States had more than 50 different markings for data that was controlled but not classified, including For Official Use Only (FOUO), Law Enforcement Sensitive (LES), Proprietary (PROPIN), Federal Tax Information (FTI), Sensitive but Unclassified (SBU), and many, many others. Some agencies made up their own labels, without any coordination. Further problems arose when civilian documents marked Confidential ended up at the National Archives and Records Administration, where CONFIDENTIAL was a national security classification. Moving from this menagerie of markings to a single centrally-managed government-wide system has taken more than a decade and is still ongoing. The UK has its own post-Cold-War simplification story.

<sup>5</sup>Curiously, in the UK ‘NATO Secret’ is less secret than ‘Secret’, so it’s a kind of anti-codeword that moves the content down the lattice rather than up.

### **9.3. MULTILEVEL SECURITY POLICY**

---

discover loopholes and use them opportunistically; there was also a keen and growing awareness of the threat from malicious code. (Viruses were not invented until the 1980s; the 70's concern was Trojans.) There was then a serious scare when it was discovered that the Pentagon's World Wide Military Command and Control System (WWMCCS) was vulnerable to Trojan Horse attacks; this had the effect of restricting its use to people with a 'Top Secret' clearance, which was inconvenient.

The next step was a 1972 study by James Anderson for the US government which concluded that a secure system should do one or two things well; and that these protection properties should be enforced by mechanisms which were simple enough to verify and that would change only rarely [51]. It introduced the concept of a *reference monitor* – a component of the operating system which would mediate access control decisions and be small enough to be subject to analysis and tests, the completeness of which could be assured. In modern parlance, such components – together with their associated operating procedures – make up the *Trusted Computing Base* (TCB). More formally, the TCB is defined as the set of components (hardware, software, human, ...) whose correct functioning is sufficient to ensure that the security policy is enforced, or, more vividly, whose failure could cause a breach of the security policy. The Anderson report's goal was to make the security policy simple enough for the TCB to be amenable to careful verification.

#### **9.3.2 The Bell-LaPadula model**

The multilevel security policy model that gained wide acceptance was proposed by Dave Bell and Len LaPadula in 1973 [210]. Its basic property is that information cannot flow downwards. More formally, the *Bell-LaPadula* (BLP) model enforces two properties:

- The *simple security property*: no process may read data at a higher level. This is also known as *no read up* (*NRU*);
- The *\*-property*: no process may write data to a lower level. This is also known as *no write down* (*NWD*).

The *\**-property was Bell and LaPadula's critical innovation. It was driven by the WWMCCS debacle and the more general fear of Trojan-horse attacks. An un cleared user might write a Trojan and leave it around where a system administrator cleared to 'Secret' might execute it; it could then copy itself into the 'Secret' part of the system, read the data there and try to signal it down somehow. It's also quite possible that an enemy agent could get a job at a commercial software house and embed some code in a product that would look for secret documents to copy. If it could then write them down to where its creator could read them, the security policy would have been violated. Information might also be leaked as a result of a bug, if applications could write down.

Vulnerabilities such as malicious and buggy code are assumed to be given. It is also assumed that most staff are careless, and some are dishonest; extensive operational security measures have long been used, especially in defence

### **9.3. MULTILEVEL SECURITY POLICY**

---

environments, to prevent people leaking paper documents. So the pre-existing culture assumed that security policy was enforced independently of user actions; Bell-LaPadula sets out to enforce it not just independently of users' direct actions, but of their indirect actions (such as the actions taken by programs they run).

So we must prevent programs running at 'Secret' from writing to files at 'Unclassified'. More generally we must prevent any process at High from signalling to any object at Low. Systems that enforce a security policy independently of user actions are described as having *mandatory access control*, as opposed to the *discretionary access control* in systems like Unix where users can take their own access decisions about their files.

The Bell-LaPadula model enabled designers to prove theorems. Given both the simple security property (no read up), and the star property (no write down), various results can be proved: in particular, if your starting state is secure, then your system will remain so. To keep things simple, we will generally assume from now on that the system has only two levels, High and Low.

#### **9.3.3 The standard criticisms of Bell-LaPadula**

The introduction of BLP caused a lot of excitement: here was a security policy that did what the defence establishment thought it wanted, was intuitively clear, yet still allowed people to prove theorems. Researchers started to beat up on it and refine it.

The first big controversy was about John McLean's *System Z*, which he defined as a BLP system with the added feature that a user can ask the system administrator to temporarily declassify any file from High to Low. In this way, Low users can read any High file without breaking the BLP assumptions. Dave Bell countered that System Z cheats by doing something his model doesn't allow (changing labels isn't a valid operation on the state), and John McLean's retort was that it didn't explicitly tell him so: so the BLP rules were not in themselves enough. The issue is dealt with by introducing a *tranquility property*. Strong tranquility says that security labels never change during system operation, while weak tranquility says that labels never change in such a way as to violate a defined security policy.

Why weak tranquility? In a real system we often want to observe the principle of least privilege and start off a process at the uncleared level, even if the owner of the process were cleared to 'Top Secret'. If they then access a confidential email, their session is automatically upgraded to 'Confidential'; in general, a process is upgraded each time it accesses data at a higher level (the *high water mark* principle). As subjects are usually an abstraction of the memory management sub-system and file handles, rather than processes, this means that state changes when access rights change, rather than when data actually moves.

The practical implication is that a process acquires the security labels of all the files it reads, and these become the default label set of every file that it writes. So a process which has read files at 'Secret' and 'Crypto' will thereafter create files marked 'Secret Crypto'. This will include temporary copies made of

### **9.3. MULTILEVEL SECURITY POLICY**

---

other files. If it then reads a file at ‘Secret Nuclear’ then all files it creates after that will be labelled ‘Secret Crypto Nuclear’, and it will not be able to write to any temporary files at ‘Secret Crypto’.

The effect this has on applications is one of the serious complexities of multilevel security; most application software needs to be rewritten (or at least modified) to run on MLS platforms. Real-time changes in security level mean that access to resources can be revoked at any time, including in the middle of a transaction. And as the revocation problem is generally unsolvable in modern operating systems, at least in any complete form, the applications have to cope somehow. Unless you invest some care and effort, you can easily find that everything ends up in the highest compartment – or that the system fragments into thousands of tiny compartments that don’t communicate at all with each other. In order to prevent this, labels are now generally taken outside the MLS machinery and dealt with using discretionary access control mechanisms (I’ll discuss this in the next chapter).

Another problem with BLP, and indeed with all mandatory access control systems, is that separating users and processes is the easy part; the hard part is when some controlled interaction is needed. Most real applications need some kind of *trusted subject* that can break the security policy; the classic example was a trusted word processor that helps an intelligence analyst scrub a Top Secret document when she’s editing it down to Secret [1270]. BLP is silent on how the system should protect such an application. So it becomes part of the Trusted Computing Base, but a part that can’t be verified using models based solely on BLP.

Finally it’s worth noting that even with the high-water-mark refinement, BLP still doesn’t deal with the creation or destruction of subjects or objects (which is one of the hard problems of building a real MLS system).

#### **9.3.4 The evolution of MLS policies**

Multilevel security policies have evolved in parallel in both the practical and research worlds.

The first multilevel security policy was a version of high water mark written in 1967–8 for the ADEPT-50, a mandatory access control system developed for the IBM S/360 mainframe [2006]. This used triples of level, compartment and group, with the groups being files, users, terminals and jobs. As programs (rather than processes) were subjects, it was vulnerable to Trojan horse compromises. Nonetheless, it laid the foundation for BLP, and also led to the current IBM S/390 mainframe hardware security architecture [940].

The next big step was Multics. This had started as an MIT project in 1965 and developed into a Honeywell product; it became the template and inspirational example for ‘trusted systems’. The evaluation that was carried out on it by Paul Karger and Roger Schell was hugely influential and was the first appearance of the idea that malware could be hidden in the compiler [1019] – and led to Ken Thompson’s famous paper ‘Reflections on Trusting Trust’ ten years later [1883]. Multics had a derivative system called SCOMP that I’ll discuss in section 9.4.1 .

### 9.3. MULTILEVEL SECURITY POLICY

---

The torrent of research money that poured into multilevel security from the 1980s led to a number of alternative formulations. *Noninterference* was introduced by Joseph Goguen and Jose Meseguer in 1982 [773]. In a system with this property, High’s actions have no effect on what Low can see. *Nondeducibility* is less restrictive and was introduced by David Sutherland in 1986 [1847] to model applications such as a LAN on which there are machines at both Low and High, with the High machines encrypting their LAN traffic<sup>6</sup>. Nondeducibility turned out to be too weak, as there’s nothing to stop Low making deductions about High input with 99% certainty. Other theoretical models include *Generalized Noninterference* and *restrictiveness* [1276]; the *Harrison-Ruzzo-Ullman* model tackles the problem of how to deal with the creation and deletion of files, on which BLP is silent [868]; and the *Compartmented Mode Workstation* (CMW) policy attempted to model the classification of information using floating labels, as in the high water mark policy [2040, 807].

Out of this wave of innovation, the model with the greatest impact on modern systems is probably the *type enforcement* (TE) model, due to Earl Boebert and Dick Kain [271], later extended by Lee Badger and others to *Domain and Type Enforcement* (DTE) [153]. This assigns subjects to *domains* and objects to *types*, with matrices defining permitted domain-domain and domain-type interactions. This is used in SELinux, now a component of Android, which simplifies it by putting both subjects and objects in types and having a matrix of allowed type pairs [1187]. In effect this is a second access-control matrix; in addition to having a user ID and group ID, each process has a security ID (SID). The Linux Security Modules framework provides pluggable security where you can set rules that operate on SIDs.

DTE introduced a language for configuration (DTEL), and implicit typing of files based on pathname; so all objects in a given subdirectory may be declared to be in a given domain. DTE is more general than BLP, as it starts to deal with integrity as well as confidentiality concerns. One of the early uses was to enforce trusted pipelines: the idea is to confine a set of processes in a pipeline so that each can only talk to the previous stage and the next stage. This can be used to assemble guards and firewalls which cannot be bypassed unless at least two stages are compromised [1430]. Type-enforcement mechanisms can be aware of code versus data, and privileges can be bound to code; in consequence the tranquility problem can be dealt with at execute time rather than as data are read. This can make things much more tractable. They are used, for example, in the Sidewinder firewall.

The downside of the greater flexibility and expressiveness of TE/DTE is that it is not always straightforward to implement policies like BLP, because of state explosion; when writing a security policy you have to consider all the possible interactions between different types. Other mechanisms may be used to manage policy complexity, such as running a prototype for a while to observe what counts as normal behaviour; you can then turn on DTE and block all the information flows not seen to date. But this doesn’t give much assurance that

---

<sup>6</sup>Quite a lot else is needed to do this right, such as padding the High traffic with nulls so that Low users can’t do traffic analysis – see [1632] for an early example of such a system. You may also need to think about Low traffic over a High network, such as facilities for soldiers to phone home.

the policy you've derived is the right one.

In 1992, *role-based access control* (RBAC) was introduced by David Ferraiolo and Richard Kuhn to manage policy complexity. It formalises rules that attach primarily to roles rather than to individual users or machines [678, 679]. Transactions that may be performed by holders of a given role are specified, then mechanisms for granting membership of a role (including delegation). Roles, or groups, had for years been the mechanism used in practice in organizations such as banks to manage access control; the RBAC model started to formalize this. It can be used to give finer-grained control, for example by granting different access rights to ‘Ross as Professor’, ‘Ross as member of the Admissions Committee’ and ‘Ross reading private email’. A variant of it, aspect-based access control (ABAC), adds context, so you can distinguish ‘Ross at his workstation in the lab’ from ‘Ross on his phone somewhere on Earth’. Both have been supported by Windows since Windows 8.

SELinux builds it on top of TE, so that users are mapped to roles at login time, roles are authorized for domains and domains are given permissions to types. On such a platform, RBAC can usefully deal with integrity issues as well as confidentiality, by allowing role membership to be revised when certain programs are invoked. Thus, for example, a process calling untrusted software that had been downloaded from the net might lose the role membership required to write to sensitive system files. I discuss SELinux in more detail at 9.5.2.

#### 9.3.5 The Biba model

The incorporation into Windows 7 of a multilevel integrity model revived interest in a security model devised in 1975 by Ken Biba [237], which deals with integrity alone and ignores confidentiality. Biba’s observation was that confidentiality and integrity are in some sense dual concepts – confidentiality is a constraint on who can read a message, while integrity is a constraint on who can write or alter it. So you can recycle BLP into an integrity policy by turning it upside down.

As a concrete application, an electronic medical device such as an ECG may have two separate modes: calibration and use. Calibration data must be protected from corruption, so normal users should be able to read it but not write to it; when a normal user resets the device, it will lose its current user state (i.e., any patient data in memory) but the calibration must remain unchanged. Only an authorised technician should be able to redo the calibration.

To model such a system, we can use a multilevel integrity policy with the rules that we can read data at higher levels (i.e., a user process can read the calibration data) and write to lower levels (i.e., a calibration process can write to a buffer in a user process); but we must never read down or write up, as either could allow High integrity objects to become contaminated with Low – i.e. potentially unreliable – data. The Biba model is often formulated in terms of the *low water mark* principle, which is the dual of the high water mark principle discussed above: the integrity of an object is the lowest level of all the objects that contributed to its creation.

This was the first formal model of integrity. A surprisingly large number of real systems work along Biba lines. For example, the passenger informa-

tion system in a railroad may get information from the signalling system, but shouldn't be able to affect it; and an electricity utility's power dispatching system will be able to see the safety systems' state but not interfere with them. The safety-critical systems community talks in terms of *safety integrity levels*, which relate to the probability that a safety mechanism will fail and to the level of risk reduction it is designed to give.

Windows, since version 6 (Vista), marks file objects with an integrity level, which can be Low, Medium, High or System, and implements a default policy of NoWriteUp. Critical files are at System and other objects are at Medium by default – except for the browser which is at Low. So things downloaded using IE can read most files in a Windows system, but cannot write to them. The goal is to limit the damage that can be done by malware.

As you might expect, Biba has the same fundamental problems as Bell-LaPadula. It cannot accommodate real-world operation very well without numerous exceptions. For example, a real system will usually require trusted subjects that can override the security model, but Biba on its own cannot protect and confine them, any more than BLP can. For example, a car's airbag is on a less critical bus than the engine, but when it deploys you assume there's a risk of a fuel fire and switch the engine off. There are other real integrity goals that Biba also cannot express, such as assured pipelines. In the case of Windows, Microsoft even dropped the NoReadDown restriction and did not end up using its integrity model to protect the base system from users, as this would have required even more frequent user confirmation. In fact, the Type Enforcement model was introduced by Boebert and Kain as an alternative to Biba. It is unfortunate that Windows didn't incorporate TE.

## 9.4 Historical Examples of MLS Systems

The second edition of this book had a much fuller history of MLS systems; since these have largely gone out of fashion, and the MLS research programme has been wound down, I give a shorter version here.

### 9.4.1 SCOMP

A key product was the *secure communications processor* (SCOMP), a derivative of Multics launched in 1983 [710]. This was a no-expense-spared implementation of what the US Department of Defense believed it wanted for handling messaging at multiple levels of classification. It had formally verified hardware and software, with a minimal kernel to keep things simple. Its operating system, STOP, used Multics' system of rings to maintain up to 32 separate compartments, and to allow appropriate one-way information flows between them.

SCOMP was used in applications such as military *mail guards*. These are firewalls that allow mail to pass from Low to High but not vice versa [538]. (In general, a device which supports one-way flow is known as a *data diode*.) SCOMP's successor, XTS-300, supported C2G, the Command and Control Guard. This was used in the time phased force deployment data (TPFDD) system whose

function was to plan US troop movements and associated logistics. SCOMP's most significant contribution was to serve as a model for the *Orange Book* [544] – the US Trusted Computer Systems Evaluation Criteria. This was the first systematic set of standards for secure computer systems, being introduced in 1985 and finally retired in December 2000. The Orange Book was enormously influential not just in the USA but among allied powers; countries such as the UK, Germany, and Canada based their own national standards on it, until these national standards were finally subsumed into the Common Criteria [1396].

The Orange Book allowed systems to be evaluated at a number of levels with A1 being the highest, and moving downwards through B3, B2, B1 and C2 to C1. SCOMP was the first system to be rated A1. It was also extensively documented in the open literature. Being first, and being fairly public, it set a target for the next generation of military systems.

MLS versions of Unix started to appear in the late 1980s, such as AT&T's System V/MLS [47]. This added security levels and labels, showing that MLS properties could be introduced to a commercial operating system with minimal changes to the system kernel. By this book's second edition (2007), Sun's Solaris had emerged as the platform of choice for high-assurance server systems and for many clients as well. *Comparted Mode Workstations* (CMWs) were an example of the latter, allowing data at different levels to be viewed and modified at the same time, so an intelligence analyst could read 'Top Secret' data in one window and write reports at 'Secret' in another, without being able to accidentally copy and paste text downwards [932]. For the engineering, see [635, 636].

#### 9.4.2 Data diodes

It was soon realised that simple mail guards and crypto boxes were too restrictive, as more complex networked services were developed besides mail. First-generation MLS mechanisms were inefficient for real-time services.

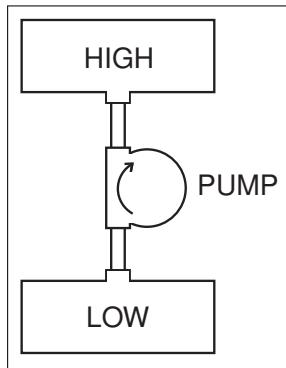


Figure 9.3: – the NRL pump

The US Naval Research Laboratory (NRL) therefore developed the *Pump* – a one-way data transfer device (a data diode) to allow secure one-way information flow (Figure 9.3. The main problem is that while sending data from Low to High is easy, the need for assured transmission reliability means that acknowl-

#### **9.4. HISTORICAL EXAMPLES OF MLS SYSTEMS**

---

edgement messages must be sent back from High to Low. The Pump limits the bandwidth of possible backward leakage using a number of mechanisms such as buffering and random timing of acknowledgements [1012, 1013, 1014]. The attraction of this approach is that one can build MLS systems by using data diodes to connect separate systems at different security levels. As these systems don't process data at more than one level – an architecture called *system high* – they can be built from cheap *commercial-off-the-shelf* (COTS) components. You don't need to worry about applying MLS internally, merely protecting them from external attack, whether physical or network-based. As the cost of hardware has fallen, this has become the preferred option, and the world's military bases are now full of KVM switches (which let people switch their keyboard, video display and mouse between Low and High systems) and data diodes (to link Low and High networks). The pump's story is told in [1015].

An early application was logistics. Some signals intelligence equipment is 'Top Secret', while things like jet fuel and bootlaces are not; but even such simple commodities may become 'Secret' when their quantities or movements might leak information about tactical intentions. The systems needed to manage all this can be hard to build; MLS logistics projects in both the USA and UK have ended up as expensive disasters. In the UK, the Royal Air Force's Logistics Information Technology System (LITS) was a 10 year (1989–99), £500m project to provide a single stores management system for the RAF's 80 bases [1386]. It was designed to operate on two levels: 'Restricted' for the jet fuel and boot polish, and 'Secret' for special stores such as nuclear bombs. It was initially implemented as two separate database systems connected by a pump to enforce the MLS property. The project became a classic tale of escalating costs driven by creeping changes in requirements. One of these changes was the easing of classification rules with the end of the Cold War. As a result, it was found that almost all the 'Secret' information was now static (e.g., operating manuals for air-drop nuclear bombs that are now kept in strategic stockpiles rather than at airbases). To save money, the 'Secret' information is now kept on a CD and locked up in a safe.

Another major application of MLS is in wiretapping. The target of investigation should not know they are being wiretapped, so the third party must be silent – and when phone companies started implementing wiretaps as silent conference calls, the charge for the conference call had to go to the wiretapper, not to the target. The modern requirement is a multilevel one: multiple agencies at different levels may want to monitor a target, and each other, with the police tapping a drug dealer, an anti-corruption unit watching the police, and so on. Eliminating covert channels is harder than it looks; for a survey from the mid-2000s, see [1707]; a pure MLS security policy is insufficient, as suspects can try to hack or confuse wiretapping equipment, which therefore needs to resist online tampering. In one notorious case, a wiretap was discovered on the mobile phones of the Greek Prime Minister and his senior colleagues during the Athens Olympics; the lawful intercept facility in the mobile phone company's switchgear was abused by unauthorised software, and was detected when the buggers' modifications caused some text messages not to be delivered [1550]. The phone company was fined 76 million Euros (almost \$100m). The clean way to manage wiretaps nowadays with modern VOIP systems may just be to write everything to disk and extract what you need later.

There are many military embedded systems too. In submarines, speed, reactor output and RPM are all Top Secret, as a history of these three measurements would reveal the vessel’s performance – and that’s among the few pieces of information that even the USA and the UK don’t share. The engineering is made more complex by the need for the instruments not to be Top Secret when the vessel is in port, as that would complicate maintenance. And as for air combat, some US radars won’t display the velocity of a US aircraft whose performance is classified, unless the operator has the appropriate clearance. When you read stories about F-16 pilots seeing an insanely fast UFO whose speed on their radar didn’t make any sense, you can put two and two together. It will be interesting to see what sort of other side-effects follow when powerful actors try to bake MAC policies into IoT infrastructure, and what sort of superstitious beliefs they give rise to.

## 9.5 MAC: from MLS to IFC and integrity

In the first edition of this book, I noted a trend to use mandatory access controls to prevent tampering and provide real-time performance guarantees [1313, 1018], and ventured that “perhaps the real future of multilevel systems is not in confidentiality, but integrity.” Government agencies had learned that MAC was what it took to stop malware. By the second edition, multilevel integrity had hit the mass market in Windows, which essentially uses the Biba model.

### 9.5.1 Windows

In Windows, all processes do, and all securable objects (including directories, files and registry keys) may, have an integrity-level label. File objects are labelled ‘Medium’ by default, while Internet Explorer (and everything downloaded using it) is labelled ‘Low’. User action is therefore needed to upgrade downloaded content before it can modify existing files. It’s also possible to implement a crude BLP policy using Windows, as you can also set ‘NoReadUp’ and ‘NoExecuteUp’ policies. These are not installed as default; Microsoft was concerned about malware installing itself in the system and then hiding. Keeping the browser ‘Low’ makes installation harder, and allowing all processes (even Low ones) to inspect the rest of the system makes hiding harder. But this integrity-only approach to MAC does mean that malware running at Low can steal all your data; so some users might care to set ‘NoReadUp’ for sensitive directories. This is all discussed by Joanna Rutkowska in [1634]; she also describes some interesting potential attacks based on virtualization.

### 9.5.2 SELinux

The case of SELinux is somewhat similar to Windows in that the immediate goal of mandatory access control mechanisms was also to limit the effects of a compromise. SELinux [1187] was implemented by the NSA, based on the Flask security architecture [1811], which separates the policy from the enforcement mechanism; a security context contains all of the security attributes associated

with a subject or object in Flask, where one of those attributes includes the Type Enforcement type attribute. A security identifier is a handle to a security context, mapped by the security server. This is where policy decisions are made and resides in the kernel for performance [819]. It has been mainstream since Linux 2.6. The server provides a security API to the rest of the kernel, behind which the security model is hidden. The server internally implements a general constraints engine that can express RBAC, TE, and MLS. In typical Linux distributions from the mid-2000s, it was used to separate various services, so an attacker who takes over your web server does not thereby acquire your DNS server as well. Its adoption by Android has made it part of the world's most popular operating system, as described in chapter 6.

### 9.5.3 Embedded systems

There are many fielded systems that implement some variant of the Biba model. As well as the medical-device and railroad signalling applications I already mentioned, there are utilities. In an electricity utility, for example, there is typically a hierarchy of safety systems, which operate completely independently at the highest safety integrity level; these are visible to, but cannot be influenced by, operational systems such as power dispatching; retail-level metering systems can be observed by, but not influenced by, the billing system. Both retail meters and the substation-level meters in the power-dispatching system feed information into fraud detection, and finally there are the executive information systems, which can observe everything while having no direct effect on operations. In cars, most makes have separate CAN buses for the powertrain and for the cabin, as you don't want a malicious app on your radio to be able to operate your brakes (though in 2010, security researchers found that the separation was completely inadequate [1085]).

It's also worth bearing in mind that simple integrity controls merely stop malware taking over the machine – they don't stop it infecting a Low compartment and using that as a springboard from which to spread elsewhere, or to issue instructions to other machines.

To sum up, many of the lessons learned in the early multilevel systems go across to a number of applications of wider interest. So do a number of the failure modes, which I'll now discuss.

## 9.6 What Goes Wrong

Engineers learn more from the systems that fail than from those that succeed, and here MLS systems have been an effective teacher. The billions of dollars spent on building systems to follow a simple policy with a high level of assurance have clarified many second-order and third-order consequences of information flow controls. I'll start with the more theoretical and work through to the business and engineering end.

### 9.6.1 Composability

Consider a simple device that accepts two ‘High’ inputs  $H_1$  and  $H_2$ ; multiplexes them; encrypts them by xor’ing them with a one-time pad (i.e., a random generator); outputs the other copy of the pad on  $H_3$ ; and outputs the ciphertext, which being encrypted with a cipher system giving perfect secrecy, is considered to be low (output  $L$ ), as in Figure 9.4.

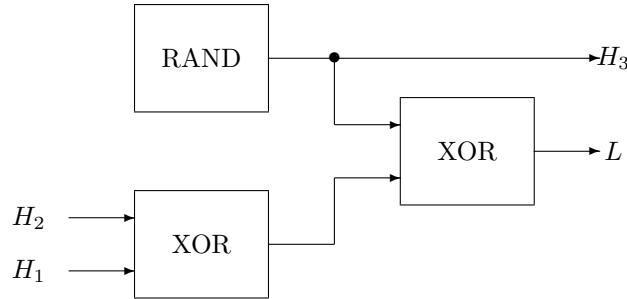


Figure 9.4 – insecure composition of secure systems with feedback

In isolation, this device is provably secure. However, if feedback is permitted, then the output from  $H_3$  can be fed back into  $H_2$ , with the result that the high input  $H_1$  now appears at the low output  $L$ . Timing inconsistencies can also break the composition of two secure systems (noted by Daryl McCullough [1260]).

In general, the *composition problem* – how to compose two or more secure components into a secure system – is hard, even at the relatively uncluttered level of proving results about ideal components [1430]. (Simple information flow doesn’t compose; neither does noninterference or nondeducibility.) Most of the low-level problems arise when some sort of feedback is introduced; without it, composition can be achieved under a number of formal models [1277]. However, in real life, feedback is pervasive, and composition of security properties can be made even harder by interface issues, feature interactions and so on. For example, one system might produce data at such a rate as to perform a service-denial attack on another. And the composition of secure components is often frustrated by higher-level incompatibilities. Components might have been designed in accordance with two different security policies, or designed according to inconsistent requirements.

### 9.6.2 The cascade problem

An example of the composition problem is given by the *cascade problem* (Figure 9.5). After the Orange book introduced a series of evaluation levels, this led to span-limit rules about the number of levels at which a system can operate [548]. For example, a system evaluated to B3 was in general allowed to

process information at Unclassified, Confidential and Secret, or at Confidential, Secret and Top Secret; there was no system permitted to process Unclassified and Top Secret data simultaneously [548].

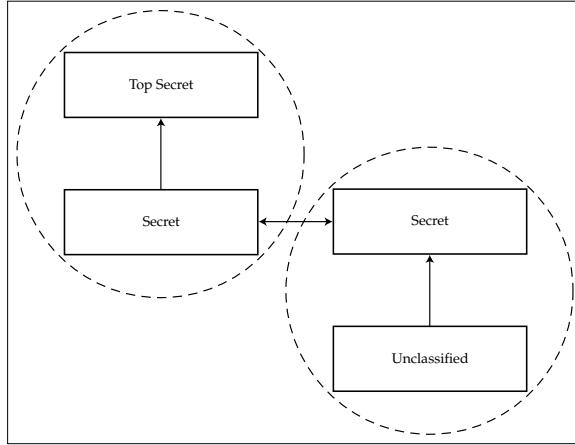


Figure 9.5: – the cascade problem

As the diagram shows, it is straightforward to connect together two B3 systems in such a way that this policy is broken. The first system connects together Unclassified and Secret, and its Secret level communicates with the second system – which also processes Top Secret information [923]. This defeats the span limit.

### 9.6.3 Covert channels

One of the reasons why span limits are imposed on multilevel systems emerges from a famous – and extensively studied – problem: the *covert channel*. First pointed out by Lampson in 1973 [1125], a covert channel is a mechanism that was not designed for communication but which can nonetheless be abused to allow information to be communicated down from High to Low.

A typical covert channel arises when a high process can signal to a low one by affecting some shared resource. In a modern multicore CPU, it could increase the clock frequency of the CPU core it's using at time  $t_i$  to signal that the  $i$ -th bit in a Top Secret file was a 1, and let it scale back to signal that the bit was a 0. This gives a covert channel capacity of several tens of bits per second [35]. Since 2018, CPU designers have been struggling with a series of cover channels that exploit the CPU microarchitecture; with names like Meltdown, Spectre, and Foreshadow, they have provided not just ways for High to signal to Low but for Low to circumvent access control and read memory at High. I will discuss these in detail in the chapter on side channels.

The best that developers have been able to do consistently with confidentiality protection in regular operating systems is to limit it to 1 bit per second or so. (That is a DoD target [545], and techniques for doing a systematic analysis may be found in Kemmerer [1036].) One bit per second may be tolerable in an environment where we wish to prevent large TS/SCI files – such as satellite

photographs – leaking down from TS/SCI users to ‘Secret’ users. However, it’s potentially a lethal threat to high-value cryptographic keys. This is one of the reasons for the military and banking doctrine of doing crypto in special purpose hardware.

The highest-bandwidth covert channel of which I’m aware occurs in large early-warning radar systems, where High – the radar processor – controls hundreds of antenna elements that illuminate Low – the target – with high speed pulse trains, which are modulated with pseudorandom noise to make jamming harder. In this case, the radar code must be trusted as the covert channel bandwidth is many megabits per second.

### 9.6.4 The threat from malware

The defense computer community was shocked when Fred Cohen wrote the first thesis on computer viruses, and used a virus to penetrate multilevel secure systems easily in 1983. In his first experiment, a file virus that took only eight hours to write managed to penetrate a system previously believed to be multilevel secure [450]. People had been thinking about malware since the 1960s and had done various things to mitigate it, but their focus had been on Trojans.

There are many ways in which malicious code can be used to break access controls. If the reference monitor (or other TCB components) can be corrupted, then malware can deliver the entire system to the attacker, for example by issuing an unauthorised clearance. For this reason, slightly looser rules apply to so-called *closed security environments* which are defined to be those where ‘system applications are adequately protected against the insertion of malicious logic’ [548], and this in turn created an incentive for vendors to tamper-proof the TCB, using techniques such as TPMs. But even if the TCB remains intact, malware could still copy itself up from Low to High (which BLP doesn’t prevent) and use a covert channel to signal information down.

### 9.6.5 Polyinstantiation

Another problem that exercised the research community is *Polyinstantiation*. Suppose our High user has created a file named `agents`, and our Low user now tries to do the same. If the MLS operating system prohibits him, it will have leaked information – namely that there is a file called `agents` at High. But if it lets him, it will now have two files with the same name.

Often we can solve the problem by a naming convention, such as giving Low and High users different directories. But the problem remains a hard one for databases [1649]. Suppose that a High user allocates a classified cargo to a ship. The system will not divulge this information to a Low user, who might think the ship is empty, and try to allocate it another cargo or even to change its destination.

Here the US and UK practices diverge. The solution favoured in the USA is that the High user allocates a Low cover story at the same time as the real High cargo. Thus the underlying data will look something like Figure 9.6.

## 9.6. WHAT GOES WRONG

---

Level	Cargo	Destination
Secret	Missiles	Iran
Restricted	—	—
Unclassified	Engine spares	Cyprus

Figure 9.6 – how the USA deals with classified data

In the UK, the theory is simpler – the system will automatically reply ‘classified’ to a Low user who tries to see or alter a High record. The two available views would be as in Figure 9.7.

Level	Cargo	Destination
Secret	Missiles	Iran
Restricted	Classified	Classified
Unclassified	—	—

Figure 9.7 – how the UK deals with classified data

This makes the system engineering simpler. It also prevents the mistakes and covert channels that can still arise with cover stories (e.g., a Low user tries to add a container of ammunition for Cyprus). The drawback is that everyone tends to need the highest available clearance in order to get their work done. (In practice, cover stories still get used in order not to advertise the existence of a covert mission any more than need be.)

### 9.6.6 Practical problems with MLS

Multilevel secure systems are surprisingly expensive and difficult to build and deploy. There are many sources of cost and confusion.

1. They are built in small volumes, and often to high standards of physical robustness, using elaborate documentation, testing and other quality control measures driven by military purchasing bureaucracies.
2. MLS systems have idiosyncratic administration tools and procedures. A trained Unix administrator can't just take on an MLS installation without significant further training; so many MLS systems are installed without their features being used.
3. Many applications need to be rewritten or at least greatly modified to run under MLS operating systems [1629].
4. Because processes are automatically upgraded as they see new labels, the files they use have to be too. New files default to the highest label belonging to any possible input. The result of all this is a chronic tendency for things to be overclassified. There's a particular problem when system components accumulate all the labels they've seen, leading to *label explosion*.

where they acquire such a collection that no single principal can access them any more. So they get put in the trusted computing base, which ends up containing a quite uncomfortably large part of the operating system (plus utilities, plus windowing system software, plus middleware such as database software). This ‘TCB bloat’ constantly pushes up the cost of evaluation and reduces assurance.

5. The classification of data can get complex:

- in the run-up to a conflict, the location of ‘innocuous’ stores such as food could reveal tactical intentions, and so may be suddenly upgraded;
- classifications are not always monotone. Equipment classified at ‘confidential’ may easily contain components classified ‘secret’, and on the flip side it’s hard to grant access at ‘secret’ to secret information in a ‘top secret’ database;
- information may need to be downgraded. An intelligence analyst might need to take a satellite photo classified at TS/SCI, and paste it into an assessment for field commanders at ‘secret’. In case information was covertly hidden in the image by a virus, this may involve special filters, lossy compression of images and so on. One option is a ‘print-and-fax’ mechanism that turns a document into a bitmap, and logs it for traceability.
- we may need to worry about the volume of information available to an attacker. For example, we might be happy to declassify any single satellite photo, but declassifying the whole collection would reveal our surveillance capability and the history of our intelligence priorities. (I will look at this *aggregation problem* in more detail in section 11.2.)
- Similarly, the output of an unclassified program acting on unclassified data may be classified, for example if standard data mining techniques applied to an online forum throw up a list of terror suspects.

6. Although MLS systems can prevent undesired things (such as information leakage), they also prevent desired things too (such as building a search engine to operate across all an agency’s Top Secret compartmented data). So even in military environments, the benefits can be questionable. After 9/11, many of the rules were relaxed, and access controls above Top Secret are typically discretionary, to allow information sharing. The cost of that, of course, was the Snowden disclosures.

7. Finally, obsessive government secrecy is a chronic burden. The late Senator Daniel Moynihan wrote a critical study of its real purposes, and its huge costs in US foreign and military affairs [1346]. For example, President Truman was never told of the Venona decrypts because the material was considered ‘Army Property’. As he put it: “Departments and agencies hoard information, and the government becomes a kind of market. Secrets become organizational assets, never to be shared save in exchange for another organization’s assets.”

More recent examples of MLS doctrine impairing operational effectiveness include the use of unencrypted communications to drones in the Afghan war (as the armed forces feared that if they got the NSA bureaucracy involved, the drones would be unusable), and the use of the notoriously insecure Zoom videoconferencing system for British government cabinet meetings during the coronavirus crisis (the government's encrypted videoconferencing terminals are classified, so ministers aren't allowed to take them home). This brings to mind a quip from an exasperated British general: "What's the difference between Jurassic Park and the Ministry of Defence? One's a theme park full of dinosaurs, and the other's a movie!"

There has been no shortage of internal strategic critique. A 2004 report by Mitre's JASON programme of the US system of classification concluded that it was no longer fit for purpose [978]. There are many interesting reasons, including the widely different risk/benefit calculations of the producer and consumer communities; classification comes to be dominated by distribution channels rather than by actual risk. The relative ease of attack has led government systems to be too conservative and risk-averse. It noted many perverse outcomes; for example, Predator imagery in Iraq is Unclassified, and was for some time transmitted in clear, as the Army feared that crypto would involve the NSA bureaucracy in key management and inhibit warfighting.

Mitre proposed instead that flexible compartments be set up for specific purposes, particularly when getting perishable information to tactical compartments; that intelligent use be made of technologies such as rights management and virtualisation; and that lifetime trust in cleared individuals be replaced with a system focused on transaction risk.

Anyway, one of the big changes since the second edition of this book is that the huge DoD research programme on MLS has disappeared, MLS equipment is no longer very actively promoted on the government-systems market, and systems have remained fairly static for a decade. Most government systems now operate system high – that is, entirely at Official, or at Secret, or at Top Secret. The difficulties discussed in the above section, plus the falling cost of hardware and the arrival of virtualisation, have undermined the incentive to have different levels on the same machine. The deployed MLS systems thus tend to be firewalls or mail guards between the different levels, and are often referred to by a new acronym, MILS (for multiple independent levels of security). The real separation is at the network level, between unclassified networks, the Secret Internet Protocol Router Network (SIPRNet) which handles secret data using essentially standard equipment behind crypto, and the Joint Worldwide Intelligence Communications System (JWICS) which handles Top Secret material and whose systems are kept in Secure Compartmentalized Information Facilities (SCIFs) – rooms shielded to prevent electronic eavesdropping, which I'll discuss later in the chapter on side channels.

There are occasional horrible workarounds such as 'browse-down' systems that will let someone at High view a website at Low; they're allowed to click on buttons and links to navigate, just not to enter any text. Such ugly hacks have clear potential for abuse; at best they can help keep honest people from careless mistakes.

## 9.7 Summary

Mandatory access control was initially developed for military applications, where it is still used in specialized firewalls (guards and data diodes). The main use of MAC mechanisms nowadays, however, is in platforms such as Android, iOS and Windows, where they protect the operating systems themselves from malware. MAC mechanisms have been a major subject of computer security research since the mid-1970's, and the lessons learned in trying to use them for military multilevel security underlie many of the schemes used for security evaluation. It is important for the practitioner to understand both their strengths and limitations, so that you can draw on the research literature when it's appropriate, and avoid being dragged into overdesign when it's not.

There are many problems which we need to be a ‘fox’ rather than a ‘hedgehog’ to solve. By trying to cast all security problems as hedgehog problems, MLS often leads to inappropriate security goals, policies and mechanisms.

## Research Problems

A standing challenge, sketched out by Earl Boebert in 2001 after the NSA launched SELinux, is to adapt mandatory access control mechanisms to safety-critical systems (see the quote at the head of this chapter, and [270]). As a tool for building high-assurance, special-purpose devices where the consequences of errors and failures can be limited, mechanisms such as type enforcement and role-based access control should be useful outside the world of security. Will we see them widely used in the Internet of Things? We've mentioned Biba-type mechanisms in applications such as cars and electricity distribution; will the MAC mechanisms in products such as SELinux, Windows and Android enable designers to lock down information flows and reduce the likelihood of unanticipated interactions?

The NSA continues to fund research on MLS, now under the label of IFC, albeit at a lower level than in the past. Doing it properly in a modern smartphone is hard; for an example of such work, see the Weir system by Adwait Nadkarni and colleagues [1372]. In addition to the greater intrinsic complexity of modern operating systems, phones have a plethora of side-channels and their apps are often useful only in communication with cloud services, where the real heavy lifting has to be done. The commercial offering for separate ‘low’ and ‘high’ phones consists of products such as Samsung’s Knox.

A separate set of research issues surround actual military opsec, where reality falls far short of policy. All armed forces involved in recent conflicts, including US and UK forces in Iraq and Afghanistan, have had security issues around their personal mobile phones, with insurgents in some cases tracing their families back home and harassing them with threats. The Royal Navy tried to ban phones in 2009, but too many sailors left. Tracking ships via Instagram is easy; a warship consists of a few hundred young men and women, aged 18-24, with nothing much else to do but put snaps on social media. Discipline tends to focus on immediate operational threats, such as when a sailor is seen snapchatting on mine disposal: there the issue is the risk of using a radio near a mine! Different navies have

tried different things: the Norwegians have their own special network for sailors and the USA is trying phones with MLS features. But NATO exercises have shown that for one navy to hack another's navigation is shockingly easy. And even the Israelis have had issues with their soldiers using mobiles on the West Bank and the Golan Heights.

## Further Reading

The unclassified manuals for the UK government's system of information classification, and the physical, logical and other protection mechanisms required at the different levels, have been available publicly since 2013, with the latest documents (at the time of writing) having been released in November 2018 on the Government Security web page [802]. The report on the Walker spy ring is a detailed account of a spectacular failure, and brings home the sheer complexity of running a system in which maybe three million people have a clearance at any one time, with a million applications being processed each year [876]. And the classic on the abuse of the classification process to cover up waste, fraud and mismanagement in the public sector is by Chapman [407].

On the technical side, textbooks such as Dieter Gollmann's *Computer Security* [779] give an introduction to MLS systems, while many of the published papers on actual MLS systems can be found in the proceedings of two conferences: academics' conference is the *IEEE Symposium on Security & Privacy* (known in the trade as 'Oakland' as that's where it used to be held), while the NSA supplier community's unclassified bash is the *Computer Security Applications Conference* (ACSAC) whose proceedings are (like Oakland's) published by the IEEE. Fred Cohen's experiments on breaking MLS systems using viruses are described in his book, '*A Short Course on Computer Viruses*' [450]. Many of the classic early papers in the field can be found at the NIST archive [1395]; NIST ran a conference series on multilevel security up till 1999. Finally, a history of the Orange Book was written by Steve Lipner [1171]; this also tells the story of the USAF's early involvement and what was learned from systems like WWMCCS.

be resented and perhaps sabotaged.

Should Level 5 automation ever happen, even in restricted environments – so that we finally see the robotaxis Google hoped to invent – then we’ll have to think about social hacking as a facet of safety. If your 12-year-old daughter calls a cab to get a ride home from school, then at present we have safeguards in the form of laws requiring taxi drivers to have background checks for criminal records. Uber tried to avoid these laws, claiming it wasn’t a taxi company but a ‘platform’; in London, the mayor had to ban them and fight them in court for years to get them to comply. So how will safeguarding work with robotaxis?

There will also be liability games. At present, car companies try to blame drivers for crashes, so each crash becomes a question of which driver was negligent. If the computer was driving the car, though, that’s product liability, and the manufacturer has to pay. There have been some interesting tussles around the safety figures for assisted driving, and specifically whether the carmakers undercount crashes with autopilot activated, which we’ll discuss in section 28.4.1.

So much is entirely predictable. But what about new attacks on the AI components of the systems themselves? For example, can you confuse a car by projecting a deceptive image on a bridge, or on the road, and cause it to crash? That’s quite possible, and I’ve already seen a crash caused by visual confusion. On the road home from my lab, there was a house at a right-hand bend whose owner often parked his car facing oncoming traffic. At night, in a left-hand driving country like Britain, your driving reflex is to steer to the left of the facing car, but then you’d notice you were heading for his garden wall, and swerve right to pass to the right of his car instead. Eventually a large truck didn’t swerve in time, and ended up in the wall.

So could clever software fool a machine vision system in new ways, or ways that might be easier for an attacker to scale? That brings us to the next topic, artificial intelligence, or to be more precise, machine learning.

## 25.3 AI / ML

The phrase *artificial intelligence* has meant different things at different times. For pioneers like Alan Turing, it ranged from the Turing test to attempts to teach a computer to play chess. By the 1960s it meant text processing, from Eliza to early machine translation, and programming in Lisp. In the 1980s there was a surge of research spurred by Japan’s announcement of a huge research programme into ‘Fifth generation computing’, with which Western nations scrambled to keep up; much of that effort went into rule-based systems, and Prolog joined Lisp as one of the languages on the computer science curriculum.

From the 1990s, the emphasis changed from handcrafted systems with lots of rules to systems that learn from examples, now called *machine learning* (ML). Early mechanisms included logistic regressions, support vector machines (SVMs) and Bayesian classifiers; progress was driven by applications such as natural language processing (NLP) and search. While the NLP community developed custom methods, the typical approach to designing a payment fraud detector or spam filter was to collect large amounts of training data, write custom code

to extract a number of signals, and just see empirically which type of classifier worked best on them. Search became intensely adversarial during the 2000s as search engine optimisation firms used all sorts of tricks to manipulate the signals on which search engines rely, and the engines fought back in turn, penalising or banning sites that use underhand tricks such as hidden text. Bing was an early user of ML, but Google avoided it for years; the engineer who ran search from 2000 until he retired in 2016, Amit Singhal, felt it was too hard to find out, for a given set of results, exactly which of the many inputs was most responsible for which result. This made it hard to debug machine-learning based algorithms for search ranking. If you detected a botnet clicking on restaurants in Istanbul and wanted to tweak the algorithm to exclude them, it was easier to change a few ‘if’ statements than retrain a classifier [1300].

A sea change started in 2011 when Dan Cireşan, Ueli Meier, Jonathan Masci and Jürgen Schmidhuber trained a deep convolutional neural network to do as well as humans on recognising handwritten digits and Chinese characters, and better than humans on traffic signs [435]. The following year, Alex Krizhevsky, Ilya Sutskever and Geoff Hinton used a similar *deep neural network* (DNN) to get record-breaking results at classifying 1.2 million images [1098]. The race was on, other researchers piled in, and ‘deep learning’ started to get serious traction at a variety of tasks. The most spectacular result came in 2016 when David Silver and colleagues at Google Deepmind produced AlphaGo, which defeated the world Go champion Lee Sedol [1737]. This got the attention of the world. Before then, few research students wanted to study machine learning; since then few want to study anything else. Undergraduates even pay attention in classes on probability and statistics, which were previously seen as a chore.

### 25.3.1 ML and security

The interaction between machine learning and security goes back to the mid-1990s. Malware writers started using tricks such as polymorphism to evade the classifiers in anti-virus software, as I described in section 21.3.5; banks and credit card companies started using machine learning to detect payment fraud, as I described in section 12.5.4; and phone companies also used it for first-generation mobiles, as I noted in section 22.2. The arrival of spam as the Internet opened up to the public in the mid-1990s created a market for spam filters. Hand-crafted rules didn’t scale well enough for large mail service providers, especially once botnets appeared and spam became the majority of email, so spam filtering became a big application.

Alice Hutchings, Sergo Pastrana and Richard Clayton surveyed the use of machine-learning in such systems, and the tricks the bad guys have worked out to dupe them [939]. As spam filtering takes user feedback as its ground truth, spammers learned to send spam to accounts they control at the big webmail firms, and mark it ‘not spam’; other statistical analysis mechanisms are now used to detect this. Poisoning a classifier’s training data is a quite general attack. Another is to look for weak points in a value chain: airline ticket fraudsters buy an innocuous ticket, pass the fraud checks, and then change it just before departure to a ticket to a high-risk destination. And there are vigorous discussions of such techniques on the underground forums where the

bad actors trade not just services but boasts and tips. Battista Biggio and Fabio Rolli give more technical background: in 2004, spammers found they could confuse the early linear classifiers in spam filters by varying some of the words, and an arms race took off from there [241].

It turns out that these attack ideas generalise to other systems, and there are other attacks too.

### 25.3.2 Attacks on ML systems

There are at least four types of attack on a machine-learning system.

First, you can poison the training data. If the model continues to train itself in use, then it might be simple to lead it astray. Tay was a chatbot released by Microsoft in March 2016 on Twitter; trolls immediately started teaching it to use racist and offensive language, and it was shut down after only 16 hours.

Second, you can attack the model’s integrity in its inference phase, for example by causing it to give the wrong answer. In 2013, Christian Szegedy and colleagues found that the deep neural networks which had been found to classify images so well in 2012 were vulnerable to *adversarial samples* – images perturbed very slightly would be wildly misclassified [1857]. The idea is to choose a perturbation that maximises the model’s prediction error. It turns out that neural networks have plenty of such blind spots, which are related to the training data in non-obvious ways. The decision space is high-dimensional, which makes blind spots mathematically inevitable [1706]; and with neural networks the decision boundaries are convoluted, making them non-obvious. Researchers quickly came up with real-world adversarial examples, ranging from small stickers that would cause a car vision system to misread a 30mph speed sign as 60mph, to coloured spectacles that would cause a man wearing them to be misrecognised as a woman, or not recognised at all [1720]. In the world of malware detection, people found that non-linear classifiers such as SVM and deep neural networks were not actually harder to evade than linear classifiers provided you did it right [241].

Third, Florian Tramèr and colleagues showed that you can attack the model’s confidentiality in the inference phase, by getting it to classify a number of probe inputs and building a successively better approximation. The result is often a good working imitation of the target model. As in the manufacture of real goods, a knock-off is often cheaper; big models can cost a lot to train from scratch. This approximation attack works not just with neural networks but also with other classifiers such as logistic regression and decision trees [1901].

What’s more, many attacks turn out to be transferable, so an attacker doesn’t need full access to the model (a so-called *white-box attack*) [1900]. Many attacks can be developed on one model and then launched against another that’s been trained on the same data, or even just similar data (a *black-box attack*). The blind spots are a function of the training data, so in order to make attacks less transferable you have to make an effort. For example, Ilia Shumailov, Yiren Zhao, Robert Mullins and I have experimented with inserting keys in neural networks so that the blind spots appear in different places, and models with different keys are vulnerable to different adversarial samples [1733]. Kerckhoff’s

principle applies in machine learning, as almost everywhere else in security.

A variant on the confidentiality attack is to extract sensitive training data. Large neural networks contain a lot of state, and the simplest way to deal with outliers is often just to memorise them. So if some business claims that a classifier trained on a million medical records is not personal data because it's "statistical machine learning", take care. Ways of combining machine learning with differential privacy, which we discussed in section 11.3, are a subject of active research [1493].

Finally, you can deny service, and one way is to choose samples that will cause the classifier to take as long as possible. Ilia Shumailov and colleagues found that one can often deny service by posing a conundrum to a classifier. Given a straight-through pipeline, as in a typical image-processing task, a confusing image can take 20% more time, but in more complex tasks such as natural language processing you can invoke exception handling and slow things down hundreds of times [1730].

More complex attacks straddle these categories. For example, there's an arms race between online advertisers and the suppliers of ad-blocking software, and as the advertisers adopt ever more complicated ways of rendering web pages to confuse the blockers, the blockers are starting to use image processing techniques on the rendered page to spot ads. However this leaves them open to advertisers using adversarial samples either to escape the filter, or to cause it to wrongly block another part of the page [1899].

So how can one use machine learning safely in the real world? That's something we're still learning, but there are some things we can say. First, one has to take a systems security approach and look at the problem end-to-end. Just as we sanitise inputs to web services, do penetration testing, and have mechanisms for responsible disclosure and update, we need to do the same for ML systems [659].

Second, we need to draw on the experience of the last twenty years' work on topics like card fraud, spam and intrusion detection. As we mentioned in section 21.4.2.2, ML systems have been largely ineffective at real-world network intrusion detection; Robin Sommer and Vern Paxson were the first to give a good explanation why. They discuss the lack of training data, the distance between theory and practice, the difficulties in evaluation, the high cost of errors and above all the inability to deal with novel attacks [1802]. The problem of keeping capable opponents out of complex corporate networks just isn't one that artificial intelligence has ever been good at.

There may occasionally be a change in emphasis, though. If we want to lower the probability of a new adversarial attack causing real damage, there are various things we can do, depending on the context. One is simply to detune the classifier; this is the approach in at least one machine-vision system used in cars. By making it less sensitive, you make it less easy to spoof, and then you complement it with other sensors such as radar and ultrasonics so that the vision system on its own is less critical. An alternative approach is to head in the other direction, by making the ML component of your system sufficiently fragile that an attack can be detected by other components – whereupon you switch to a defensive mode of operation, such as a low-sensitivity limp-home mode or

stopping and waiting for a human to drive. In other words, you set out to build in situational awareness. This is how we behave in real life; as I discussed in section 3.2.5.1, the ancestral evolutionary environment taught us to take extra care when we sense triggers such as adversarial intent and violations of tribal taboos. So we've experimented with using neural networks trained so that a number of outputs and activations are considered to be taboo and avoided; if any of these taboos is broken, an attack can be suspected [1733].

The fundamental problem is that once we start letting machine learning blur the boundary between code and data, and systems become data-driven, people are going to game them. This brings us to the thorny problem of the interaction of machine learning and society.

### 25.3.3 ML and society

The surge of interest in machine learning since 2016, and its representation as ‘artificial intelligence’ in the popular press, has led to a lot of speculation about ethics. For example, the philosopher Dan Dennett objects on moral grounds to the existence of persons that are immortal and intelligent but not conscious. But companies already meet that definition! The history of corporate wrongdoing shows that corporations can behave very badly indeed (we discussed some examples in section 12.2.6). The most powerful ML systems belong to corporations such as Google, Amazon, Microsoft and IBM, all of which have had tussles with authority. The interplay between ML, big data and monopoly adds to the thicket of issues that governments need to navigate as they ponder how to regulate tech. One aspect is that the tech majors’ ML offerings are now becoming platforms on their own, and used by lots of startups solving specific real-world problems [658].

One cross-cutting issue is prejudice. Aylin Caliskan, a Turkish research student at Princeton, noticed that machine translations from Turkish to English came out with gender bias; although Turkish has no grammatical gender, the English translations of Turkish sentences would assign doctors as ‘he’ and nurses as ‘she’. On further investigation, she and her supervisors Joanna Bryson and Arvind Narayanan found that essentially all machine translation systems in use were not merely sexist, but racist and homophobic too [369]. In fact a large number of natural-language systems based on machine learning inhale the prejudices of their training data. If the big platforms’ ML engines can suffuse prejudice through the systems on which hundreds of downstream firms rely, there is definitely a public-policy issue.

A related policy problem is *redlining*. When insurance companies used postcode-level claim statistics to decide the level of premiums, it was found that many minority areas suffered high premiums or were excluded from cover, breaking anti-discrimination laws. I wrote in the second edition of this book in 2008: “If you build an intrusion detection system based on data mining techniques, you are at serious risk of discriminating. If you use neural network techniques, you’ll have no way of explaining to a court what the rules underlying your decisions are, so defending yourself could be hard. Opaque rules can also contravene European data protection law, which entitles citizens to know the algorithms used to process their personal data.”

A second cross-cutting issue is snake oil, and the AI/ML gold rush has led to thousands of startups, many of them stronger on marketing than on product. Manish Raghavan and colleagues surveyed ‘AI’ systems used in employment screening and hiring, finding dozens of firms that claim their systems match new hires to the company’s requirements. Most claim they don’t discriminate, yet as few employers retain comprehensive and accessible data on employee performance, it’s entirely unclear how such systems can even be trained, let alone how a firm that used such a system might defend a lawsuit for discrimination [1571]. Applicants quickly learn to game the system, such as by slipping the word ‘Oxford’ or ‘Cambridge’ into their CV in white text. A prudent employer would demand more transparent mechanisms, and devise independent metrics to validate their outcomes. Even that is nontrivial, as machine learning can discover correlations that we do not understand.

Arvind Narayanan has an interesting analysis of snake oil in AI [1382]. ‘AI’ and even ‘ML’ are generic terms for a whole grab-bag of technologies. Some of them have made real progress, like DNNs for face recognition, and indeed AlphaGo. So companies exploit this hype, slapping the ‘AI’ label on whatever they’re selling, even if its mechanisms use statistical techniques from a century ago. Digging deeper, Arvind argues that machine-learning systems can be sorted into three categories:

1. ML has made real progress on tasks of *perception*, such as face recognition (see section 17.3), the recognition of songs by products like Shazam (see section 24.4.3), medical diagnosis from scans, and speech-to-text – at all of which it has acquired the competence of skilled humans;
2. ML has made some progress on tasks of *judgment*, such as content recommendation and the recognition of spam and hate speech. These have many hard edge cases about which even skilled humans disagree. The systems that perform them often rely on substantial human input – from a billion email users clicking the ‘report spam’ button to the tens of thousands of content moderators employed by the big tech companies;
3. ML has made no progress on tasks of *social prediction*, such as predicting employee performance, school outcomes and future criminal behaviour. A very extensive study by Matthew Sagalnik and over 400 collaborators has concluded that insofar as life outcomes can be predicted at all, this can be done as well using simple linear regressions based on a handful of variables [1638].

This is a falsifiable claim, so we’ll see how accurate it is over time, and if there’s a fourth edition of this book in 2030 we’ll have a lot more data then. A major theme of research meanwhile will be to look for better ways for people and machines to work together. Intuitively, we want people to do the jobs involving judgment and machines to do the boring stuff; but making that actually work can be harder than it looks. Often people end up being the machine’s servants, and according to one VC firm, 40% of ‘AI’ startups don’t actually use ML in any material way; they’re merely riding the wave of hype and employ people behind the scenes [1960]. One way or another, there will be lots of bumps in the road, and lots of debates about ethics and politics.

Perhaps the best way to approach the ethics is this. Many of the problems now being discussed in the context of AI ethics arose years ago for research done using traditional statistical methods on databases of personal information. (Indeed, linear regressions have been used continuously for about a century; they've just been rebranded as machine learning.) So our first port of call should be existing law and policy. When we discussed ethics in the context of records-based health and social-policy research in section 10.4.5.1, we observed that many of the issues arose because IT companies and their customers ignored the wisdom that doctors, teachers and others had accumulated over years of dealing with paper-based records. The same mistakes are now being repeated, and excused as before with sales hype around ‘innovation’ and ‘disruption’.

In the case of predicting which children are likely to turn to crime, it's been known for years that such indicators can be deeply stigmatising. In section 10.4.6 we noted that if you tell teachers which kids have had contact with social services, then the teachers will have lower expectations of them. Both child welfare and privacy law argue against sharing such indicators. How much more harmful might it be if clueless administrators buy software that claims to be making predictions using the inscrutable magic that enabled AlphaGo to beat Lee Sedol? As for ‘predictive policing’, studies suggest that it might just be another way to get the computer to justify a policy of ‘round up the usual suspects’ [677]. (In section 14.4 we discussed how curfew tags also have this effect.) Similar issues arise with the use of ML techniques to advise judges in bail hearings about whether a suspect poses a flight risk or reoffending risk, and also in sentencing hearings about whether a suspect is dangerous. Such technologies are likely to propagate existing social biases and power structures, and provide lawmakers with an excuse to continue ineffective but populist policies, rather than nudging them to tackle the underlying problems.

ML is nonetheless likely to upset some of the equilibria that have emerged over the years on issues like surveillance, privacy and censorship, as it makes even more powerful tools available to already powerful actors, as well as creating new excuses to revive old abuses. Many countries already restrict the use of CCTV cameras; now that face-recognition systems enable pedestrians to be recognised, do we need to restrict them more? As we saw in section 17.3, a number of cities (including San Francisco) have decided the answer is ‘yes’. In section 11.2.5 we discussed how location and social data can now make it very hard to be anonymous, and how people’s Facebook data could be mined for political ad targeting. ML techniques make it easier to do traffic analysis, by spotting patterns of communication [1719]; in fact, police and intelligence agencies depend ever more on traffic and social-network analysis of the sort discussed in sections 21.7, 23.3.1 and 26.2.2.

In short, the charge sheet against machine learning is that it is one of the technologies helping entrench the power of the tech majors while pushing the balance between privacy and surveillance towards surveillance and facilitating authoritarian government in other ways. It may be telling that Google and Microsoft are funding big research programs to develop AI for social good.

So what can we do as a practical matter to get some privacy in this electronic village in which we now live?

## Chapter 26

# Surveillance or Privacy?

Experience should teach us to be most on our guard to protect liberty when the government's purposes are beneficent...

The greatest dangers to liberty lurk in insidious encroachment by men of zeal, well meaning but without understanding.

– Supreme Court Justice Louis Brandeis

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

– Lord Acton

The arguments of lawyers and engineers pass through one another like angry ghosts.

– Nick Bohm, Ian Brown and Brian Gladman

### 26.1 Introduction

Governments have ever more interests online, ranging from surveillance to censorship, from privacy to safety, and from market competition to fair elections. Their goals are often in tension with the reality of a globalised online world, and with each other too. They crystallise around a number of specific policy concerns, from terrorism and counterinsurgency, through national strategic and economic advantage, to the suppression of harmful or unpopular content and the maintenance of human rights. In this chapter we explore the nexus of surveillance, censorship, forensics and privacy.

The Internet has transformed the world in lots of complicated ways, like other big technologies before it – electricity, the steam engine, writing, agriculture and fire. The relationship between the citizen and the state has changed everywhere, with the state usually acquiring more power and control. In the early years, as the PC replaced the mainframe and the Internet opened up to all, many pioneers

## **26.1. INTRODUCTION**

---

were utopians: we believed that free access to information would be liberating at the personal level, and would destabilise authoritarian governments too. Yet governments and large companies learned in time to use the new tools. The terrorist attacks of September 11, 2001, on New York and Washington had a real impact, by creating the incentive for mass surveillance and weakening political opposition to it. The move of business online created the tools, and a commercial market for personal information to pay for them. While the pendulum swung back during the 2010s towards surveillance capitalism, the COVID-19 pandemic looks set to increase state surveillance once more, with the trade-off being not privacy versus security but privacy versus health.

It's been a boom time for surveillance. It's not just the NSA capabilities revealed in 2013 by Ed Snowden; nation-state competitors like Russia and China also have serious capabilities; while there are more primitive but still effective systems in less developed countries like Syria.

The 2010s also saw growing cyber conflict and disruption with states interfering covertly in other states' affairs. The USA and Israel used the Stuxnet malware to damage and delay Iran's push to acquire nuclear weapons, and this caused a rush by other states to acquire cyber-weapons of various kinds. Since the Russian interference in the 2016 US election, legislators in a number of countries want to regulate social media: a lot of politicians have stopped ignoring technology once they realised their jobs were on the line.

There are many thorny issues. First, are open societies with democracy and a free press more vulnerable, because we're easier to exploit? And if so what can we do about it? We face real challenges to our core values – expressed in the USA as the Constitution, and in Europe as the Convention on Human Rights. Since 9/11 we've seen one authoritarian measure after another, ranging from large-scale surveillance of communications to detention without trial and even torture. Many of these measures were not just illegal and immoral but ineffective or even counterproductive: torturing Iraqi secret policemen alongside al-Qaida terrorists in the Abu Ghraib prison was what forged them into the core of Islamic State. Can't we find better ways to defend freedom? And how can we reassert and defend our core values?

Second, there's the political economy of security. President Eisenhower warned in his valedictory speech that 'we must guard against the acquisition of unwarranted influence, whether sought or unsought, by the military industrial complex. The potential for the disastrous rise of misplaced power exists and will persist'. Since 9/11, we've seen a security-industrial complex capturing policy in the same ways that the defence industry did at the start of the Cold War. Politicians of left and right have stoked a culture of fear, abetted by security agencies and the press. This has been deepened since the financial crisis of 2008 by the rise of nationalism.

Security technology arguments are often used to bamboozle or intimidate legislators. For example, all through the Irish republican terrorist campaign from the 1970s through 1990s, the British police had to charge arrested terrorist suspects within four days. But after 9/11, this was quickly raised to 28 days; then the government said it needed 90 days, claiming they might have difficulty decrypting data on PCs seized from suspects. The real problem was police

## *26.1. INTRODUCTION*

---

inefficiency at managing forensics. Now if the police had just said ‘we need to hold suspects for 90 days because we don’t have enough Somali interpreters’ then common sense could have kicked in; Parliament might well have told them to use staff from commercial translation agencies. But talk of decryption seems a good way to turn legislators’ brains to mush. People who understand cryptography have a duty to speak out.

The focus on terrorism starved the rest of law enforcement. About half of all crime is now online, and yet the resources devoted to fighting it are tiny. Many scammers operate with impunity.

There are further problems around censorship. Concerns about online abuse are real, but this is a difficult area. Abuses range in seriousness from videos of murder and child rape at the top end, down through hate speech, rape threats and cyber-bullying to news manipulation which, at scale, can be toxic. Countries are starting to pass laws requiring firms like Facebook to do the censorship for them, which causes many tensions. The companies don’t like the extra costs, and thoughtful citizens don’t like the idea of censorship being in the hands of private monopolies – or the idea that everything we upload, from pictures and videos to private messages, is filtered. So the firms have an incentive to redesign their systems so that they’re harder to abuse; Facebook, for example, claims to be rebuilding its systems to focus more on groups, which are harder for extremists to game, and to make more use of end-to-end encryption, so it can claim ignorance. Such arguments cut no ice in major incidents, such as when a shooter killed people at two mosques in Christchurch, New Zealand, in March 2019 and used Facebook to share live video of the crime. This forced the company to start censoring white supremacist groups, a politically sensitive task it had previously avoided [1913]. The COVID-19 pandemic led the company to rapidly do many things that the industry had previously denounced as impossible, undesirable or impractical: removing misinformation, banning exploitative ads and pushing official advice [984]. The tensions between privacy and censorship may continue to work out in unpredictable ways.

Privacy regulation is already complex. U.S. laws are fragmented, with federal laws on specific topics such as health data and video rentals and the FTC punishing firms that violate their published privacy policies, while state laws drive security-breach disclosure. Europe is very different: the General Data Protection Regulation provides a comprehensive framework, backed up by human-rights law that has been used to strike down laws on surveillance. The overall effect, from the viewpoint of the IT industry, is that Europe is becoming the world’s privacy regulator; Washington doesn’t care, and nobody else is big enough to matter. (There are strong signs that this regulatory power will be extended steadily to safety as well, although we’ll leave that to the chapter on assurance.)

In this chapter, I’m going to discuss the evolution of surveillance, then look at terrorism before discussing censorship and privacy regulation, and finally trying to put the whole thing in context.

## 26.2 Surveillance

The 2010s saw a huge increase in technical surveillance, not just by governments but also by commercial firms monitoring our clickstream and location history in order to target ads better – described by Shoshana Zuboff as ‘Surveillance Capitalism’ [2075]. The two interact in various ways. In some countries, like the USA, law enforcement and intelligence agencies don’t just get information from their own collection systems but use warrants to get it from firms like Google and Facebook too. In others, like China, these firms are banned because they refused to give complete access to the authorities; in others, like Iran and Syria, the police agencies just beat people’s passwords out of them, or phish their friends, or hack their phones.

This is a huge subject, and all I can reasonably provide is a helicopter tour: to place surveillance in its historical context, sketch what’s going on, and provide pointers to primary sources.

### 26.2.1 The history of government wiretapping

Rulers have always tried to control communications. In classical times, couriers were checked at customs posts, and from the Middle Ages, many kings either operated a postal monopoly or granted it to a crony. The letter-opening and codebreaking facilities of early modern states, the so-called *Black Chambers*, are described in David Kahn’s history, ‘The Codebreakers’ [1001].

When electronic communications came along, governments tried to keep control. In most of Europe, the telegraph service was set up as part of the post office and owned by the government; in Britain, the telegraph industry was nationalized by Gladstone in 1869. A profusion of national rules caused so much trouble that the *International Telegraph Union* (ITU) was set up in 1865 to standardise things [1818]. In the USA, Western Union was the first nationwide industrial monopoly and dominated the market through the nineteenth century. Union and Confederate soldiers tapped each others’ telegraph lines, and the New York Police Department started wiretapping operations in 1895.

The invention of the telephone led to tussles over privacy. In the USA, the Supreme Court ruled in 1928 in *Olmstead vs United States* that wiretapping didn’t violate the fourth amendment provisions on search and seizure as there was no physical breach of a dwelling; Justice Brandeis famously dissented. In 1967, the Court reversed itself in *Katz vs United States*, ruling that the amendment protects people, not places. The following year, Congress legalized Federal wiretapping (in ‘title III’ of the Omnibus Crime Control and Safe Streets Act) following testimony on the scale of organized crime. In 1978, following an investigation into the Nixon administration’s abuses, Congress passed the Federal Intelligence Surveillance Act (FISA), which controls wiretapping for national security. In 1986, the Electronic Communications Protection Act (ECPA) relaxed the Title III warrant provisions. By the early 1990s, the spread of deregulated services from mobile phones to call forwarding had started to undermine the authorities’ ability to wiretap, as did technical developments such as adaptive echo cancellation in modems.

## 26.2. SURVEILLANCE

---

So the 1994 Communications Assistance for Law Enforcement Act (CALEA) required all communications companies to make their networks tappable in ways approved by the FBI. By 1999, over 2,450,000 telephone conversations were legally tapped following 1,350 court orders [634, 1257]; by 2017 the number of wiretap orders had almost tripled to 3,813, but 94% were against portable devices such as cellphones [1927]<sup>1</sup>. A further 1,598 orders were granted in whole or in part by the Foreign Intelligence Surveillance Court (FISC) while 26 were denied.

Even before 9/11, some analysts believed that there were at least as many unauthorized wiretaps as authorized ones [558]. First was phone company collusion: while a phone company must give the police access if they present a warrant, in many countries they are also allowed to help – and there have been many reports over the years of phone companies being cosy with the government. Second, there's intelligence-agency arbitrage: if the NSA wants to wiretap an American citizen without a warrant they can get an ally to do it, and return the favour later. It was said, for example, that Margaret Thatcher used the Canadian intelligence services to wiretap ministers suspected of disloyalty [728]. Such practices were denied by the agencies for years but the Snowden leaks showed them to be reality; for example, the NSA got GCHQ to tap the links between Google data centres, as I described in 2.1. Third, in some countries, wiretapping is uncontrolled if one of the subscribers consents – so calls from phone boxes are free to tap (the owner of the phone box is the legal subscriber). Companies may wiretap their staff to detect fraud and voluntarily pass the product to the police or security agencies; there was a scandal in the UK when it emerged that the security services were involved in an unlawful, clandestine scheme to blacklist construction industry staff who had tried to organise unions [658]. Finally, in many countries, the police get hold of email and other stored communications by subpoena rather than warrant. They did this in America too before a court stopped the practice in 2007 [1161] – but the judgment didn't stop private actors such as bounty hunters and bail agents buying phone location histories from data aggregators [489].

But even if the official figures have to be doubled or tripled, democratic regimes use wiretapping very much less than authoritarian ones. The surveillance leader now is China, which uses pervasive technical monitoring in regions with minority populations such as Xinjiang and Tibet, with surveillance cameras mounted over street corners, mosques and schools hooked up via face-recognition software to databases recording who was seen where and when. There are also intrusive physical measures ranging from frequent street checkpoints, through billeting party members in the homes of minority families, to mass incarceration in labour camps [1110].

The incidence of wiretapping has also been highly variable within and between democracies. In the USA, for example, only about half the states use it, and for much of the 20th century most taps were in the ‘Mafia’ states of New York, New Jersey and Florida (though Nevada and California have now caught up) [1927]. There is similar variation in Europe. Wiretaps are very common in the Netherlands: they have up to 1,000 taps on the go at once with a tenth of

---

<sup>1</sup>The relevant law is 18 USC (US Code) 2510–2521, while FISA's regulation of foreign intelligence gathering is now codified in US law as 50 USC 1801–1811.

## *26.2. SURVEILLANCE*

---

America's population [356]. In a Dutch homicide investigation, it's routine to tap everyone in the victim's address book for a week to monitor how they react to the death. The developed country with the most wiretaps is Italy, thanks to its history of organised crime [1160]. In the UK, domestic wiretaps are supposed to need a ministerial warrant, and cannot be used in evidence; so the police use room bugs and computer exploits instead. If you can root a gangster's phone or laptop you can record, and mail home, everything said nearby, whether it's said to someone in the same room, or on a call. International calls have been routinely recorded for decades and stored for some days to weeks in case they turn out to be of interest, a model followed by many other countries; for example, after the Mumbai massacre in 2008, India could dig out recordings of phone calls the terrorists made to their controllers in Pakistan.

Automation is shifting the costs of wiretapping from per-call labour costs to one-off capital costs. Before CALEA was introduced, in 1993, US police agencies spent only \$51.7 million on wiretaps – perhaps a good estimate of their value before the issue became politicised [862]. The implementation of CALEA cost over \$500m, and that was before it was extended to VOIP in 2007. VOIP was harder: “The paradigm of VoIP intercept difficulty is a call between two road warriors who constantly change locations and who, for example, may call from a cafe in Boston to a hotel room in Paris and an hour later from an office in Cambridge to a giftshop at the Louvre” [220]. During the 2010s things became harder still as people moved from physical platforms, such as their cellphone, to virtual platforms such as Facebook, Skype and Signal. So the trend for policymakers has been to make capital investments that cut the marginal costs of access. For example, ten years ago, if the UK police were investigating three similar rapes, they might have had to pay the phone companies thousands of pounds to assemble cellsite dumps so they could look for any mobile phones that were present at all three locations. Now, after spending hundreds of millions and getting several laws passed, they have access to databases of mobile phone locations, and all it takes is a database query. This changes the nature of both police and intelligence work.

The USA also changed its laws to facilitate bulk surveillance. 43 days after the 9/11 attacks, Congress passed the Patriot Act, which allowed increased access by law enforcement to stored records (including financial, medical and government records), ‘sneak-and-peek’ searches of homes and businesses without the owner’s knowledge, and the use by the FBI of National Security Letters to get access to financial, email and telephone records.

But this was not enough for the agencies. In December 2005, the New York Times revealed that President Bush had signed a secret 2002 order mandating warrantless wiretapping of US residents suspected of terrorism, contrary to law [1606]. In 2006, USA Today revealed that the NSA had covertly obtained full call-data records (CDRs) for the 200m customers of AT&T, Verizon and BellSouth, the nation’s three biggest phone companies. The CDR program had been started by the DEA in 1992 under the older President Bush, and targeted calls by Americans to and from certain countries; it was ramped up after 9/11, when his son authorised the collection of CDRs for all internal US calls too [877]. Qwest did not cooperate, because its CEO at the time, Joe Nacchio, maintained that the NSA needed a court order. The NSA put pressure on

Qwest by threatening to withhold classified contracts, so Qwest's lawyers asked the NSA to take its proposal to the FISA court. They refused, saying the court might not agree with them. It's since emerged that they had put pressure on Qwest to hand over data even before 9/11 [768]. In October 2007, Verizon admitted to senators that it had given the FBI second-generation call data on its customers against national security letters on 720 occasions since 2005 [1376]. In November 2007, the Washington Post revealed that the NSA had tapped a lot of purely domestic phone calls and traffic data, and had also tapped AT&T's peering centre in San Francisco to get access to Internet traffic [1377]. After two years of debate, Congress amended FISA to grant retroactive immunity to phone companies who cooperated with unlawful wiretapping, and to change the law so that the NSA no longer needs even a FISA warrant to tap a call if one party's believed to be outside the USA or a non-US person. (This split both parties, with Senators Obama and Feinstein supporting the amendment while Senators McCain, Biden, Reid, Leahy and Clinton opposed it.)

### 26.2.2 Call data records (CDRs)

Historically, more police communications intelligence has come from the analysis of telephone call data records and other metadata rather than wiretaps. We discussed in the chapter on telecoms security how the police use such data to trace networks of criminal contacts, and how criminals respond by burying their signals in innocuous traffic using techniques such as pre-paid mobile phones and PBX hacking.

Again, this is nothing new. Rulers have long used their control over postal services to track the correspondents of suspects, even when the letters weren't opened. The introduction of postage stamps in 1840 was an advance for privacy as it made it much easier to send a letter anonymously. Some countries got so worried about the threat of sedition that they passed laws requiring a return address to be written on the back of the envelope. The development of the telegraph, on the other hand, was an advance for surveillance; as messages were logged by sender, receiver and word count, traffic totals could be compiled and were found to be an effective indicator of economic activity [1818]. The First World War taught the combatants how much intelligence could be gleaned from measuring the volume of enemy radio traffic, even when it couldn't conveniently be deciphered [1001, 1380]. Later twentieth-century conflicts reinforced this.

When I wrote the first edition of this book, I noted that the USA had 1,329 wiretap applications approved in 1998, while there were 4886 subpoenas (plus 4621 extensions) for *pen registers* (devices which record all the numbers dialed from a target phone line) and 2437 subpoenas (plus 2770 extensions) for *trap-and-trace* devices (which record the calling line ID of incoming calls, even if the caller tries to block it). Law-enforcement agencies were also starting to switch in the 1990s to using subpoenas for the call-detail records in the phone companies' databases. Bell Atlantic, for example, responded to 25,453 subpoenas or court orders for toll billing records of 213,821 of its customers in 1989–92, while NYNEX processed 25,510 subpoenas covering an unrecorded number of customers in 1992 alone [402]. Scaled up across the seven Baby Bells, this suggests that perhaps half a million customers were having their records

## 26.2. SURVEILLANCE

---

seized every year in the 1990s, and that traffic data were collected on perhaps a hundred times as many people as were subjected to wiretapping.

Statistics went dark after 9/11, during the period of unlawful collection, although the NSA did reveal in 2006 that it wanted “to create a database of every call ever made within the nation’s borders” so it could map the entire US social network for the War on Terror [395]. After Snowden revealed in 2013 that it had built databases of pretty well all traffic data for all communications worldwide, Congress passed the Freedom Act in 2015 and we started to get an annual Statistical Transparency Report from the Director of National Intelligence. The April 2018 report gives some figures for 2017; these relate only to national-security matters, but give some feel for the balance between content and traffic data. Wiretap warrants are stable at about 1,500 per year in the USA (targeting about 300 US persons and 1000 others), as well as a rising number of targets overseas – 106,469 in 2016 and 129,080 in 2017. In addition, there were 7,512 US residents whose communications content was retrieved (e.g. subpoenas for email) while 16,924 residents had non-content (such as traffic data) retrieved, along with 56,064 non-residents. There were also 87,834 collected business records, which might include records of which subscriber was using which IP address [1464].

Now the US intelligence community only considers a communication to be ‘intercepted’ when a human analyst looks at it; analysis by software doesn’t count (UK law counts both). As I described in Section 23.3.1, the usual procedure when hunting for suspects is contact chaining, also known as a ‘snowball search’. If someone blows themselves up in a terror attack, analysts will use software that looks at all the people they communicated with, and then everyone these direct contacts communicated with, and exceptionally even out to a third degree of separation. The standard depth-two search typically gives some tens of thousands of indirect contacts. These contacts are then compared against millions of names on various suspect lists – religious extremists, right-wing hate groups, organised crime – and the analysts then home in on the links with any known suspects. (The analogy is rolling a snowball downhill, then melting it and seeing what dirt you find in the bottom of the bucket.) So the analyst may look at only half a dozen people who were in contact with the dead terrorist and also with members of some religious group, but tens of thousands of innocent people had their call data records looked at by the software. The DNI report estimates that in 2017, 534,396,285 call data records (CDRs) were examined automatically in this way – a large increase from 151,230,968 in 2016.

Yet there was a long debate in Congress about allowing Section 215 of the Patriot Act (as amended by FISA) to lapse. This was the section that allows the bulk collection of CDRs [416]; the NSA has said that it doesn’t want it. The bulk collection of communications data was one of the matters highlighted by Ed Snowden that sparked the most controversy. On June 8th 2013, the press disclosed Boundless Informant, an NSA visualisation tool that shows a heat map of where metadata are collected for both voice and computer communications; in a 30-day period ending in March 2013, 3 billion records were collected from 504 sources (or SIGADs). Although the most intensive collection was in the Middle East, Snowden said that more records were collected on Americans in America than on Russians in Russia [756]. On another reading of the material,

## **26.2. SURVEILLANCE**

---

Boundless Informant collected 3 billion phone records via US telecommunications providers, plus a further 97 billion emails and 124 billion phone calls round the world [816, p. 92]; overall, 20 billion events a day are collected [816, p. 98]. However, a declassified report revealed that while the NSA call-data record program in the USA cost over \$100m, it produced only two leads and one significant investigation [1656]. In 2020, the clause was allowed to lapse in March but reinstated in May; the politics was messy. Susan Landau and Asaf Lubin explained that with 4g mobile networks, traditional CDRs don't identify both the caller and the called party reliably any more [1126]. In any case, the action is shifting from the plain old telephone system to messaging systems.

As for targeted collection in specific criminal investigations, under 18 USC 3123 [1925], the investigative officer merely has to certify to a magistrate 'that the information likely to be obtained by such installation and use is relevant to an ongoing criminal investigation'. This can be any crime – felony or misdemeanour – and under either Federal or State law. Since CALEA, warrants are still required for such communications data as the addresses to which a subscriber has sent e-mail messages, but basic toll records can be obtained under subpoena – the subscriber need not be notified, and there is no court supervision once the order has been made. The US Department of Justice is required by law to publish statistics for its non-national-security law-enforcement activities but appears reluctant to do so; the American Civil Liberties Union (ACLU) extracted figures for 2011–12 only after freedom-of-information (FOI) litigation, which revealed that the combined number of original orders for pen registers and trap and trace devices used to spy on phones increased by 60%, from 23,535 in 2009 to 37,616 in 2011 [765]. I've been unable to find anything more recent.

Bulk access to traffic data has been also led to serious political tussles in Europe. The UK pushed through a Data Retention Directive in the European Union in 2006, under which member states had to store telecommunications data – including IP address and timing of every email, phone call and text message sent or received – for between 6 months and 24 months, and make all this available to law enforcement and intelligence agencies. The Directive was struck down in 2014 by the European Court of Justice after Digital Rights Ireland brought a lawsuit arguing that blanket data collection violated the EU Charter of Fundamental Rights.

In Britain, targeted access to communications data requires only a notice from a senior police officer to the phone company or ISP, not a warrant; and data can be provided to a wide range of public-sector bodies, just as in the USA. Following the Data Retention Directive, the Blair government wanted to centralise things; it argued that the police needed a 'communications database' and pushed a law to establish it. Fate intervened when some wicked person stole a copy of all the expenses claims filed by members of parliament and sold it to the Daily Telegraph. It turned out that numerous ministers and others had been making embarrassing claims; several honourable members went to jail, and most of the well-known politicians in Britain had to make repayments. (I told the tragic tale of the Home Secretary, Jacqui Smith – who had been promoting the communications database – in section 8.6.5 above.) We heard nothing more of the communications database until Ed Snowden told us in 2013 that they'd just built it anyway, even without parliamentary approval.

After the European Court struck down data retention, and Snowden revealed some highly objectionable activities by GCHQ, the UK passed the 2014 DRIP Act to assert that what GCHQ had been doing was legal after all. It was clear that the European Court would object eventually, but some breathing space was needed and the Act gave this (it had a two-year sunset clause; Prime Minister Cameron's liberal coalition partners wouldn't give him any more). Eventually, in the wake of the Brexit vote, Parliament passed the Investigatory Powers Act, which pretty well enables GCHQ to do as it pleases and compel any company in the jurisdiction to assist it. The interesting action in the future will be, first, the extent to which the large US firms will help, and second, the line to be taken by the European Court of Human Rights<sup>2</sup>. I'll return to these issues later.

### **26.2.3 Search terms and location data**

It has become ever clearer over the past 20 years that the regulation of surveillance that evolved in the phone-company era is not really fit for purpose in the era of the Internet. Back then, you got either a full wiretap and recorded the content, or made do with traffic data from call data records. But as things moved online, communications data and content got all mixed up, as what's content at one level of abstraction is often communications data at the next. Some people might think of a URL as just the address of a page to be fetched, but a URL such as <http://www.google.com/search?q=marijuana+cultivation+UK> contains the terms entered into a search engine as well as the search engine's name. Clearly, some policemen would like a list of everyone who submitted such an enquiry. This became a live issue in 1999, when the UK government modernised its surveillance law; academics, NGOs and industry managed to get a 'Big Brower Amendment' into the resulting Regulation of Investigatory Powers Act of 2000 defining traffic data as the information necessary to identify the communicating machine; for URLs, this means everything up to the first slash.

In the USA, the Department of Justice issued a subpoena to a number of search engines to hand over two full months' worth of search queries, as well as all the URLs in their index, claiming it needed the data to bolster its claims that the Child Online Protection Act did not violate the constitution and that filtering could be effective against child pornography. (Recall we discussed in section 11.2.3 how when AOL released some search histories, a number of them were easily identifiable to individuals.) AOL, Microsoft and Yahoo quietly complied, but Google resisted. A judge finally ruled in 2006 that the Department would get no search queries, and only a random sample of 50,000 of the URLs it had originally sought [2035].

The next issue was mobile-phone location data, which ended up being treated differently in different jurisdictions. In Britain, all information about the location of mobile phones counts as traffic data, and officials get it easily; but in the USA, the Court of Appeals ruled in 2000 that when the police get a warrant for the location of a mobile, the cell in which it is active is sufficient, and that to require triangulation on the device (an interpretation the police had wanted)

---

<sup>2</sup>Britain's departure from the EU will let it escape the European Court of Justice, which is an EU institution, but not the Court of Human Rights, as this is an institution of the Council of Europe

would invade privacy [1926]. Also, even cell-granularity location information would not be available under the lower standards applied to pen-register subpoenas. Yet despite these rules, there were massive leaks of information. It emerged in 2019 that AT&T and Sprint had both been selling their customers' location information to data brokers for years, including not just cellsite data but GPS; and this had routinely been bought by bounty hunters and bail agents to track defaulters [489]. Location data is now being collected by many governments with a view to tracing contacts of COVID-19 sufferers and epidemiology more generally. It's also collected by lots of apps: the Untappd beer-rating app is run by millions of beer drinkers who record hundreds of time-stamped locations, which enabled journalists to track US military and intelligence personnel around the world [1538].

#### 26.2.4 Algorithmic processing

The analysis of call data is only one aspect of a much wider issue: law-enforcement matching of bulk datasets. The earliest serious use of multiple-source data appears to have been in Germany in the late 1970s to track down safe houses used by the Baader-Meinhof terrorist group. Investigators looked for rented apartments with irregular peaks in utility usage, and for which the rent and electricity bills were paid by remote credit transfer from a series of different locations. This worked: it yielded a list of several hundred apartments among which were several safe houses. The tools to do this kind of analysis are now shipped with a number of the products used for traffic analysis and for managing major police investigations. The extent to which they're used depends on the local regulatory climate; there have been rows in the UK over police access to databases of the prescriptions filled by pharmacists, while in the USA doctors are alarmed at the frequency with which personal health information is subpoenaed from insurance companies by investigators. There are also practical limits imposed by the cost of understanding the many proprietary data formats used by commercial and government data processors. But it's common for police to have access at least to utility data, such as electricity bills which get trawled to find marijuana growers, and there's little to stop them using commercially available data such as feeds from credit reference agencies.

Since AlphaGo beat Lee Sedol in 2016, there's been a host of machine-learning startups, and quite a few aim to make law enforcement easier one way or another. But it's not as easy as it looks. Terrorists are so rare as a percentage of the population that any tests you use to 'detect' them would require extraordinary specificity if you're not to drown in false positives. Combining multiple sensors is hard, and if you're looking for a needle in a haystack, it's not always smart to build a bigger haystack. As Jeff Jonas, once the chief scientist at IBM's data-mining operation, put it, "techniques that look at people's behavior to predict terrorist intent are so far from reaching the level of accuracy that's necessary that I see them as nothing but civil liberty infringement engines" [757].

### **26.2.5 ISPs and CSPs**

The 2000s saw rapid growth of intrusive surveillance at both Internet Service Providers (ISPs) and Communications Service Providers (CSPs – firms like Google and Yahoo). Tapping data traffic at an ISP is harder than voice used to be; there are many obstacles, such as transient IP addresses given to most customers and the increasingly distributed nature of traffic. In the old days (say 2002), an ISP might have had modem racks, and a LAN where a wiretap device could be located; nowadays many customers come in via DSL, and providers use switched networks that often don't have any obvious place to put a tap. The ISP simply became the natural control point.

Many countries now have laws requiring ISPs to help, and the usual way to do it at a large ISP is to have equipment already installed that will send copies of packets of interest (or NetFlow records) to a separate classified network. The FBI's system, DCSNet, is very slick – allowing agents point-and-click access to traffic and content from participating phone companies [1761]. (Information about which companies have been brought onboard is closely held, but smart bad guys use small ISPs.) And things often go wrong because the police don't understand ISPs; they subpoena the wrong things, or provide inaccurate timestamps so that the wrong user is associated with an IP address. For an analysis of failure modes, see Clayton [442].

The smartphone revolution has changed the natural control point from the ISP to the CSP. A modern criminal might get up, check his messages on Gmail or WhatsApp using his home wifi, then get on a bus into town and do the same using his 3G or 4G data connection, then perhaps use wifi at a Starbucks or a public library ... and in none of these cases does a wiretap at the ISP tell anything much beyond the fact that a particular service has been used. As the traffic to that communications service is encrypted, the police have to serve paperwork on the service to get anywhere. This is what led the FBI to set up the Prism system, whereby intelligence agencies can get customer data from Google, Yahoo, Apple, Microsoft, Facebook and others at the press of a button. It is also what led the UK, in its 2016 Investigatory Powers Act, to grant itself the power to order any company to do anything it physically can in order to assist law-enforcement of intelligence investigations. More and more countries are passing such laws, which put the service providers in conflict with other countries' laws.

One big flashpoint is the tension between EU privacy and data-protection law, which requires due process for privacy infringement, and US surveillance law which demands that US firms hand over foreigners' data on demand. But there are many more. Google left China rather than give the police unfettered access to all user data. And as a senior Google executive told me, 'If a family court in India orders you to hand over the Gmail of someone who lives in Canada and imposes a lifelong secrecy order, how do you simultaneously employ people in India, and give believable assurances of privacy to people in Canada?'

Finally, there are lots of issues around the much richer data available from CSPs like Facebook, which not only collect highly sensitive data at scale but enable sensitive facts to be deduced from traffic data in ways that were not previously possible. As I discussed in section 11.2.5, Michal Kosinski and colleagues

figured out that he could tell whether someone was straight or gay from four Facebook likes [1086], after which some of his colleagues collected Facebook data at industrial scale and weaponised it for political campaigning, leading to the Cambridge Analytica scandal when it was discovered that social-network data had been used in 2016 to intervene unlawfully and at scale in both the Brexit referendum in the UK and the presidential election in the USA. What sort of controls should there be on the use of social analysis methods by law-enforcement and intelligence agencies, or for that matter by public-health agencies? (We'll return to the broader issues raised by these techniques later.)

### **26.2.6 The Five Eyes' system of systems**

We discussed the technical meat of the Snowden revelations in 2.2.1. These did not come entirely from the blue; there had been many previous disclosures about signals intelligence collection. David Kahn's influential history of cryptography sets the scene by describing what happened up till the start of World War 2 [1001]. An anonymous former NSA analyst, later identified as Perry Fellwock, then revealed the scale of NSA operations in 1972 [674]. "Information gathering by NSA is complete," he wrote. "It covers what foreign governments are doing, planning to do, have done in the past: what armies are moving where and against whom; what air forces are moving where, and what their capabilities are. There really aren't any limits on NSA. Its mission goes all the way from calling in the B-52s in Vietnam to monitoring every aspect of the Soviet space program."

While Fellwock's motive was opposition to Vietnam, the next major whistleblower was a British wartime codebreaker, Frederick Winterbotham, who wanted to write a memoir of his wartime achievements and, as he was dying, was not bothered about prosecution. In 1974, he revealed the Allies' success in breaking German and Japanese cipher systems during that war [2031], which led to many further books on World War 2 signals intelligence (Sigint) [438, 1002, 2007]. Thereafter there was a slow drip of revelations by investigative journalists, quite a few of whose sources were concerned about corruption or abuse of the facilities by officials monitoring targets they should not have, such as domestic political groups. Whistleblower Peg Newsham revealed that the NSA had illegally tapped a phone call made by Senator Strom Thurmond [373, 374]. James Bamford pieced together a lot of information on the NSA from open sources and by talking to former employees [160], while New Zealand journalist Nicky Hager [849] dug up a lot of information following the New Zealand intelligence community's failure to obey an order from their Prime Minister to downgrade intelligence cooperation with the USA.

The first high-profile exposé of US economic espionage was made in a 1999 report to the European parliament [644], which was concerned that after the collapse of the USSR, European Union member nations were becoming the NSA's main targets [377]. By then, people who paid attention were aware that data, faxes and phone calls get collected at a large number of nodes ranging from where international communications cables land in friendly countries (or are tapped clandestinely underwater), through observation of traffic to and from commercial communications satellites and special Sigint satellites that collect

## 26.2. SURVEILLANCE

---

traffic over hostile countries, to listening posts in member states' embassies [644].

During the Cold War, much of the effort was military, aimed at understanding Soviet radar and communications, and at gaining a decisive advantage in location, jamming and deception. Without an ability to conduct electronic warfare, a modern state is not competitive in air or naval warfare or even in tank battles. Most of the personnel at NSA were military, and its director has always been a serving general or admiral. A lot of effort still goes into understanding the signals of potential adversaries.

One might question whether this huge worldwide system of systems still gives value for money. Politicians have justified its budgets since 9/11 in terms of terrorism, and there have indeed been some successes against terrorists – notably the arrest of an alleged 9/11 terrorism planner after he used a mobile phone SIM from a batch bought by a known terrorist in Switzerland. But electronic warfare against insurgents in Iraq proved less productive, as I discussed in Chapter 19. And it's clear that more effort should have been put into human intelligence. In an article published just before 9/11, an analyst wrote "The CIA probably doesn't have a single truly qualified Arabic-speaking officer of Middle Eastern background who can play a believable Muslim fundamentalist who would volunteer to spend years of his life with shitty food and no women in the mountains of Afghanistan. For Christ's sake, most case officers live in the suburbs of Virginia. We don't do that kind of thing." Another put it even more bluntly: "Operations that include diarrhea as a way of life don't happen" [758]. Nearly two decades after the start of the wars in Afghanistan, Iraq, Syria and North Africa, we haven't trained enough soldiers to carry a basic conversation in Arabic, Dari or Pushtu.

Although other countries may complain about US Sigint collection, for them to moralise about it is hypocritical. Other countries also run intelligence operations, and are often much more aggressive in conducting economic and other non-military espionage. The real difference between the Five Eyes countries and the others is that no-one else has built the 'system-of-systems'. Indeed, there are network effects in Sigint as elsewhere: while non-aligned countries like India were happy to buy their warplanes from the old Soviet Union, they nowadays tend to share intelligence with the USA, as it has a much bigger network than the Russians or the Chinese [84]. The Snowden documents reveal NSA information sharing with over 60 other countries.

My own view is that, like the armed forces of which they are often a part, signals intelligence agencies are both necessary but potentially dangerous. An army can be a good servant but is likely to be an intolerable master. The issue is not whether such resources should exist, but how they are held accountable. In the USA, hearings by Senator Church in 1975 detailed a number of abuses such as the illegal monitoring of US citizens [423]; this led to FISA. The Snowden revelations in turn led to action by all three arms of the US government, albeit of limited effect<sup>3</sup>.

The structural problems remain, though. The NSA is responsible for both

---

<sup>3</sup>President Obama set up the NSA review group and accepted most of its recommendations, but his positive work was undone by President Trump. Congress passed the USA Freedom Act which imposed some limits on the bulk collection of communications data on US residents by US agencies. Chief Justice Roberts made some changes to the FISA court.

attack and defence, and defence tends to play second fiddle. Imagine that you're the Director of the NSA, and one of your engineers comes to you with a cool new zero-day exploit of Windows. Do you tell Microsoft, thereby protecting 300m Americans, or do you keep it secret, so you can attack 1.2bn Chinese? Stated in those terms, the answer is obvious. This *equities issue* is the one issue on which President Obama declined to follow the advice of the NSA review group. The group recommended that in almost all cases, vulnerabilities that come to the attention of the NSA should be reported to vendors for fixing; the NSA prefers to stockpile them instead. Indeed it has a \$100m a year budget for Bullrun, a program to insert them into commercial products by means fair and foul, as discussed in section 2.2.1.5. And when bugs occur naturally, the NSA uses them where it can; it was reported in 2014, for example, that the hugely disruptive Heartbleed bug in SSL had been exploited by the NSA for two years before it was discovered independently and fixed [2065].

In some countries things are cleaner: in both France and Germany, there are separate agencies for attack and defence. But in most countries, the oversight of intelligence isn't even discussed. In the UK, it's only the European courts that forced the government to admit to the scale of surveillance, and to legislate some controls on it. New cases continually highlight excessive collection, by both electronic and human methods. In 2019, the European Court of Human Rights ordered the UK police to delete from its 'extremism' database the records of some 60 demonstrations attended by John Catt, a 94-year-old protester with no criminal record – a verdict applauded even in the conservative press [2024].

That is the high-level picture of how surveillance has evolved over the past few decades. Another aspect is scale. Cross-border bandwidth increased from 11Tbit/sec in 2007, when the systems described by Ed Snowden were being built, to 704Tbit/sec in 2017; this firehose creates yet more pressure for the agencies to collect traffic from CSPs or other edge systems rather than from ISPs or the backbone, as they can target the collection much better. The resulting pressure for government access to data is remarkably similar to the pressure for government access to cryptographic keys in the 1990s, which was a formative experience for many governments (as well as for industry and civil society) on issues of surveillance and technology policy.

### 26.2.7 The crypto wars

Technology policy during the 1990s was dominated by acrimonious debates about *key escrow* – the Clinton administration doctrine that anyone who encrypted data should give the government a copy of the key, so that the civilian use of cryptography would not interfere with intelligence gathering.

I was involved as one of the academics whose research and teaching was under threat from the proposed controls, and in 1998 I was one of the people who set up the Foundation for Information Policy Research, a UK Internet-policy think-tank, which wrestled with crypto policy, export policy, copyright and related issues. In 2003 we set up European Digital Rights (EDRi) along with other European NGOs to campaign on these issues in Brussels. In the next few sections I'll lay out a brief background to the crypto wars, and then discuss how governments have failed to get to grips with the Internet.

### 26.2.7.1 The back story to crypto policy

Many countries made laws in the mid-19th century banning the use of cryptography in telegraph messages, and some even forbade the use of languages other than those on an approved list. Prussia went as far as to require telegraph operators to keep copies of the plaintext of all messages [1818]. Sometimes the excuse was law enforcement – preventing people obtaining horse race results or stock prices in advance of the ‘official’ transmissions – but the real concern was national security. This pattern was to repeat itself again in the twentieth century.

After the immense success that the Allies had during World War 2 with signals intelligence, the UK and US governments agreed in 1946 to continue intelligence cooperation. This ‘BRUSA agreement’ was joined by Canada in 1948 and by Australia and New Zealand in 1956, giving the ‘Five Eyes’ partnership in signals intelligence. They decided to prevent the proliferation of cryptographic equipment and know-how. Until the 1980s, about the only vendors were companies selling into government markets, who could mostly be trusted not to do anything overseas which would upset their major customers at home. This was reinforced by export controls that were operated “in as covert a way as possible, with the minimum of open guidance to anyone wanting, for example, an export licence. Most things were done in behind-the-scenes negotiation between the officials and a trusted representative of the would-be exporter.” [206]

In these negotiations, the authorities would try to steer applicants towards using weak cryptography where possible, and where confronted with a more sophisticated user would try to see to it that systems had a ‘back door’ (known in the trade as a *red thread*) which would give access to traffic. Anyone who tried to sell decent crypto domestically could be dissuaded by various means. If they were a large company, they would be threatened with loss of government contracts; if a small one, they could be strangled with red tape as they tried to get licenses and product approvals. The upshot was that most governments used weak crypto, and the NSA could break it with ease. But this wasn’t the whole story, as we learned in the Bühler case.

Hans Bühler worked as a salesman for the Swiss firm Crypto AG, a leading supplier of cryptographic equipment to governments without the technical capability to build their own. He was arrested in 1992 in Iran when the authorities figured out that the Iraqis had been reading their traffic during the Iran-Iraq war; they accused him of selling them cipher machines which had been tampered with so that the NSA could get at the plaintext. Crypto AG paid 1.44 billion Rials – then about a million US dollars – to bail him, but fired him once he got back to Switzerland. Bühler then alleged on Swiss radio and TV that the firm was secretly controlled by the German intelligence services and that it had been involved in intelligence work for years [335]. One story was that when the founder of Crypto AG, Boris Hagelin, decided to retire, he contacted William Friedman, the NSA’s chief scientist; Friedman was a friend, and the US government had been a big customer, buying Hagelin machines during World War 2. Hagelin sold his company secretly to the NSA, which had it secretly controlled by German nominees. The equipment it sold was routinely red threaded [1205]. Crypto AG’s line was that these allegations were concocted by the NSA to

undermine the company, as it was one of the third world's few sources of cryptographic equipment. Bühler's story was told in a book by Res Strehle [1837]. It is now known that Crypto AG was run by the German Bundesnachrichtendienst in collaboration with the agencies of Denmark, Sweden, the Netherlands and France, and with the CIA. The backdoors in their equipment were used, for example, by the UK to decipher Argentinian communications during the Falklands war in 1982 – the outcome of which was “materially influenced, if not decided” by this operation [970].

#### **26.2.7.2 DES and crypto research**

Despite the poor quality of early banking cryptosystems, the NSA still worried in the seventies that the banking sector might evolve good algorithms that would escape into the wild. Many countries were still using rotor machines or other equipment that could be broken using the techniques developed in World War 2. How could the banking industry's thirst for a respectable cipher be slaked, not just in the US but overseas, without this cipher being adopted by foreign governments and driving up the costs of intelligence collection?

The solution was the Data Encryption Standard (DES). At the time, as I mentioned in section 5.4.3.2, there was controversy about whether 56 bits were enough. We now know that this was deliberate. The NSA did not at the time have the machinery to do DES keysearch; that came later. But by giving the impression that they did, they managed to stop most foreign governments adopting it. The rotor machines continued in service, in many cases reimplemented using microcontrollers; Crypto AG and other biddable vendors continued to thrive; and the traffic continued to be harvested. Foreigners who encrypted their important data with such ciphers merely marked that traffic as worth collecting.

A second initiative was to undermine academic research in cryptology. In the 1970s this was done directly by harassing the people involved; by the 1980s it had evolved into a subtler strategy. While the Pentagon funded research into computer security, it tried to divert crypto research into theoretical channels and claimed that more practical published research work was all old hat: ‘we did all that stuff thirty years ago; why should the taxpayer pay for it twice?’ The insinuation that DES may have had a ‘trapdoor’ inserted into it fitted well with this playbook. A side effect we still live with is that the crypto and computer security communities got separated from each other in the early 1980s as the NSA worked to sideline one and build up the other.

By the mid 1990s this line had become exhausted. Agency blunders in the design of key escrow systems debunked their story that they were way ahead of the rest of us in cryptology, and in any case the fight moved to a different battlefield.

#### **26.2.7.3 Crypto War 1 – the Clipper chip**

Crypto policy went mainstream in 1993 with the launch of the Clipper chip. After AT&T proposed the introduction to the US domestic market of an encrypting

telephone that would have used Diffie-Hellman key exchange and triple-DES to protect traffic, the NSA persuaded the Clinton administration to promote a different standard. This would use a classified block cipher, Skipjack, implemented in a tamper-resistant chip and with a protocol that made a spare ('escrowed') key available to the agencies to decrypt traffic. This 'Escrowed Encryption Standard' led to a public outcry; an AT&T computer scientist, Matt Blaze, found a protocol vulnerability in Clipper that defeated the escrow mechanism [258] and the proposal was withdrawn.

Several more attempts were made through the 1990s to promote the use of cryptography with government access to keys. Key escrow acquired various new names, such as *key recovery*; certification authorities which kept copies of their clients' private decryption keys became known as *Trusted Third Parties* (TTPs) – somewhat emphasising the NSA definition of a trusted component as one which can break security. In the UK, a key escrow protocol was introduced for the public sector [980], and this was used to try to get the private sector to adopt it as well; but we found a number of vulnerabilities in it too [115].

The pro-escrow people said that as crypto provided confidentiality, and confidentiality could help criminals, there needed to be some way to defeat it. The anti-escrow lobby started out by arguing that since crypto was necessary for privacy, there must not be a way to defeat it. Reality was more complex [56]. Most crypto applications are about authentication rather than confidentiality, so help the police rather than hindering them. As for criminals, they mainly require unobtrusive communications – and back in the 1990s, encrypting a phone call was a good way to bring attention to yourself. If you wanted to be unobtrusive, it was better to just buy a prepaid phone. As for privacy, most violations result from abuse of authorized access by insiders. Finally, a much more severe problem for policemen is to find acceptable evidence, for which decent authentication can also be helpful.

The debate got rapidly tangled up with export controls on weapons, the means by which cryptography was traditionally controlled. US software firms were not allowed to export products containing cryptography that was too hard to break, and this was also used as a means of controlling cryptography at home; Americans who put cryptography software on their websites were liable to prosecution for making it available to foreigners. A US software author, Phil Zimmermann, was hauled up before a grand jury for arms trafficking after a program he wrote – PGP – 'escaped' on to the Internet. He became a folk hero and made a fortune as his product grabbed market leadership. Others, such as Bruce Schneier, printed cryptographic algorithms in books as a way of exercising their constitutional right to free speech [1667]. The conflict became international: the US State Department tried hard to persuade other countries to control cryptography too (I'll go into more detail in Section 26.2.9 on export control below). Imposing American policy worldwide became one of the missions of Vice-President Gore (a reason why many tech people contributed to the Bush campaign in 2000).

The apparent resolution of Crypto War 1 came in two phases. In 1999, the European Union's Commissioner for the Single Market, Martin Bangemann, pushed through the Electronic Signature Directive, a law that banned the compulsory licensing of certification authorities. This undermined the demand from

the NSA and GCHQ that all private signing keys should be escrowed – not just decryption keys, but also signature verification keys. The Germans objected that escrowing signature keys would let the agencies not just read messages, but forge them too, undermining trust in electronic commerce and authentication generally. When the EU followed the German line rather than the British one, it followed that individuals could either use their signature keypairs for encryption, or to authenticate Diffie-Hellman keys and use those for encryption. European officials mollified the US administration by passing an export control regulation that extended EU export controls from physical goods to intangibles such as software, so that European firms faced the same export controls on cryptographic software as US firms [651].

Second, in 2000 when Al Gore was running for president and wanted to get Silicon Valley onside, the administration decided to call a halt. Meetings were held at the FBI offices in Quantico between the agencies and the tech majors, leading to an agreement that the agencies would no longer push for vulnerabilities to be inserted into products and systems. Instead, the agencies would exploit the many naturally-occurring vulnerabilities, and the NSA inveigled itself into the patching cycle. When a software vulnerability is reported to the CERT ecosystem, it finds its way to the CERT at the Software Engineering Institute in Pittsburgh, which is sponsored by the DoD. This shares it with the NSA and also reports it to the vendor for fixing. The patch cycle typically takes a month or two – sometimes more, if coordinating vulnerability disclosure and product testing is hard – giving the NSA a window to exploit the bug.

Those of us who were active in digital rights in Europe were generally pleased at the e-signature directive but appalled at intangible export controls; we set up European Digital Rights (EDRi) in 2003 to create a lobbying presence in Brussels, backed by dozens of individual NGOs in European countries. We thought that the surveillance issue had been largely settled and that future fight would be over issues like software copyright and data protection. In 2013, Ed Snowden showed us how wrong we'd been. The NSA and the other agencies had simply gone underground, and had been running a covert program called Bullrun with a budget of \$100m a year to undermine commercial cryptography, interfering with standards, implementations, supply chains and much else. But that came later.

One of the engineering lessons from Crypto War 1 is that doing key escrow properly is hard. Making two-party security protocols into three-party protocols increases the complexity and the risk of serious design errors, and centralizing the escrow databases creates huge targets; I discussed this in a paper ‘The Risks of Key Recovery, Key Escrow, and Trusted Third-Party Encryption’ that I wrote with ten other cryptographers and that became the most highly-cited reference on the subject [4]. Where escrow is required it’s usually better done with simple local mechanisms. In one army, every officer must write down his passphrase on a piece of paper, put it in an envelope, stamp it ‘Secret’ and hand it to his commanding officer, who puts it in his office safe. That way the keys are kept in the same place as the documents whose electronic versions they protect, and there’s no central database for an airplane to bomb or a spy to steal. But trying to automate this and scale it up leads to trouble. The UK government idea was that everyone’s private key would be generated from their email address using a

super-secret master key generated by GCHQ and kept in equipment controlled by their departmental security officer, so that both the department and GCHQ could decrypt traffic if they had to. The result was a clunky system that couldn't easily deal with the frequent changes of name as government departments were reorganised and renamed. The demand for customised central control leads to vast IT projects that run years late and millions over budget, or just never work at all. Problems providing officials with working email systems led to them using private accounts instead, and eventually the Cameron government more or less gave up; routine email in the Cabinet Office (the stuff below Top Secret) started to use a branded version of G Suite, the paid-for version of Gmail. By the coronavirus pandemic, the cabinet was using Zoom for meetings, despite known insecurities; there did in fact exist a secure videoconferencing system, but as it was classified, ministers weren't allowed to take it home.

Crypto War 1 left a significant legacy, with both technical and political aspects. On the technical front, the mandated use of weak cryptography made DVDs easy to rip, made cars easier to steal, made Bluetooth easy to hack, and made millions of building locks easy to defeat – including the building where I work<sup>4</sup>. The business models of firms selling hotel door locks have been undermined as they can no longer lock in their customers to buying their proprietary card stock. As for policy, authoritarian governments such as Russia's passed harsh crypto control laws; Britain went from a laissez-faire policy under John Major in the mid 1990s to Tony Blair's Regulation of Investigatory Powers (RIP) Act of 2000 which enables the police to demand that I hand over a key or password in my possession, and the Export Control Act of 2002 instructs me to get an export licence if I send any cryptographic software outside Europe that uses keys longer than 56 bits<sup>5</sup>. I'll return to export control later.

### 26.2.8 Crypto War 2 – Going spotty

The 2013 disclosures by Edward Snowden have led to a resumption, after a fashion, of the crypto wars. In fact, the NSA and its partners never stopped, but just took their ‘crypto enabling’ activities underground. They were not only harvesting everyone’s SMSes and email from the backbone, and getting content from major service providers using warrants at much larger scale than we imagined. They were hacking allies, as when GCHQ hacked Belgacom [734] – an amazing story about how one EU member state attacked the critical infrastructure of another, and went on to wiretap the European Commission. Another example was New Zealand’s contribution to the Five Eyes which includes spying on small neighbours such as Samoa, Tonga and French Polynesia [850]. The NSA had lied to Congress, for example about collecting call data records on US citizens. They were bypassing legal controls: GCHQ could get my gmail from Google using Prism, as I’m not a US resident, and we’d always suspected this, but it had always been denied. They were also getting it from major services by covert means – by tapping the communications between Google’s data centres.

---

<sup>4</sup>See section 4.3.1 for car theft, section 5.7.2.2 for attacks on Bluetooth and section [?] for attacks on door locks.

<sup>5</sup>Thankfully, the person who does the exporting is the person who clicks on the link – so if you’re in Iran, you would be a very bad person if you clicked on the link on my website to download the Serpent block cipher. You have been warned!

## 26.2. SURVEILLANCE

---

In 2015, a UK court ruled that for the UK to obtain mass surveillance data on UK residents via the USA had been unlawful, as it contravened the European Convention on Human Rights [304].

All this had a real effect on behaviour. First, the service providers cleaned up their act; Google had been starting to encrypt its internal network but accelerated the program to ensure that the only way to get their users' data was through the front door, by a warrant. Microsoft and Yahoo followed. Second, most messaging systems offered end-to-end encryption to reassure users (and also to save system operators the cost of complying with warrants). Third, the policy conversation started tackling more realistic problems, such as jurisdiction; given that most of the material of interest to the world's police forces is kept on servers belonging to US companies, who can get access to it, and on what terms? While countries like the UK worked at getting faster access to US data, others went for localisation. India had already insisted that all private Blackberry users keep their messages on servers in India; China banned Facebook and Google to ensure its residents used Chinese systems instead; and many countries have passed data-localisation laws to ensure that some kinds of personal data are kept within the jurisdiction. Most countries in Africa, for example, require financial data to be kept locally; I'll discuss the European Union's data-protection regulation and its interaction with US firms later.

Although the agencies no longer ask for access to all keys, the escrow arguments came back in new forms post-Snowden. GCHQ, along with the FBI, started to argue that providers of messaging services such as WhatsApp and FaceTime should be compelled to build in a facility whereby law enforcement can be added as a silent conference-call party (so-called 'ghost users') when they get a warrant. FBI Director James Comey led the charge along with GCHQ Director Robert Hannigan, who accused Facebook in 2014 of helping terrorism [1566] by requiring him to go through the procedures of the UK/USA Mutual Legal Assistance Treaty to get information. Facebook's response was that they were just obeying US and EU privacy laws; the relevant service centre was in Ireland, not the UK, so Hannigan couldn't simply use UK law to force them to help him. He and Comey were supported by UK Prime Minister David Cameron.

My cryptographer colleagues and I reconvened to write an update of our analysis, 'Keys Under Doormats', which explains how many of the problems with 1990s key escrow proposals simply come back in a new form if you mandate government access to data instead of to keys [5]. The effects if anything are likely to be worse, as we are now much more dependent on the Internet than in the 1990s. It would be a bad thing if governments were to force designers to abandon security mechanisms such as forward secrecy, authenticated encryption and strict transport security that have become widespread in the meantime; and because of the many interactions between systems that have been secured in different ways, the risk of mandated vulnerabilities having serious and unanticipated side-effects is now much greater. Building in exceptional access also creates huge targets in the wiretapping systems themselves, and extra complexity that can lead to further security failures. Indeed, the 2010 Chinese hack of Google's wiretapping system suggests that even the best-run companies cannot keep out state actors all the time – and that hack was aimed at the systems

Google built to service wiretaps. The Chinese obviously wanted to know which of their agents in the USA was under suspicion. There are huge problems around jurisdiction. If Facebook carries a WhatsApp message from a user in France to a user in Argentina, do only these two governments get access, or does the NSA demand it too? Since Snowden, everyone knows they will, and nobody believes they could keep such a capability under control. Any demand for such systems raises a lot of questions of both law and engineering, some of which we spelled out in our analysis [5].

The next move came in 2016 when the FBI tried to force Apple to produce an operating system ‘upgrade’ for the iPhone that would unlock it, using as their test case a locked iPhone that had been used in a terrorist attack in San Bernardino. Apple’s Tim Cook had resisted pressure to install back doors before, and saw the case as a serious threat to Apple users’ privacy and to the Apple brand; he fought the FBI in court [1006]. Comey testified that the agency would not be able to get at such vital information without assistance from Apple. The case divided American opinion, with Republicans supporting the FBI (and candidate Trump, as he then was, calling for a boycott of Apple) while most Democrats, and the tech industry, supported Tim Cook. My colleague Sergei Skorobogatov worked out how to defeat the iPhone PIN retry counter [1777], as I discussed in 3.4.8.3. As for the FBI, they bought a commercial iPhone exploit from an Israeli firm, Cellebrite, and dropped the case.

In the chaos following the Brexit referendum, the new UK Prime Minister Theresa May (who as home secretary had been a surveillance hawk) pushed the Investigatory Powers Act through UK parliament. This law grants ministers the power to order any company to do anything physically possible to facilitate signals intelligence collection, and to keep quiet about it forever. In 2018, two senior GCHQ mathematicians, Ian Levy and Crispin Robinson, suggested how government access to messaging services might work [1153]; their idea was that when GCHQ presented Facebook with a warrant, they would add a GCHQ public key quietly to the target’s keyring, so that they’d become a silent conference party to all his calls. My colleague Bruce Schneier responded in detail [1678]: the fact that such an approach would work with some systems (it would work with WhatsApp but not with Signal) is actually a bug that’s being fixed by better transparency mechanisms, and mandating it would prevent the bugfix. In any case, such an access power is excessive; intelligence agencies should not have it because of their history of abusing such access, or simply losing it. In section 2.2.3 I described how the NSA tool EternalBlue was stolen and used by the Russians against Ukraine in the NotPetya worm, causing billions of dollars of collateral damage to US firms in 2016; by 2019 it was being used in ransomware that shut down email and other services in the city of Baltimore, just up the road from the NSA [1529].

In 2019, Mark Zuckerberg announced that Facebook will shift its emphasis from public posts to ephemeral, end-to-end encrypted messaging by unifying WhatsApp with Instagram and Messenger [1439]. Some cynics suggested that this would make it easier to hide fake news and hate speech from both the media and the law, and cut the costs of moderation as well as the PR damage from scandals; others that it was to prevent either the EU or the US government from ordering the breakup of the company [1911, 1931]. In October, the US

Attorney General joined the UK Home Secretary and the Australian Minister for Home Affairs in asking Zuck to think again, highlighting the risk of ‘a single platform that would combine inaccessible messaging services with open profiles, providing unique routes for prospective offenders to identify and groom our children’. Time will tell whether Zuck can do abuse detection using metadata alone; we’ll consider moderation and other forms of censorship below.

### **26.2.9 Export control**

One spillover from the crypto wars was the imposition of more uniform export controls than before, particularly in Europe; here’s a quick summary. International arms control agreements (COCOM and Wassenaar) bind most governments to implement export controls on cryptographic equipment, and the latter is implemented in the European Union by an EU regulation compelling Member States to control and license the export of *dual-use goods* – goods which have both civilian and military uses. Cryptanalytic products fall under the military regime, whereas software that just uses cryptography for protection falls under dual-use.

National policy used to vary more, and during the 1990s European researchers like me could write crypto software and publish it on our web pages, while our US colleagues were prevented from doing that by the US International Trafficking in Arms Regulations (ITAR). US firms complained and in 1997, Vice-President Al Gore persuaded the incoming British Prime Minister Tony Blair to extend export control to intangibles. He initially tried to sell this to the UK parliament, but the relevant committees weren’t keen, so Blair had it pushed through as an EU regulation and his ministers then happily told us “Our hands are tied – we have to do this as it’s EU law”. (Such policy laundering, as it’s called, has been endemic in Europe and is one of the factors that fuelled the movement to get Britain to leave the EU.)

Tens of thousands of academics and small software companies are now breaking the law without knowing it by exporting products (or even by giving away software) containing crypto with keys longer than 56 bits. There are open general export licenses (OGELs) that you can use, but you have to understand the mechanisms and file the paperwork. And it’s not just cryptography. For example, in our hardware tamper-resistance research we use an ion beam workstation, which is like an electron microscope only it fires metal ions at the target rather than electrons, so you can modify a chip by cutting tracks and adding new ones. Like cryptography, this is on the dual-use list. In the old days, we had to get an export licence when we bought one, and another seven years later when we threw it in a skip. Now, we’re in theory supposed to get a licence whenever we share a script we’ve written for the machine with someone who isn’t an EU citizen or resident. The practical outcome is that tens of thousands of scientists happily break the law – which can make them vulnerable to pressure from the agencies. The number has surely shot up now that the pandemic has led to many people working from home, often from overseas, and that the UK has left the EU. How I deal with such issues personally is to be very careful that all such software and scripts are on my website, which enables me to use a public-domain exemption, and rely on the fact that it’s the person who clicks

### **26.3. TERRORISM**

---

on the link who performs the export.

The civil war in Syria exposed the dark side of export control in 2012. People from several digital rights NGOs lobbied the UK government, asking it to use export control law to prevent a UK company selling bulk surveillance equipment to the Assad government. UK NGOs argued that mass surveillance equipment should not just be on the dual-use list but the military list, that the intelligence community includes bulk collection in ‘cryptanalysis’ which is military; and its sale to a government involved in wholesale abuses was against human-rights law. The lady from GCHQ fought this tooth and nail; the sales were going through an arms dealer in Dubai so how could the vendor be sure of the destination; they came from a German subsidiary so it was the Germans’ problem; Wassenaar was a forum for military issues rather than human rights ones; and even that mass surveillance is also used for marketing. The real issue was that GCHQ feared that UK troops would end up in Syria and they were determined that if President Assad was going to have black boxes on his network, they should be British black boxes rather than Ukrainian ones. Eventually the German chancellor Angela Merkel admitted in public that she had decided to allow surveillance equipment to be sold to Syria, and that it was one of the hardest decisions she’d taken. In August 2013, the UK Parliament voted against authorising military action in Syria, and President Obama decided not to go it alone. In due course, the export control issue was referred to European agencies and quietly forgotten.

One unpleasant side-effect of this fight lingers: a system of vetting foreign students at UK universities. GCHQ was opposed to Chinese students studying cryptography, and the security service briefed that an Iraqi woman who’d got a PhD in Britain had gone on to direct part of Saddam Hussein’s alleged research programme into weapons of mass destruction. Briefers raised a scare about people from countries on the terrorist list, such as Sudan, being allowed to study medicine. Tony Minson, a professor of virology and Cambridge colleague, argued that nature can do much nastier things than people can, and if there were no competent public-health people in Khartoum when something like Ebola came down the Nile, we’d regret it. He was of course ignored. We got an ‘Academic Technology Approval Scheme’, and graduate students coming to the UK have to get an ‘ATAS clearance’ to get a visa.

## **26.3 Terrorism**

Talk about terrorism has driven a lot of policy around surveillance and privacy, especially since 9/11. The tide is starting to recede, but it’s still a card that politicians play when they want to scare us, and the media often play along. There has been talk of cyber-terrorism; that basically hasn’t happened, but there are real concerns about encrypted chat services and social media being used to groom and recruit young people to criminal organisations ranging from right-wing hate groups to Islamic State. So what can we say about terrorism?

Political violence is nothing new; anthropologists have found that tribal warfare was endemic among early humans, as indeed it is among chimpanzees [1132]. Terror has long been used to cow subject populations – by the Maya, by the

### **26.3. TERRORISM**

---

Inca, by William the Conqueror. Terrorism of the ‘modern’ sort also goes back centuries. Guy Fawkes tried to blow up Britain’s Houses of Parliament in 1605; his successors, the Irish Republican Army, ran a number of campaigns against the UK. In the latest, from 1969–97, some three thousand people died, and the IRA even blew up a hotel where the Prime Minister, Margaret Thatcher, was staying for a party conference, killing several of her colleagues. During the Cold War, the Russians supported not just the IRA but the Baader Meinhof Gang in Germany and many others; the West armed and supported jihadists fighting the Russians in Afghanistan. Some terrorists, like Baader and Meinhof, ended up in jail, while others – such as the IRA leaders Gerry Adams and Martin McGuinness, the Irgun leader Menachim Begin, the French resistance leader Charles de Gaulle and the African anti-colonial leaders Jomo Kenyatta, Robert Mugabe and Nelson Mandela – ended up in office.

What general lessons can be drawn from this history? Well, there’s good news and bad news.

#### **26.3.1 Causes of political violence**

The biggest piece of good news is that the trend in terrorist violence has been steadily downward [1350]. There were many insurgencies in the 1960s and 70s, some ethnic, some anti-colonial, and some ideological. Many were financed by the Soviet Union or its allies as proxy conflicts in the Cold War, although a handful (notably the Nicaraguan Contras and the resistance to the Soviets in Afghanistan) were financed by the West. The end of the Cold War removed the motive and the money.

The second (and related) point is that the causes of civil conflict are partly economic. An influential study by Paul Collier and Anke Hoeffler for the World Bank looked at wars from 1960–1999 to see whether they were caused largely by grievances (such as high inequality, a lack of political rights, or ethnic and religious divisions), or by greed (some rebellions are more economically viable than others) [459]. The world has plenty of grievances, but the data show that the incidence of rebellion was more determined by whether it could be sustained. (Indeed, Cicero said two thousand years ago that “Endless money forms the sinews of war.”) Thus the IRA campaign got significant support from the Soviet bloc and Libya; the Tamil revolt in Sri Lanka was sustained by funds from ethnic Tamils in the USA and India; and Al-Qaida was financed by rich donors in the Gulf states. So we know one way to tackle an insurgency: cut off their money supply. It’s not entirely that simple, of course; the loss of Soviet support for the ANC (and Angola and Mozambique) reduced the pressure on the last white government of South Africa but gave them the space to do a historic peace deal with Nelson Mandela.

#### **26.3.2 The psychology of political violence**

Less encouraging findings come from scholars of psychology, politics and the media. Psychology gives a lot of insight into the underlying mechanisms. I mentioned the affect heuristic in Section 3.2.5: where people rely on affect, or

### **26.3. TERRORISM**

---

emotion, calculations of probability tend to be disregarded. The prospect of a happy event, such as winning the lottery, will blind most people to the long odds and the low expected return; similarly, a dreadful event, such as a terrorist attack, will make most people disregard the fact that such events are exceedingly rare [1787]. Most of the Americans who died as a result of 9/11 probably did so since then in car crashes, after deciding to drive rather than fly: the shift from flying to driving led to about 1,000 extra fatalities in the following three months, and about 500 a year since then [1677].

There are other effects at the border between psychology and culture. A study of the psychology of terror by Tom Pyszczynski, Sheldon Solomon and Jeff Greenberg looked at how people cope with the fear of death [1564]. They got 22 municipal court judges in Tucson, Arizona, to participate in an experiment in which they were asked to set bail for a drug-addicted prostitute. They were all given a personality questionnaire first, in which half were asked questions such as ‘Please briefly describe the emotions that the thought of your own death arouses in you’ to remind them that we all die one day. The judges for whom mortality had been made salient set an average bail of \$455 while the control group set an average bond of \$50 – a huge effect for such an experiment. Further experiments showed that the mortality-salience group had not just become mean: they were also prepared to give larger rewards to citizens who performed some public act. It turns out that when you remind people of death, it makes them adhere more strongly to their cultural norms and defend their worldview more vigorously. This helps explain why cyber-terrorism just hasn’t happened. Hacking a couple of substations and turning off a town’s electricity can be mighty inconvenient, but it just doesn’t have the same emotional effect as a bleeding child. The media analysis confirms this; coverage is strongly correlated with fatalities, and increases by 46% for each extra dead body [1026].

The 9/11 attacks brought mortality to the forefront of people’s minds, and were also an assault on symbols of national and cultural pride. It was natural that the response included religion (the highest level of church attendance since the 1950s), patriotism (in the form of a high approval rating for the President), and for some people bigotry too. It was natural that, as the memory of the attacks receded, society would repolarise because of divergent core values. Curiously, when they’re reminded that they’re mortal, both conservatives and liberals take a more polarised view of an anti-American essay written by a foreign student – except in experiments where they are first reminded of the Constitution, in which case conservatives defend the student’s right to free speech even more vigorously than liberals do [1564].

So a national leader trying to keep a country together following an attack should constantly remind people what they’re fighting for. This is what the best leaders do, from Churchill’s radio broadcasts to Roosevelt’s fireside chats. In more recent years, some countries have taken a bipartisan approach to terrorism – as when Germany faced the Baader-Meinhof Gang, and Britain the IRA. In others, politicians have given in to the temptation to use fearmongering to get re-elected.

A study by the University of Alabama of over 200,000 articles on the 136 different attacks in the USA between 2005 and 2016 showed that attacks by Muslims get 357% more news coverage than other terrorist attacks [1026]. Islamic

extremists were labelled terrorists 78.4% of the time, whereas far-right extremists were identified as terrorists only 23.6% of the time. Political leadership does matter. Perhaps the best recent response was that of New Zealand Prime Minister Jacinda Ardern to the Christchurch shooting; she not only described it immediately as terrorism but refused to name the shooter. On the other hand, the Pittsburgh synagogue shooting was simply described as a ‘wicked act of mass murder’ by the US President. In each case, the media followed [1335].

What are the dynamics here, and which approaches work best?

### 26.3.3 The role of institutions

There’s a whole academic subject – *public-choice economics* – devoted to explaining why governments act the way they do, and for which one of its founders James Buchanan won the Nobel prize in 1986. As he put it in his prize lecture, “Economists should cease proffering policy advice as if they were employed by a benevolent despot, and they should look to the structure within which political decisions are made.” Much government behaviour is explained by the incentives facing individual public-sector decision makers. It’s natural for officials to build empires as they’re ranked by their span of control rather than by the profits they generate. Similarly, politicians maximise their chances of reelection rather than the abstract welfare of the public. Understanding their decisions requires methodological individualism – analysis of the incentives facing individual presidents, congressmen, generals, police chiefs and newspaper editors, rather than the potential gains or losses of a nation. We know it’s prudent to design institutions so that their leaders’ incentives are aligned with its goals – we give company managers stock options to make them act like shareholders. But this is harder in a polity. What’s the equivalent for presidents and prime ministers? How is the national interest even to be defined?

Public-choice scholars argue that both markets and politics are instruments of exchange. In the former we seek to optimise our utility individually, while in the latter we do the same but using collective actions to achieve goals that we cannot attain in markets because of externalities or other failures. The political process in turn is thus prone to specific types of failure. Intergenerational bargaining is hard: it’s easy for politicians to borrow money to buy votes now, and leave the bill with the next generation, who can’t vote yet. But then why do some countries have much worse public debt than others? The short answer is that institutions matter. Political results depend critically on the rules that constrain political action.

Although public-choice economics emerged in response to problems in public finance in the 1960s, it has some clear lessons. Constitutions matter, as they set the ground rules of the political game. So do administrative structures, as officials are self-interested agents too. In the UK, for example, the initial response to 9/11 was to increase the budget for the security service; but this hundred million dollars or so didn’t offer real pork to the security-industrial complex. So all the pet projects got dusted off, and the political beauty contest was won by a national ID card, a grandiose project that in its original form would have cost £20 billion [1182]. Washington insiders remarked that a similar dynamic was involved in the decision to invade Iraq: although the 2001 invasion

### **26.3. TERRORISM**

---

of Afghanistan had been successful, it had not given much of a role to the Pentagon barons who'd spent careers assembling fleets of tanks, capital ships and fighter-bombers, or much of a payoff to the defense industry either. Indeed, USAF Colonel Karen Kwiatkowski retired at the start of the Iraq war, described how intelligence assessments were politically manipulated, and later ran for Congress [1113]. Similar things were said in the aftermath of World War 1, which was blamed on the 'merchants of death'.

An institution of particular concern must be the media, whether the old-fashioned press or the social media that are taking over some of their functions. 'If it bleeds, it leads', as the saying goes; bad news sells more papers than good. The self-interest of media owners combines with that of politicians who want to get re-elected, officials who want to build empires, and vendors who want to sell security stuff. They pick up on, and amplify, the temporary blip in patriotism and the need for heroes that terrorist attacks naturally instil. Fearmongering gets politicians on the front page and helps them control the agenda. And the recommender algorithms of many social media platforms learn to promote fear and outrage, as they increase the time people spend on the platform and the number of ads they click on.

#### **26.3.4 The democratic response**

Yet people also learn over time. The worldwide reaction to 9/11 was sharp; it was more muted four years later, in July 2005, when four suicide bombers killed 52 people on London's public transport and injured about 700. The initial response of the public was gritty resignation: 'Oh, well, we knew something like this was going to happen – bad luck if you were there, but life goes on.'<sup>6</sup>

And as populations learn, so might political elites. John Mueller has written a history of the attitudes to terrorism of successive US administrations [1350]. Presidents Kennedy, Johnson, Nixon and Ford ignored terrorism. President Carter made a big deal of the Iran hostage crisis, and like 9/11 it gave him a huge boost in the polls at the beginning, but later it ended his presidency. His Secretary of State Cyrus Vance later admitted they should have played down the crisis rather than giving undeserved credibility to the Iranian 'students' who'd kidnapped US diplomats. President Reagan mostly ignored provocations, but succumbed to temptation over the Lebanese hostages and shipped arms to Iran to secure their release. However, once he'd distanced himself from this error, his ratings recovered quickly. In America, people got fed up with President Bush's fear-based policies and elected President Obama whose line was "9/11 is not a way to scare up votes but a challenge that should unite America and the world against the common threats of the 21st century". Much the same happened in the UK, where Margaret Thatcher was re-elected twice after treating terrorists as common criminals. Later, Tony Blair played the fear game, and his departure from office was met with a sigh of relief; his successor Gordon Brown forbade ministers from using the phrase 'war on terror', and David Cameron's government continued that. Mature voters prefer politicians who stand up to terrorists

---

<sup>6</sup>The press went along with this for a couple of days: then there was an explosion of fearmongering. It seems that ministers needed a day or two of meetings to sort out their shopping lists and decide what they would try to shake out of Parliament.

rather than using them as props in their re-election campaigns.

The harshest teacher may be the coronavirus. For years, a pandemic has been at the top of Britain's risk register, yet far less was spent preparing for one than on anti-terrorist measures, many of which were ostentatious rather than effective. This misallocation of resources looks set to cost far more of us our lives than any terrorist could have dreamed of. The US and UK governments justified torture in the 2000s by talking of an al-Qaida cell stealing a nuclear bomb and detonating it in New York or London. Yet a 10 kT atomic demolition munition set off in a major city might cost 50–100,000 lives, compared with the 50–100 million who died in the 1918–19 pandemic. The rhetoric of terror puffed up the security agencies at the expense of public health, predisposing governments in America, Europe, India and Africa to disregard the lesson of SARS in 2003 – unlike the governments of China, Singapore, Taiwan and South Korea.

## **26.4 Censorship**

I wrote in the first edition that “the 1990s debate on crypto policy is likely to be a test run for an even bigger battle, which will be over anonymity, censorship and copyright.” By the second edition, I noted that “copyright law has largely stabilised”, and it was during 2008 that power over content distribution shifted from the music majors and Hollywood to tech firms like Apple and Amazon. I also noted that “censorship has become a much bigger issue over the past few years”. Now, a decade later, censorship is front and centre. It has two faces: state censorship, and content filtering by service companies.

Rulers have long censored books, although the invention of the printing press made their job a whole lot harder. When John Wycliffe translated the Bible into English in 1380–1, the Lollard movement he started was suppressed along with the Peasants’ Revolt. But when William Tyndale had another go in 1524–5, printing let him spread the word so widely that the princes and bishops could not suppress it. They had him burned at the stake, but by then over 50,000 copies of the New Testament had been printed, and the Reformation was under way. After that upset, printers were closely licensed and controlled; things only eased up in the eighteenth century.

Censorship nowadays is done for a variety of motives. Most countries block images of child sex abuse; during the 1990s, as the dotcom boom got underway, governments started looking for some handle on the Internet, and a view arose that images of child sex abuse were about the one thing that all states could agree should be banned. In due course the 2004 Cybercrime Convention obliged signatory states to ban sexual images of under-18s. Most governments go further and block some kinds of hate speech. Britain bans websites that ‘radicalise’ young people by glorifying terrorism. Finally, censorship is sometimes imposed by the courts.

The invention of the Internet has made the censors’ job easier in some ways and harder in others. It’s easier for the authorities to order changes in material that not many people care about: for example, courts that find a newspaper guilty of libel order the offending material to be removed. Changing the histor-

ical record wasn't possible when it consisted of physical copies in libraries, and the centralisation of human knowledge in the servers of a small number of firms – from Amazon's e-book system to the servers of the major news organisations – takes us, in some sense, back to the 15th century. It's also easier for the authorities to observe the transmission of disapproved material, as they can monitor electronic communications more easily than physical packages. On the other hand, nowadays everyone can be a publisher; much of the really unpleasant material online comes from millions of individuals posting sort-of anonymously to social media, to the comment pages of newspapers, and to individuals whom they wish to harass and intimidate. Censors have learned to harness this. While a decade ago China had tens of thousands of people who took down dissident speech, now they have millions of citizen volunteers who drown it out. Once, speech was scarce, and the censors tried to silence the speaker; now it's the listener's attention that's scarce, and so different tactics work.

To tease out the issues, let's look at some contexts.

#### 26.4.1 Censorship by authoritarian regimes

When I wrote the second edition of this book, I was cautiously optimistic that the government of China would fail in its attempts to censor all online content. However the authorities there have become steadily more effective at suppressing any forms of organisation and human solidarity outside of party control.

By 2006, observers noted that online discussion of local news events had led to the emergence of 'public opinion' that for the first time was not in thrall to media managers [1470]. China had 137 million Internet users then, including a quarter of the population in the big cities, and 'the Great Firewall of China' was already a complex system of controls giving defence in depth against a range of material, from pornography to religious material to political dissent [1469]. The defences work at three levels.

First, there are the perimeter defences. China's border routers filter on IP addresses to block access to known 'bad' sites like the Voice of America and the BBC; they also use DNS cache poisoning. Deep packet inspection at the TCP level is used to identify emails and web pages containing forbidden words such as 'Falun Gong'; such connections are torn down. Ten years ago, much of the work was done at this level. Nowadays, since most traffic is encrypted, that's not so easy. In 2020, the firewall started dropping TLS 1.3 traffic using *Encrypted Server Name Indication* (ESNI) as this stops the censor telling which subdomain the traffic's going to; this amounted to over 30% of traffic by the beginning of July [433].

Second, there are application-level defences, which now do much of the work. Nowadays some services are blocked and some aren't, depending on whether the service provider is prepared to help the regime with both surveillance and censorship. Google and Facebook are largely blocked; China has promoted Tencent, Alibaba and Baidu instead. Now that the borders that matter most are those of firms rather than of nations, the Chinese government has aligned its industrial policy with its politics. This is the big change; we never believed ten years ago that China would build an entire ecosystem of Chinese-owned

## *26.4. CENSORSHIP*

---

online service providers to keep western influence at bay. Language provides one barrier, but there are strong technical barriers too: the perimeter defences now focus on blocking Tor and VPNs that could be used by Chinese residents to use non-approved services.

Third, there are social defences. There were already 30,000 online police a decade ago; now many more citizens have been engaged in the process, and rather than trying to block all dissident speech the strategy is to swamp it. Loyal citizens are expected to post lots of pro-regime comments and to flame anybody who criticises authority, whether local or national. A social credit system gives people positive points for such pro-social behaviour, while they can lose points for anything considered antisocial. Online monitoring is being integrated with the monitoring of physical space, such as by CCTV cameras with face recognition and emotion recognition – which is particularly aggressive in areas with rebellious minority populations, such as the Tibetans and Uighurs. Since 2014, a system in Sinkiang for ‘re-education’ has pioneered a fusion of techniques from the western ‘war on terror’ and Maoist social control, leading to the internment of hundreds of thousands of Muslims on the basis of a scoring system whose inputs include whether a suspect prays regularly or has a VPN on their phone. The U.S. Congress has denounced this regime for ‘crimes against humanity’: dozens of the contractor companies have been placed on the sanctions list [359].

So China appears to be winning the censorship battle, using populist but authoritarian techniques. Russia’s Internet is fairly open, and although the government had an ally take over the main social network, and has organised armies of trolls to shout down its opponents, the opposition politician Alexei Navalny has his own YouTube channel with millions of viewers, and attempts to censor Telegram have been met with street protests. Putin has fought back with a ‘digital sovereignty’ law enabling him to order ISPs to install surveillance and censorship equipment.

The Arab Spring has also been significant. This series of uprisings started in Tunisia in December 2010 after a street vendor, Mohamed Bouazizi, set himself on fire after an official confiscated his wares and humiliated him. Protests were organised using Facebook and other social media, leading to the downfall of the government, and spreading to neighbouring countries too. The government of Egypt also fell, along with those of Libya and the Yemen; in Egypt’s case a Google employee, Wael Ghonim, turned Internet activist after the police beat a man to death in Alexandria on suspicion that he had video evidence of their involvement in a drug deal. The government of Syria almost fell, but fought back in a civil war that killed hundreds of thousands and displaced millions. A number of other Arab countries, such as Bahrain, suffered significant unrest and cracked down. As I write in 2020, only Tunisia has managed the transition to democracy. In Egypt, one military dictator has been replaced by another; Libya is in chaos, and Yemen, like Syria, is racked by war. The lesson drawn by the world’s autocrats is that, to stay in power, they’d better study the methods used by China. Arab countries do censor the Internet (as do most of the less-developed countries) but their infrastructure is still fairly easily defeated using VPNs or Tor. They also buy in kit for both bulk surveillance and targeted work; for a description of how the UAE hired US mercenaries to set up an equivalent of the NSA, see Bing and Schectman [247].

To what extent was the Arab Spring a function of technology, and to what extent was this just marketing hype put out in 2011 by companies like Facebook and Google while things seemed to be going well? It's unclear. Some of the populations that rose up made little use of the Internet, particularly those of Libya and the Yemen; on the other hand, a revolt in Burma in 2007 was catalysed by the Internet, even though only 1% of the population had access [1471]. In the Arab world, the Qatari TV station Al-Jazeera may have done more work than the Internet, by showing news videos of uprisings elsewhere in the region.

#### 26.4.2 Filtering, hate speech and radicalisation

Democracies' laws on hate speech vary widely. At one end, the USA has constitutional protection for free speech; so do France and Germany. But interpretations differ. France and Germany both prohibit the sale of Nazi memorabilia, and hate speech ('Volksverhetzung') has been a crime in Germany for decades. In January 2018 the authorities started enforcing it against online service providers, with the threat of a fine of €50m if any service provider with more than 2m customers doesn't take down any such material within 24 hours. Whatever the service companies say about the cost of taking down bad stuff, the German example shows they can do it when they have to. Many countries now ban terrorist material and extreme violence, the definition of which is never straightforward. It might seem a good thing to ban not just beheading videos but all videos of murder, such as drug gangs shooting a customer who didn't pay his debts. But it gets complex quickly. Platforms that enforce such a policy end up deleting evidence, both of local killings and of human-rights violations overseas.

Already much of the material you put online gets filtered automatically to look for material that's forbidden by local laws, or by a platform's terms of service. Facebook's former CISO Alex Stamos described the tension between privacy and censorship as a spectrum: people expect end-to-end encrypted chat such as WhatsApp to be private rather than censored, and broadcast media to be censored rather than private, with the difficult stuff in the middle, like Facebook groups. By now, most social media are censored. The platforms vary widely; Facebook is perhaps the tightest, and bans even nudity<sup>7</sup>; though it is much more forgiving of hate speech from President Trump than from others, and in return appears to receive much less attention on the antitrust front [1790]. Authoritarian countries are becoming more aggressive about forcing service firms to block content they deem to be illegal; for example, Facebook's service was slowed to a crawl in Vietnam in early 2020 until the company agreed to suppress dissent [1506].

Behind the AI systems that try to spot forbidden content are thousands of content moderators. Filtering is expensive, and the costs are not just financial, but human; we've seen an increasing number of news articles about the psychological toll on staff who have to spend all day looking at videos of gang murders and terrorist beheadings, animal cruelty, child abuse, and other un-

---

<sup>7</sup>Facebook bans photos of female nipples but not male ones, so dozens of naked women demonstrated in 2019 in New York holding pictures of men's nipples over their own; men and women demonstrated with pictures of female nipples [616].

## 26.4. CENSORSHIP

---

pleasantness [1438]. Many moderators are in less developed countries; just as we dump a lot of unpleasant refuse there, we also dump a lot of the Internet's nastiest trash [414]. It's also problematic to outsource censorship to large service monopolies. They act in a quasi-judicial manner, regulating the speech of billions of people but without the transparency and due process we expect of government decisions. The world sees them allowing abuse by the rich and powerful while ignoring the weak. Perhaps it was inevitable that firms would snuggle up to power and then try to direct political speech; this has become a factor in the backlash against the whole tech sector.

One focus of debate is section 230 of the US Communications Decency Act of 1996 (CDA) which states that 'No provider or user of an interactive computer service shall be treated as the publisher or speaker of any information provided by another information content provider' so platforms cannot be held liable for bad stuff provided by users; it also left platforms free to remove anything 'obscene, lewd, lascivious, filthy, excessively violent, harassing, or otherwise objectionable.' When it passed the CDA, Congress was concerned that firms that moderated content could be treated as publishers and held liable for all of it (including copyright infringement and libel) while firms that didn't would be treated as distributors and escape liability. How could we get a civil internet without killing innovation? Section 230 made firms like YouTube and Facebook possible, but protected sites whose business model is based on revenge porn, defamation, or getting a cut of illegal gun sales [1419]. It also enabled service firms to acquire some of the powers of states. Back then, the Internet had 10-20m users, mostly geeks; now most human activity is online, and it's not sustainable for a handful of American firms to act as censor, prosecutor and judge for 200-odd countries. As a result the CDA, and similar laws elsewhere, are starting to be trimmed: in the USA in 2018 with laws on sex trafficking and in Europe with a 2019 law on copyright [1598]. The tensions can only get worse.

When making laws to restrict speech, it's a good idea to stop and look at the historical context. Tim Wu's 'The attention merchants' [2050] is a history of propaganda since the 1830s when the first mass-market newspapers appeared, stuffed with grisly crime reports and adverts for patent medicines; this gave politicians their first industrial mass-market channel. Radio followed, and was used skilfully by Hitler. TV was next, and its nature was shaped by advertising; people invented quiz shows, soaps and much else to grab eyeballs. A second useful perspective is Yochai Benkler's 'Network propaganda' which analyses the 2016 US election campaign. He traces the history of political polarisation and argues that the root cause of the outcome wasn't technology or Russian interference so much as the asymmetric media systems of right and left that have developed over the past 20 years; the left and centre-right are fact-based while the right is a propaganda feedback loop [227]. A third perspective is the critique of recommender systems by former Googler Tristan Harris: the platforms' algorithms learn that to maximise the time people spend on site, they should be fed articles that stoke fear, anxiety and outrage.

The reactions of governments to fake news are mostly ineffective. The most capable may be Finland, which has been a target of Russian propaganda since Tsarist times. Its government has been promoting critical thinking and media literacy in schools and elsewhere since 2014, making it every citizen's job

## 26.4. CENSORSHIP

---

to spot and counter information that's designed to sow division. In Britain we have laws designed to please tabloid newspapers rather than to push back against them. Schoolteachers and university professors are supposed to report students who seem at risk of being radicalised, and have procedures to figure out whether seminars or other talks could radicalise them; there are also laws against online material that might lead them astray. If such an approach were applied consistently it might lead to banning much of the literature produced or funded by religious institutions from Saudi Arabia [1263], but action against our largest arms export customer isn't going to happen anytime soon. White supremacists are at least as much of a threat, having murdered a member of the UK parliament during the Brexit campaign; but our government is much less keen on cracking down on them, and the people who broke the law by spending too much money on that campaign (including Russian money) did not end up in jail, but at the heart of government. In general, Internet censorship lets the government claim it's doing something, but doesn't really work well, and undermines whatever our diplomats might say about freedom of speech to the world's despots. I'd prefer to enforce existing laws on incitement to murder (and campaign finance, leave other political material in the open, let the police monitor the traffic to the worst of the sites, and train them to use the existing laws better [642]. In the longer term, the key is education, as Finland has shown.

As for targeting Muslim students, this runs directly against the criminological evidence. The few UK students who've signed up to extremist organisations have been those who experienced lack of respect socially, perhaps being rejected by their peers, were searching for identity but couldn't find it in the religion of their parents – then fell in with small groups of other disaffected youngsters. They came under the influence of radical preachers, who offered ideals, community, kinship, caring and brotherhood. The radicalisation of white boys into white supremacist groups is not hugely different. Research by Max Abrahms also shows that terrorists mostly joined their movement in a search for social solidarity; that's why they recruit from lonely young men rather than from among political activists. Their groups become institutions to which members cleave, rather than agents of change; that's why they can respond to sensible peace offers with increased violence, and indulge in fratricidal conflict with similar groups [6]. In fact, as Lydia Wilson pointed out after interviewing large numbers of young people who'd gone to Syria to join Daesh and ended up in Kurdish jails, the process whereby young men (and occasionally women) find their identity by joining terror groups or crime gangs is no different from finding identity by joining religions, sports clubs or dance bands [2022]. Zoë Quinn's more recent experience of angry online mobs during the Gamergate drama, which we discussed in section 2.5.1, draws much the same conclusion [1567]. The people who join extreme organisations in search of social solidarity need to think of themselves as the good guys; you need to undermine that, and you can't do it by excluding them.

For all these reasons, it is unwise to model terrorist groups as rational economic actors, and just as unwise to try to prevent radicalisation on similar assumptions. The best approach is to have an environment that doesn't exclude people – one in which students get to know others from different backgrounds on the staircase in their residence, in small teaching groups, and in project groups – and with hundreds of sports and student societies to choose from, so everyone

can find a gang to belong to. That's how great universities have always worked anyway.

## 26.5 Forensics and Rules of Evidence

Our last main policing topic is how information can be recovered from computers, mobile phones and other electronic devices for use in evidence. This has been getting more problematic over the past twenty years because of first, the sheer volumes of data; and second, the fact that while much of it is seized from platforms such as mobile phones and laptops, more and more of it is held on cloud services that require paperwork and often quite substantial delays. The rising costs and operational difficulties lead to more selective law enforcement, with whole categories of online harms where states rarely intervene. As a result, many bad people, from cybercriminals to creeps, bullies and extremists, operate online with near-total impunity.

### 26.5.1 Forensics

Computer forensics has been a growing problem for the police since at least the 1980s; by the early 2000s both the facilities and the staff training were hopelessly behind. The move of everything online during the 2010s has made matters still worse. When the police raid even a small-time drug dealer nowadays, they can get half-a-dozen mobile phones, several laptops, and gadgets such as a navigator or a Fitbit that hold his location history. The suspect may also have dozens of accounts for webmail, social-networking sites and other services. We have all sorts of clever ways of extracting information from the data – for example, you can identify which camera took a picture from the pattern noise of the CCD array [1192], and even use this to figure out which parts of a photo might have been tampered with.

The use of digital material in evidence depends, however, on both law and economics. Material has to be lawfully collected, whether with a search warrant or equivalent powers; and the forensic officer has to maintain a *chain of custody*, which means being able to satisfy a court that evidence wasn't tampered with afterwards. That means using trustworthy tools to make evidential copies of data; to document everything that's done; and to have means of dealing appropriately with any private material that's found (such as privileged lawyer-client emails, or the trade secrets of a suspect's employer). The traditional approach to computer forensics is described in standard textbooks such as Sammes and Jenkinson [1644].

Since the world moved to smartphones and cloud services, the centre of gravity has shifted to a handful of companies that sell mobile forensics tools to police and intelligence agencies. They supply kiosks to police forces that enable unskilled officers to download mobile-phone contents, and to use the tokens on them to download data from suspects' accounts in the cloud. Some police forces are working hard to get the legal issues sorted out (such as Police Scotland, who don't use 'cloud forensics' without a warrant) but many just grab and keep all the data.

At the more sophisticated end of the trade, there's an arms race between forensics and countermeasures. Police forces used to always turn PCs off, so that hard disks could be copied for prosecution and defence lawyers. Phishing gangs exploited this by making their phishing software memory-resident, so that the evidence would self-destruct. And since laptops started to ship with decent encryption, the risk is multiplied. By 2013, when the FBI arrested Ross Ulbricht – the creator of the Silk Road underground drugs market – one agent's mission was to put his hand in the laptop to stop Ulbricht closing it, and he already had the right kind of power cord to plug it in [482].

In the old days, people – and small businesses – who got caught up in a police investigation and had their computers seized could wait years to get them back, even if they were just a bystander, or if they were charged but eventually acquitted. Nowadays, people have seizure-proof offsite backup thanks to cloud services. These services also make life harder for the police where suspects' material sits on servers overseas. The fight between Facebook and GCHQ I referred to in Section 26.2.8 arose when two terrorists murdered a British soldier, Lee Rigby, near Woolwich barracks in March 2013 by running him over with a car and then stabbing him. While they were at the crime scene, facing off against the police, Facebook fed the police and security services data instantly, but once the two had been shot and were in custody in hospital, requests had to go through the UK/US mutual legal assistance treaty. This involves the police filing forms at the US Embassy in London that are then considered at length in the Department of Justice in Washington. The forms are often sent back as UK police staff don't understand US law and complete them incorrectly. Even where everything goes right, it can take six weeks for the FBI to serve the paperwork on Facebook in Menlo Park, California, and collect the data. So we found we'd gone from a world in which, after a raid, the police would have your data and you wouldn't, to one in which you still have your data but the police don't – unless you cooperate, or unless you're a serious enough bad guy to be worth the time and attention of diplomats.

Since about 2017, there's been a third option: cloud forensics. What this means in practice is that your phone is hacked by the police's forensic kiosk and gives up access tokens to your email, your photos, your Facebook and your other cloud services. Some UK police forces think this is wonderful; they treat the downloaded data as 'data at rest' as if it had been found on the phone itself and keep it forever. Others consider that it can only be obtained by consent or with a further warrant. The incentives to grab cloud data are strong, but the mechanisms involved (phone hacking followed by impersonation of the user) are likely to strike most citizens as unfair. And ever more devices are now acquiring an attached cloud service and an app. Will the police investigate traffic offences in future by seizing the driver's phone and using it to download the car's logs from the manufacturer's server? This is a current policy topic in 2020: for example, the UK privacy regulator called for a statutory code of practice to be developed [958]. As it happens, courts already have some rules about what evidence can be used.

### 26.5.2 Admissibility of evidence

When courts were first confronted with computer evidence in the 1960s there were many concerns about its reliability. There was not just the engineering issue of whether the data were accurate, but the legal issue of whether computer-generated data were inadmissible as hearsay. Different legislatures tackled this differently. In the US, most of the law is found in the Federal Rules of Evidence where 803(6) allows computer data to be introduced as records ‘made at or near the time by, or from information transmitted by, a person with knowledge, if kept in the course of a regularly conducted business activity... unless the source of information or the method or circumstances of preparation indicate lack of trustworthiness.’ The UK is similar, and the rules of electronic evidence in the common-law countries (including Canada, Australia, South Africa and Singapore) are analysed by Stephen Mason [1236].

The definition of ‘writing’ and ‘signature’ is of interest, and varies by jurisdiction. In Britain, courts took the view that an email is writing just as a letter is: the essence of a signature is the signer’s intent [2042, 2043]. The US approach was similarly pragmatic. In 2000, Congress enacted the Electronic Signatures in Global and National Commerce (‘ESIGN’) Act, which gives legal force to any ‘sound, symbol, or process’ by which a consumer assents to something. So pressing a telephone keypad (‘press 0 to agree or 9 to terminate this transaction’), clicking a hyper-link to enter a web site, or clicking ‘continue’ on a software installer, the consumer consents to be bound to a contract [669]. This makes click-wrap licenses perfectly valid in America. Nonetheless, DocuSign has built a business offering digital signatures as a service for firms who want something a bit more showy.

In Europe the Electronic Signature Directive, which came into force in 2000, gave special force to an *advanced electronic signature*, which basically means a digital signature generated with a smartcard or hardware security module. Europe’s smartcard industry thought this would earn them lots of money, but it languished for years. In many countries, the risk that a paper check will be forged is borne by the relying party: if someone forges a check on my account, then it’s not my signature, and I have not given the bank my mandate to debit my account; so if they negligently rely on a forged signature and do so, that’s their lookout. However, if I ever accept an advanced electronic signature device, then I become liable to anyone in the world for any signature that appears to have been made by this device, regardless of whether or not I actually made it! This, coupled with the facts that smartcards don’t have a trusted user interface and that the PCs which most people would use as an interface are easily subverted, made such electronic signatures unattractive. Following further lobbying, Europe updated the law with the eIDAS Regulation (910/2014) which tries to improve the incentives for adoption, by requiring all organisations delivering public services to accept electronic signatures since 2018. A number of EU countries now insist that you use such a signature to file your taxes, rather than permitting it. There’s a hierarchy whereby a signature can be ‘advanced’ or ‘qualified’ depending on the certification of the technology used, and a qualified electronic signature must be accepted for any purpose for which a handwritten signature was previously required. Dozens of signature creation products were duly certified and brought to market. The assurance mechanisms used

to certify such products are defective in many ways, as I will discuss later in section 28.2.7.2. The European Commission duly made a reference implementation available to help governments get started with verifying all the signatures; in 2019 bugs were discovered in it that would let any citizen impersonate any other [429].

### **26.5.3 What goes wrong**

Many things can go wrong with police investigations, and the computerised kind are no different. An old pitfall is relying on evidence extracted from the systems of one party to a dispute, without applying enough scepticism about its dependability. Recall the Munden case described in Section 12.4.3. A man was falsely accused and wrongly convicted of attempted fraud after he complained of unauthorized withdrawals from his bank account. On appeal, his defence team got an order from the court that the bank open its systems to the defence expert as it had done to the prosecution. The bank refused, the bank statements were ruled inadmissible and the case collapsed. The same has happened multiple times since then, including two terror cases involving curfew tags which I discussed in section 14.4.

The worst failure of computer evidence of which I'm aware was Operation Ore. After the US Postal Service raided a porn site in Texas, they discovered hundreds of thousands of credit card numbers that they thought had been used to buy child sex abuse images, and some eight thousand of these were from UK cardholders. Some 3,000 homes got raided in the early 2000s, until the police finally realised that most of the cardholders were probably victims of card fraud. The vice squad used unskilled staff in their initial analysis of the seized material, and were slow to learn – because they were fixated on getting porn convictions, because they didn't have the forensic capacity to process all the seized computers quickly, because they didn't understand card fraud (they preferred to leave that to the banks) and because of politics (Prime Minister Tony Blair himself had ordered the raids). So several thousand men had their lives disrupted for months or even years, and the sad story of police bungling and cover-up is told by Duncan Campbell in [375, 376]. For some, the revelation that the police had screwed up came too late; over thirty men, faced with prosecution, killed themselves. At least one of them, Commodore David White, commander of British forces in Gibraltar, appears to have been innocent [886]. The gangsters in Indonesia and Brazil who organised and photographed the child abuse do not seem to have been seriously pursued. America handled this case much better. Some 300,000 US credit card numbers were found on the same servers, but US police forces used the data for intelligence rather than evidence, identifying suspects of concern – such as people working with children – and quietly investigating them. Over a hundred convictions for actual child abuse followed.

Sometimes systems are deliberately designed to not provide evidence; an example is the policy adopted by Microsoft after embarrassing emails came out during their antitrust battles with the US government in the 1990s. The firm reacted with a policy that all emails are discarded after a fixed period of time unless someone takes positive action to save them, and many other firms followed

suit. Another example is the move by service firms in the mid-2010s to adopt end-to-end encryption, so they don't have access to customer message traffic and don't have to employ hundreds of lawyers to deal with requests for it.

The biggest problem with computer forensics, though, has always been sheer lack of money. Despite all the cool tricks that intelligence agencies can use to extract information from computer systems, a county drugs squad often won't have the budget to do even basic computer forensics except for occasional big cases. They can't even afford to send every wrap of white powder off to the lab to see if it's illegal or not. In normal cases, they were only able to use digital material that was easily available, such as copies of messages on the phones of cooperative witnesses, until mobile-phone forensic kiosks came along around 2016–8 and made masses of data available from seized handsets at low marginal costs. Hence the huge pressure to use the kiosks, even before robust legal procedures could be developed. And, of course, the use of forensic tools by regular police officers with no specialist training raises the risk of future miscarriages of justice. Judicial education is also an issue; few judges understand probability theory, and indeed the UK Court of Appeal has refused to accept analysis of evidence based on Bayes' theorem. Quite apart from the injustice of a court system that denies mathematics, there's the practical issue that defendants faced with computer evidence that's the result of bugs, or simply misrepresented, may have no practical way to prove their innocence.

## **26.6 Privacy and Data Protection**

Privacy and data protection are one subject on which the USA and Europe have taken separate paths. A concentrated interest (such as business wanting to use our personal information to exploit us) usually prevails over a diffuse interest (such as the desire of individuals to keep control of our personal information), and the usual remedy is law. The remedy is imperfect because the concentrated interest lobbies the lawmakers and will attempt to capture any regulator they set up. And Europe, for historical reasons regulates more than America does. The resulting gulf was highlighted powerfully in May 2014 when, in the USA, the Presidential Council of Advisers on Science and Technology (PCAST) published “Big Data: A Technological Perspective” [1546]. This report, whose authors included Google’s Eric Schmidt and Microsoft’s Craig Mundie, painted a picture of a world full of smart objects connected to cloud servers, with an ecology in which sensors reported to cloud analytics which in turn provided information to users, such as advertisers. PCAST warned that the spread of voice and gesture interfaces meant that pretty soon, every inhabited space on the planet would have microphones and cameras in it, whose output would be processed centrally for energy efficiency. They argued that privacy controls could not be imposed on the sensors, as they’ll be too numerous; that they should not be imposed on the central service aggregators; and that the controls would therefore have to fall on how the information was used.

Less than two weeks later, the European Court of Justice disagreed. A Spanish lawyer, Mario Costeja González, had complained that searches for his name brought up two ancient press reports of an auction sale of his repossessed

house. He asked the Spanish data protection authorities to order Google to stop serving these results as they were out of date and no longer relevant. Google argued that it was just reporting the contents of a newspaper. The case went to the ECJ, which found in Gonzàlez' favour, creating what the media colourfully if inaccurately called a ‘right to be forgotten’, later codified into Europe’s General Data Protection Regulation from 2018. Google and other online service providers had to set up mechanisms whereby people could complain about search results that are ‘inadequate, irrelevant or no longer relevant, or excessive in relation to the purposes for which they were processed’ and have them removed. The mechanisms are contentious: Gonzàlez’ results are removed from Google searches in Spain, but European regulators want them removed globally. Google’s supporters claim that this would interfere with its right to free speech in the USA.

How did this rift come about?

### 26.6.1 European data protection

Fear of technology undermining privacy isn’t a recent development. As early as 1890, Justices Warren and Brandeis warned of the threat to privacy posed by ‘recent inventions and business methods’ – specifically photography and investigative journalism [1988]. After banks, tax collectors and welfare agencies started using computers in the early 1960s, people started to worry about the privacy implications if all our transactions could be collated and analyzed. In Europe, business argued that only government could afford enough computers to be a serious privacy threat. This became a human-rights issue, given living memory of the Gestapo in most European countries and of communist secret police forces in the East<sup>8</sup>.

A patchwork of data protection laws started to appear starting with the German state of Hesse in 1969. Because of the rate at which technology changes, the successful laws have been technology neutral. Their common theme was a regulator (whether at national or state level) to whom users of personal data had to report and who could instruct them to cease and desist from inappropriate processing. The practical effect was usually that the general law became expressed through a plethora of domain-specific codes of practice.

Over time, processing by multinational businesses became an issue too, and people realised that purely local or national initiatives were likely to be ineffective against them. Following a voluntary code of conduct promulgated by the OECD in 1980 [1476], data protection was entrenched by a Council of Europe convention in January 1981, which entered into force in October 1985 [475]. Although strictly speaking this convention was voluntary, many states signed up to it for fear of losing access to data-processing markets. It required certain minimum safeguards for *personal information*, which generally means any data kept on an identifiable human being, or *data subject*, such as bank account

---

<sup>8</sup>In Germany, privacy is now entrenched in the constitution, and trumps even the ‘war on terror’. The highest court found unconstitutional a 2001 police action to create a file on over 30,000 male students or former students from Muslim-majority countries – even though no-one was arrested as a result. It ruled that such exercises could be performed only in response to concrete threats, not as a precautionary measure [344].

details and credit card purchasing patterns. Data subjects have the right to inspect personal data held on them, have records changed if inaccurate, understand how they're processed, and in many cases prevent them being passed on to other organizations without their consent. Almost all commercial data are covered. There are exemptions for national security, but they are not as complete as the spooks would like: there was a big row when it turned out that data from SWIFT, which processes interbank payment instructions, were being copied to the Department of Homeland Security without the knowledge of data subjects; SWIFT eventually agreed to stop processing European data in the USA [1485, 1486].

The quality of implementation varied widely. In the UK, for example, Margaret Thatcher unashamedly did as little as possible to comply; a data protection body was established but starved of funds and technical expertise, and many exemptions were provided for both government and industry<sup>9</sup>. In Germany, which had written a right to privacy into its post-war constitution, the data protection bodies became proper law-enforcement agencies. Many other countries, such as Australia, Canada, New Zealand and Switzerland passed comparable privacy laws in the 1980s and early 1990s: some, like Switzerland, went for the German model while others, like Iceland and Ireland, followed the British one.

By the early 1990s the difference between national laws was creating barriers to trade. Some businesses avoided controls altogether by moving their data processing to the USA. So data protection was finally elevated to the status of European Union law in 1995 with a Data Protection Directive [647]. This set higher minimum standards than before, with particularly stringent controls on highly sensitive data such as health, religion, race and political affiliation. It also set out to prevent personal information being shipped to 'data havens' such as the USA in the absence of comparable controls enforced by contract or treaty.

The British implementation was again minimal, falling far short of European requirements [597]. For example, data controllers could pretend that lightly-anonymised information was no longer personal information, just so long as they themselves did not possess the auxiliary data needed to re-identify it. The Information Commissioner's Office was overwhelmed, and severely conflicted as a result of being simultaneously the public sector's adviser on privacy and the privacy enforcer; the enforcement arm was reluctant to take action against systems blessed by their colleagues in the advisory arm. Ireland's enforcement was even weaker – its industrial strategy for the past 50 years has been to attract US firms' European headquarters. So in addition to having low corporate taxes, the Dublin government located its data protection office in Portarlington, a town of less than 10,000 people, gave it only 30 staff, and did not allow it to publicise the results of investigations.

This so annoyed countries with tighter privacy laws such as France and Germany that they pushed for the General Data Protection Regulation (GDPR), which passed in 2016 and came into force in May 2018. This was the most heavily lobbied piece of European legislation ever, with over 3,000 amendments discussed in committee in the European Parliament [82]; it was helped over the line

---

<sup>9</sup>In one case where you'd expect there to be an exemption, there wasn't; journalists who kept notes on their laptops or PCs which identified people were formally liable to give copies of this information to the data subjects on demand.

by the Snowden disclosures, although it had been cooking for some time before that<sup>10</sup>. GDPR took direct effect in all EU member states, removing the wriggle room for Britain or Ireland to introduce loopholes; but lobbyists got quite a few of those in the Regulation already (particularly for ‘research’, whether of the scientific or marketing kind). The main effect on normal businesses is to force them to document all their uses of personal information and write down, in advance, what the legal basis is for each of them; it’s not enough to try and figure things out once challenged. For information-intensive businesses, the implications could be more significant, and there have been fascinating disclosures of how Facebook executives lobbied to amend the regulation – effectively using the Irish prime minister, Enda Kenny, as their advocate in Brussels [1418].

Despite the many carve-outs inserted by the lobbyists, GDPR is still providing regulators with tools to push back. France fined Google €50m for failing to tell users enough about its data consent policies or give them enough control over how their information is used [1534]. The fact that consent can no longer be coerced or presumed may become a big deal, and there are many further cases in the pipeline.

### 26.6.2 Privacy regulation in the USA

In the USA, business has mostly managed to persuade government to leave privacy largely to ‘self-regulation’. Although there’s a patchwork of state and federal laws, they are application-specific and fragmented. In general, privacy in federal government records and in communications is regulated, while business data are largely uncontrolled. The few islands of regulation include the Fair Credit Reporting Act of 1970, which governs disclosure of credit information and is broadly similar to European rules; the Video Privacy Protection Act or “Bork Bill”, enacted after a Washington newspaper published Judge Robert Bork’s video rental history following his nomination to the US Supreme Court; the Drivers’ Privacy Protection Act, enacted to protect privacy of DMV records after the actress Rebecca Schaeffer was murdered by an obsessed fan who hired a private eye to find her address; and the Health Insurance Portability and Accountability Act which protects medical records and which I discussed in Chapter 9. Most states also have a breach disclosure law, which requires firms suffering any security failure that compromises residents’ personal information to inform them about it. Several torts also provide a basis for civil action in a surprising number of circumstances; for a survey, see Daniel Solove [1801].

The first case that started to put privacy on CEOs’ radar came in 2006, when Choicepoint paid \$10m to settle a lawsuit brought by the FTC after it failed to vet subscribers properly and let crooks buy the personal information of over 160,000 Americans, leading to at least 800 cases of ‘identity theft’ [671]. In 2007, it came out that the store chain TJ Maxx had had 45.7 million customers’ credit card details stolen [1159]; Albert Gonzales got 20 years in prison for this in 2010, and it’s reckoned that the breach cost the company \$800m. The FTC sued Facebook over deceptive changes to privacy settings and settled in 2011, just before its IPO, requiring it to get user consent for certain changes and

---

<sup>10</sup>Snowden revealed some egregious abuses such as the large-scale collection of by GCHQ of Yahoo video chats in Operation Optic Nerve, including intimate video chats [14].

subjecting it to 20 years of audits [181]. The real shock to CEO-land came when Target’s CEO, Gregg Steinhafel, was fired in May 2014 following a hack of more than 100m credit card numbers the previous December; the CIO was also replaced [702]. The C-suite carnage has continued, both in the USA<sup>11</sup> and elsewhere<sup>12</sup> moving cybersecurity steadily up the corporate agenda.

In 2018, California passed a consumer privacy law, the California Consumer Privacy Act (CCPA). This followed a privacy ballot initiative which, if it had gone to a ballot and passed, would have entrenched an even tougher privacy law. The ballot in turn followed the Cambridge Analytica scandal where the Facebook data of 87 million users was harvested without their knowledge or consent and used to target behavioural advertising during the 2016 election campaign. The big tech companies’ defence was to negotiate the new law instead of the ballot initiative, so they could have it amended later, or even trumped by a Federal law. CCPA is somewhat similar to European data-protection law: it empowers consumers to request the deletion of personal information, opt out of its sale, and access it in a format that enables its transfer to third parties. The European right to be forgotten is a non-starter thanks to the US First Amendment. CCPA can be enforced by the state attorney general but also by private action. A really important policy question now is whether this law is progressively copied by other states, or whether Big Tech manages to emasculate it<sup>13</sup>. But the USA is not the only serious player here.

### 26.6.3 Fragmentation?

Since 1998, European law has forbidden companies from sending personal data to organizations in countries where the law does not provide comparable protection or other safeguards – in practice, that means America and India. The first attempt to resolve this was the *Safe Harbour Agreement* whereby a data processor in America or India would promise their European customer to abide by European law. In 2000, the European Commission adopted an executive decision to the effect that this would give ‘adequate protection’. However, it left no practical recourse for EU citizens who felt their rights had been violated.

The case that killed Safe Harbour was brought by Max Schrems, an Austrian lawyer, against Facebook. Following the Snowden revelations, he argued that for Facebook in Ireland (its EU headquarters) to pass his data to the USA for processing was unlawful, as the law and practice of the United States offer no protection against surveillance by the public authorities, specifically the NSA, which can collect it all via Prism. The European Court of Justice agreed and in 2015 it struck down the Safe Harbour principles. The USA and the EU then agreed to replace them with a fresh arrangement, called Privacy Shield, which

---

<sup>11</sup>Amy Pascal of Sony in 2014, Walter Stephan of FACC in 2016, Richard Smith of Equifax in 2017; and maybe we can note Marissa Meyer of Yahoo who forfeited her bonus and stock in 2017, and perhaps even Travis Kavalnick of Uber whose successor publicised a hack that had been covered up.

<sup>12</sup>Dido Harding of TalkTalk, UK, in 2017; Bruce Liang of Integrated Health Information Systems, Singapore, in 2019; and maybe we can count Martin Winterkorn of VW and Rupert Stadler of Audi too, who presided over the company hacking its car emissions.

<sup>13</sup>Their lobbyists are already attacking it, but as I write in 2020, there’s a ballot initiative that would entrench it in California law and put it beyond the grasp of state legislators.

## 26.6. PRIVACY AND DATA PROTECTION

---

adds and ombudsperson to whom an EU citizen can complain if they think the NSA might have spied on them [1474]; Max took this to the European Court of Justice, which duly struck it down in July 2020[1683]. The defendant was the Irish Data Protection Commissioner, who spent almost €3M defending the position that she had the right to look the other way as US tech firms with their EU headquarters in Ireland ride roughshod over privacy law. The court also ruled that privacy authorities have a duty to take action when they receive a complaint. It also made clear that the NSA's right under US law to get free access to the data of people who are not US persons is not consistent with US firms keeping data on EU citizens under US custody and control<sup>14</sup>.

Many companies that process data in the USA had in the meantime fallen back on contract, forcing customers to agree to their personal data being shared before they do business with them. This has a long and sordid history (it's how medical insurers get away with selling your data to drug companies), and the ECJ allowed the continued use of *standard contractual clauses* (SCCs) to protect data. But this isn't straightforward. First, the data controller has to establish that there's an adequate level of protection in the country where the data will be held, and second, you can't simply impose such terms on consumers in the world of the GDPR as coercive consent is specifically disallowed. It is hard to see how US firms can establish adequacy when US law provides unfettered access to foreigners' data on US soil and the Snowden disclosures document the systematic use (and, from the EU law viewpoint, abuse) of this access.

So this is developing into a real fight, with real consequences for how and where the world's server farms are located and controlled. Some of the better-informed firms assume that they will eventually have to process European data in Europe and under European law; Microsoft put a data centre in Germany under the control of a German trustee for a couple of years, but then changed its mind, while Google has done its privacy research and development for some years in Munich. And public opinion in the USA isn't that different from Europe: most Americans think their personal data is less secure now, that the risks of surveillance capitalism outweigh the benefits, that they don't understand what's going on, that they have no control and neither companies nor government are accountable for abuse, but that they just don't have any alternative. Oh, and 20% suffered some kind of online fraud in the last twelve months [144].

Meanwhile, data-protection law is pushing into new areas where it gives a way of responding to abuses. For example, after the Brexit referendum, the UK Information Commissioner fined Facebook £500,000<sup>15</sup> after they let Cambridge Analytica harvest personal data on 87 million people worldwide, and used this to target election ads in both the Brexit referendum and the US 2016 presidential election [957]. As many modern practices in marketing and in political propaganda involve offences under data-protection law, this gives scope for regulatory

<sup>14</sup>There is also a case pending at the European Court of Human Rights, brought by Big Brother Watch against US mass surveillance [420], which has been granted an appeal to the Grand Chamber. If this goes the same way, the ECJ judgment will be extended to those countries that are members of the Council of Europe but not of the EU, such as the UK and Russia.

<sup>15</sup>The UK fine was the maximum allowed under pre-GDPR data-protection law; since then the maximum is 4% of the defendant's turnover, which should bring European penalties into line with American ones.

innovation. The US equivalent is the FTC's use of truth-in-advertising law to punish firms that break their privacy policies or previous agreements about user privacy; and Facebook was in due course fined \$5bn by the FTC. The Electronic Privacy Information Center<sup>16</sup> had been arguing ever since the Cambridge Analytica scandal broke that Facebook had violated the terms of its 2011 settlement with the FTC.

## 26.7 Freedom of Information

Information tends to flow from the weak to the powerful, increasing their power and making it harder for others to hold them to account. As James Madison wrote:

A popular government without popular information or the means of acquiring it is but a prologue to a farce or a tragedy, or perhaps both. Knowledge will forever govern ignorance: And a people who mean to be their own Governors, must arm themselves with the power which knowledge gives.

In the aftermath of Watergate, Congress passed the Freedom of Information Act, and other countries followed; Britain got one in 1997<sup>17</sup>. More radical versions have been tried: tax returns are published in Iceland and in some Swiss cantons, and the practice cuts evasion, as rich men fear the loss of social status that a low declared income would bring. The most radical version is proposed by David Brin, in 'The Transparent Society' [322]. He reasons that the falling costs of data acquisition, transmission and storage will make pervasive surveillance technologies available to the authorities, so the only real question is whether they are available to the rest of us too. He paints a choice between two futures – one in which the citizens live in fear of a Chinese-style policing system and one in which officials are held to account by public scrutiny. He argues that essentially all information should be open – including, for example, all our bank accounts. The cameras will exist: will they be surveillance cams or webcams? Social media often seem to be pushing us in that direction. In any case, Freedom of Information Acts typically let the citizen demand copies of information held by the state unless there's a good reason to withhold it, and help ensure that the flow of information between the citizen and the state isn't entirely one-way.

However, transparency leads to interesting tussles. Many European countries have clean-slate laws whereby most criminal convictions are expunged after a period of time that depends on the severity of the offence, and in 2019 Pennsylvania, Utah and California followed suit [607]. But how can such laws be enforced now that web search engines exist? Do you tag the names of offenders in newspaper accounts of trials with an expiration date, and pass laws compelling search and archive services to respect them? The Google Spain case gives us the answer: someone whose conviction has expired has a right to have it suppressed in searches, although it may remain in the newspaper archive for those who know where to look.

<sup>16</sup>Full disclosure: I'm a member of their advisory board.

<sup>17</sup>Tony Blair later described it as his biggest mistake.

That's one example of the shifting boundary between data protection and freedom of information. Another has been the monitoring of former child sex offenders, with laws in some states requiring that registers of offenders be publicly available, and riots in the UK following the naming of some former offenders by a Sunday newspaper and at least one innocent person being lynched. A third is the release of crime statistics: home owners object to their neighbourhood being stigmatised, and if the data are too granular there may be some risk of individual victims being identified. For further examples, see Section 11.1 on inference security.

## **26.8 Summary**

Public policy is increasingly entangled with the work of the security engineer. The largest single concern of governments, if we measure it in dollar terms, is intelligence; a typical government spends a hundred times more money collecting information on its enemies, real and potential, than it does on fighting cyber-crime. Intelligence collection is also in conflict with both defensive security and with privacy, both of which have historically come second. However, since the Snowden revelations made clear the scale of US data collection worldwide, and of Five Eyes operations against allied countries, the balance has started to shift, and the effects have propagated through privacy and data protection law, albeit slowly and with so far little effect on the agencies themselves. Perhaps when the analysis is done, Snowden's effect on the agencies' capabilities will be largely technical (through getting people to use cryptography more, and more intelligently) while the policy effect may be to curb some of the excesses of 'surveillance capitalism' by making privacy more salient to more people. The strains between the US and European ways of dealing with privacy are becoming more significant and in the medium term we may see more localisation – where US companies have to keep data on EU citizens on servers in Europe and perhaps even under the control of European trustees. Other countries are starting to follow suit.

Censorship is a real issue; some countries, like China, ban many of the large US service firms outright, while more and more are demanding that they take down not just abusive material but also material that offends local political sensitivities. The Internet still makes it harder for countries that won't go as far as China to censor subversive content, but much of the optimism we had ten years ago has dissipated with the failure of the Arab Spring. Even the developed countries push the large service firms to moderate and filter user-generated content at scale, and despite the cost and complexity, it's becoming universal except on end-to-end encrypted services. It's now 25 years since AOL barred users from living in Scunthorpe, and large-scale filtering still raises a host of policy issues whether we're talking about copyright, radicalisation, harassment or fake news.

The security-industrial complex, whose growth was fuelled by the climate of fear whipped up after the 9/11 attacks, has got a second wind from China and the Arab Spring, as the world's authoritarians buy surveillance systems to keep track of their populations. This has led to the proliferation of computer and

network exploitation tools that erode our security, our liberty, and our quality of life. This proliferation is aided and abetted by Western governments who should know better, and is bound to be extended as social media firms and others are co-opted into ever more content screening as a condition of doing business. Understanding and pushing back on the surveillance ecosystem while mitigating online harms is the highest priority for security engineers who have the ability to get involved in public life – whether directly, or via our writing and teaching. And research also helps. Individual academics can't hope to compete with national leaders in the mass media, but the careful accumulation of data and knowledge over the years can and will undermine their excuses. I don't mean just knowledge about why extreme airport screening measures are a waste of money; we also must disseminate knowledge about the economics and psychology that underlie maladaptive government behaviour, and its terrible consequences in terms of spending money on security theatre that should have been spent on pandemic preparedness.

## **Research Problems**

Technology policy involves a complex interplay between science, engineering, psychology, law and economics. There is still too little serious cross-disciplinary research, and initiatives which speed up this process are almost certainly a good thing. Since 2002 I've worked to build up the security-economics research community; and since 2008 we've run an annual workshop on security and human behaviour to engage psychologists, anthropologists and philosophers too. But we need much, much more. Where are the historians, the sociologists and the political scientists? (And perhaps if there's a fourth edition, we'll add the philosophers.)

## **Further Reading**

It's extraordinarily easy for technology policy arguments to get detached from reality, and many of the scares conjured up to get attention and money (such as 'cyberterrorism') are the modern equivalent of the monsters that appeared on medieval maps to cover up the cartographer's ignorance. An engineer should look for primary sources – from material written by experienced insiders such as R.V. Jones [992] to the thousands of documents leaked by Ed Snowden. As for the use of information warfare techniques in the Brexit referendum and the 2016 US election, Carole Cadwalladr's movie 'The Great Hack' is unmissable.

There's a good book on the history of wiretapping and crypto policy by Whit Diffie and Susan Landau, who had a long involvement in the policy process [558], and an NRC study on cryptography policy was also influential [1411]. There's a video on my website of the history of the crypto wars from a European perspective.

The history of export control is tied up with Soviet attempts to buy US computer, semiconductor and energy technology during the 1970s and 80s, and the US and French intelligence community's work to block them and feed them

## **26.8. SUMMARY**

---

misleading information: see the memoir on Gus Weiss, a CIA maverick involved in this work [723].

Resources on online censorship include Reporters without Borders, who publish a ‘Handbook for bloggers and cyber-dissidents’ on how to circumvent censorship, with a number of case histories of how blogging has helped open up the media in less liberal countries [1594].

The standard work on computer forensics is by Tony Sammes and Brian Jenkinson [1644], while Privacy International has a survey of mobile phone forensics [1555] and the Department of Justice’s “Guidelines for Searching and Seizing Computers” also bear some attention [550]. For early computer crime case histories, see Peter Neumann [1429] and Dorothy Denning [539]. The standard work on computer evidence in the common law countries is by Stephen Mason [1236].

On the topic of privacy versus data protection, there is a huge literature but no concise recent guide that I know of. Recent material can be found on the web sites of organizations such as EPIC [631], EFF [618], FIPR [708] and EDRI [643], and of Max Schrems [1683].

As for the policy problems around the filtering of inflammatory content and propaganda, the two most thought-provoking books for me are those by Tim Wu [2050] and Yochai Benkler [227], while Facebook’s former CISO Alex Stamos now discusses the tech companies’ view of filtering political ads [999].

Finally, the definitive story of the Cambridge Analytica scandal is told in the book by the whistleblower Chris Wylie [2052], and in the journalism by Carole Cadwalladr based on information that he and others supplied [363].

## Chapter 27

# Secure Systems Development

My own experience is that developers with a clean, expressive set of specific security requirements can build a very tight machine. They don't have to be security gurus, but they have to understand what they're trying to build and how it should work.

– Rick Smith

When it comes to being slaves to fashion, American managers make adolescent girls look like rugged individualists.

– Geoff Nunberg

The fox knows many things; the hedgehog one big thing.

– Archilochus

### 27.1 Introduction

So far we've discussed a great variety of security applications, technologies and concerns. If you're a working engineer, manager or consultant, paid to build or maintain a system with some security assurance requirements, you will by now be looking for a systematic way to go about it. This brings us to such topics as risk analysis, system engineering methodology, and, finally, the secret sauce: how you manage a team to write secure code.

The secret is that there isn't actually a secret, whether sauce or anything else. Lots of people claim there is one and get religious fervour for the passion of the moment, from the Orange Book in the 1980s to Agile Development now. But the first take offered on this was the right one. In the 1960s Fred Brooks led the team on the world's first really large software project, the operating system for the IBM S/360 mainframe. In his classic book "The Mythical Man-Month" he describes all the problems they struggled with, and his conclusion is that "there is no silver bullet" [328]. There's no magic formula that makes

an intrinsically hard job easy. There's also the famous line from Archilochus at the head of this chapter: the fox knows many things, while the hedgehog knows one big thing. Managing secure development is fox knowledge rather than hedgehog knowledge. An experienced security engineering manager has to know thousands of little things; that's why this book is so fat! And the security engineering manager's job is getting harder all the time as software gets everywhere and starts to interact with safety.

In 2017, I changed the way I teach undergraduates at Cambridge. Up till then we'd taught security courses separately from software engineering, with the latter focusing on safety. But most real-world systems require both, and they're entangled in complex ways. Both safety and security are emergent properties that really have to be baked in from the beginning. Both involve systematic thinking about what can go wrong, whether by accident or as a result of malice. Accidents can expose systems to attacks, and attacks can degrade systems so they become dangerous. The course was developed further by my colleague Alastair Beresford while I was on sabbatical in 2019, and the 2020 course on Software and Security Engineering is now online as ten video lectures, thanks to the pandemic [89]. That course is designed to give our first-year undergraduates a solid foundation for later work in security, cryptography and software engineering. Like this book, it introduces the basics, from definitions through the basics of protocols and crypto, then the importance of human and organizational issues as well as technical ones, illustrated with case histories. It discusses how you set goals for safety and security, how you manage them as a system evolves, and how you instil suitable ways of thinking and working into your team. Success is about attitudes and work practices as well as skills.

The two questions you have to ask are, “Are we building the right system?” and “Are we building it right?” In the rest of this chapter I’m going to start with how we assess and manage risks – to both safety and security; and then go on to discuss how we build systems, once we’ve got a specification to work to. I’ll then discuss some of the hazards that arise as a result of organisational behaviour – a real but often ignored kind of insider threat.

## 27.2 Risk Management

At the heart of both safety engineering and security engineering lie decisions about priorities: how much to spend on protection against what. Risk management must be done within a broader framework of managing all the risks to an enterprise or indeed to a nation. That is often done badly. The coronavirus crisis should have made it obvious to everyone that although pandemics were at the top of the risk register of many countries, including the UK, most governments spent much more of their resilience budget on terrorism, which was several places down the list. Countries with recent experience of SARS or MERS, such as Taiwan and South Korea, did better: they were ready to test residents and trace contacts at scale, and responded quickly. Britain wasted two months before realising the disease was serious, at a cost of tens of thousands of lives.

So what actually is a *risk register*? A common methodology, as used by

the governing body of my university, is to draw up a list of things that could go wrong, giving them scores of 1 to 5 for seriousness and for probability of occurrence, and multiplying these together to get a number between 1 and 25. For example, a university might rate ‘loss of research contract income due to economic downturn’ at 5/5 for seriousness if 20% of its income is from that source, and rate ‘probability’ at 4/5 as downturns happen frequently but not every year, giving a raw product of 20. You then write down the measures you take to mitigate each of these risks, and have an argument in a risk committee about how well each risk is mitigated. For example, you control the risk of variable research contract income by making a rule that it can be used to hire only contract staff, not tenured faculty; you might then agree that this rule cuts that risk from 20 to 16. You then rank all the risks in order and assign one senior officer to be the owner of each of them.

National risk assessments are somewhat similar: you rate each possible bad event (pandemic, earthquake, forest fire, terrorist attack, ...) by how many people it might kill (millions? thousands? dozens?) and then rate it for probability by how many you expect each century. The UK national risk register, for example, put pandemic influenza at the top, with a 5 for severity (could kill up to 750,000) and a 4 for likelihood, saying in 2017: “one or more major hazards can be expected to materialise in the UK in every five-year period. The most serious are pandemic influenza, national blackout and severe flooding” [361]. You then work out what’s reasonably practical by way of mitigation, be it quarantine plans and PPE stockpiles for a pandemic, or building codes and zoning to limit the damage from floods and earthquakes. You do the cost-benefit analysis and turn priorities into policy. You can get things wrong in various ways. The UK largely ignored pandemics because the National Security Council had been captured by the security and intelligence agencies; they prioritised terrorism, and the health secretary was not a regular attendee [1848]. I already discussed terrorism in section 26.3; here I’ll just add that another aspect of the failure was policy overshoot. When 9/11 taught the world that terrorist attacks can kill thousands rather than just dozens, and the agencies got a lot more of the resilience budget, it made them greedy: they started talking up the risk of terrorists getting hold of a nuke so they’d have an even scarier threat on the register to justify their budgets.

In business too you can find that both political behaviour and organisational behaviour get in the way of rational risk management. But you often have real data on the more common losses, so you can attempt a more quantitative approach. The standard method is to calculate the *annual loss expectancy* (ALE) for each possible loss scenario, as the expected loss multiplied by the number of incidents expected in an average year. A typical ALE analysis for a bank’s IT systems might have several hundred entries, including items such as we see in Figure 27.1.

Note that while accurate figures are likely to be available for common losses (such as ‘teller takes cash’), the incidence of low-probability high-risk losses such as a large money-transfer fraud is largely guesswork – though you can sometimes get a rough sanity check by asking for insurance quotes.

ALEs have long been standardized by NIST as the technique to use in US government procurements. The UK government uses a tool called CRAMM for

Loss type	Amount	Incidence	ALE
SWIFT fraud	\$50,000,000	.005	\$250,000
ATM fraud (large)	\$250,000	.2	\$100,000
ATM fraud (small)	\$20,000	.5	\$10,000
Teller takes cash	\$3,240	200	\$648,000

Figure 27.1: – items of annualized loss expectancy (ALE)

systematic analysis of information security risks, and the modern audit culture is spreading such tools everywhere. But the process of producing such a table for low-probability threats tends to be just iterative guesswork. The consultants list all the threats they can think of, attach notional probabilities, work out the ALEs, add them up, and find that the bank's ALE exceeds its income. They then tweak the total down to whatever will justify the largest security budget that their client the CISO has said is politically possible. I'm sorry if this sounds a bit cynical; but it's what often seems to happen. The point is, ALEs may be of some value, but you need to understand what parts are based on data, what parts on guesswork and what parts on office politics.

Product risks are different. Different industries do things differently because of the way they evolved and the history of regulation. The rules for each sector, whether cars or aircraft or medical devices or railway signals, have evolved in response to accidents and industry lobbying. Increasingly, the European Union is becoming the world's safety regulator as it's the biggest market, as Washington cares less about safety than Brussels does, and as it's simpler for OEMs to engineer to EU safety specifications than to have multiple products. I'll discuss safety and security certification in more detail in the next chapter. For present purposes, software for cars, planes and medical devices must be developed according to approved procedures, subjected to analyses we'll discuss later, and tested in specific ways.

Insurance can be of some help in managing large but unlikely risks. But the insurance business is not completely scientific either. Your insurance premiums used to give some signal of the risk your business was running, especially if you bought cover for losses of eight figures or above. But insurance is a cyclical industry, and since about 2017 a host of new companies have started offering insurance against cybercrime, squeezing the profits out of the market. As a result, customers will no longer put up with intrusive questionnaires, let alone site visits from assessors. So most insurers' ability to assess risk is now limited; I will discuss the mechanics of what they do further in section 28.2.9. They are also wary of correlated risks that give rise to many claims at once, as that would force them to hold greater reserves; as some cyber risks are correlated, policies tend to either exclude them or be relatively expensive [275]. (The coronavirus crisis is teaching firms about correlated risk as some insurers refuse to pay up on business-interruption risk policies – even those that explicitly mention the risk of staff not being able to get to the office because of epidemics; businesses are asking insurers in turn what the point of insurance is.)

Actuarial risks aside, a very important reason for large companies to take out insurance cover – and for much other corporate behaviour – is to protect

executives, rather than shareholders. The risks that are being tackled may seem on the surface to be operational but are actually legal, regulatory and PR risks. Directors demand liability insurance, and under UK and US law, professional negligence occurs when a professional fails to perform their responsibilities to the level required of a reasonably competent person in their profession. So negligence claims are assessed by the current standards of the industry or profession, giving a strong incentive to follow the herd. This is one reason why management is such a fashion-driven business (as per the quote at the head of this chapter). This spills over into the discourse used to justify security budgets. During the mid 1980's, everyone talked about hackers (even if their numbers were tiny). From the late 80's, viruses took over the corporate imagination, and people got rich selling antivirus software. In the mid-1990s, the firewall became the star product. The late 1990s saw a frenzy over PKI. By 2017 it was blockchains. Amidst all this hoopla, the security professional must keep a level head and strive to understand what the real threats are.

We will return to organisational behaviour in a later section. First, let's see what we can learn from safety engineering.

## 27.3 Lessons from safety-critical systems

*Critical computer systems* are those in which a certain class of failure is to be avoided if at all possible. Depending on the class of failure, they may be safety-critical, business-critical, security-critical, or critical to the environment. Obvious examples of the safety-critical variety include flight controls and automatic braking systems. There's a large literature on this subject, and a lot of methodologies have been developed to help manage risk intelligently.

### 27.3.1 Safety engineering methodologies

Safety engineering methodologies, like classical security engineering, tend to work systematically from a safety analysis through to a specification, then to a product, and assume you're building safety in from the start rather than trying to retrofit it. The usual procedure is to identify hazards and assess risks; decide on a strategy to cope with them (avoidance, constraint, redundancy ...); trace the hazards to hardware and software components which are thereby identified as critical; identify the operator procedures which are also critical and study the various applied psychology and operations research issues; set out the safety functional requirements which specify what the safety mechanisms must do, and safety integrity requirements that specify the likelihood of a safety function being performed satisfactorily; and finally decide on a test plan. The outcome of testing is not just a system you're confident to run live, but an integrated part of a *safety case* to justify running it. The basic framework is set out in standards such as ISO 61508, a basic safety framework for relatively simple programmable electronics such as the control systems for chemical plants. This has been extended with more specialised standards for particular industries, such as ISO 26262 for road vehicles.

This safety case will provide the evidence, if something does go wrong, that

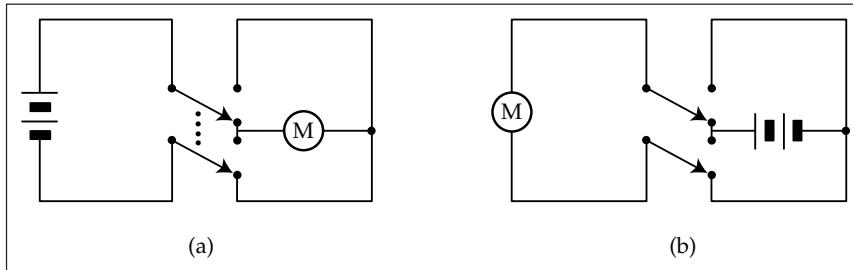


Figure 27.2: – hazard elimination in motor reversing circuit

you exercised due care. It will typically consist of the hazard analysis, the safety functional and integrity requirements, and the results of tests (both at component level and system level) which show that the required failure rates have been achieved. The testing may have to be done by an accredited third party; with motor vehicles firms get away with the safety case being done by a different department in the same company, with independent management. Vehicles are a more complex case because of their supply chains. At the top is the brand, whose badge you see on the front of the car. Then there's the *original equipment manufacturer* (OEM) which in the case of cars is usually the same company, but not always; in other industries the brand and the OEM are quite separate. A modern car will have components from dozens of manufacturers, of which the Tier 1 suppliers who deal directly with the brand do much of the research and development work but get components from other firms in turn. In the car industry, the brand puts the car through type approval and carries the primary liability, but demands indemnities from component suppliers in case things go wrong (the law in most countries does not allow you to disclaim liability for death and injury). The brand relies on the supply chain for significant parts of the safety functionality and integrity and thus for the safety case. There are also tensions: as we already noted, safety certification can prevent the timely application of security patches. Let's now look at common safety engineering methods and what they can teach us.

### 27.3.2 Hazard analysis

In an ideal case, we might be able to design hazards out of a system completely. As an example, consider the motor reversing circuits in Figure 27.2. In the design on the left, a double-pole double-throw switch reverses the current passing from the battery through the motor. However, this has a potential problem: if only one of the two poles of the switch moves, the battery will be shorted and a fire may result. The solution is to exchange the battery and the motor, as in the modified circuit on the right. Here, a switch failure will only short out the motor, not the battery. Safety engineering is not just about correct operation, but about correct failure too.

Hazard elimination is useful in security engineering too. We saw an example in the early design of SWIFT in section 12.3.2: there, the keys used to authenticate transactions between one bank and another were exchanged between the

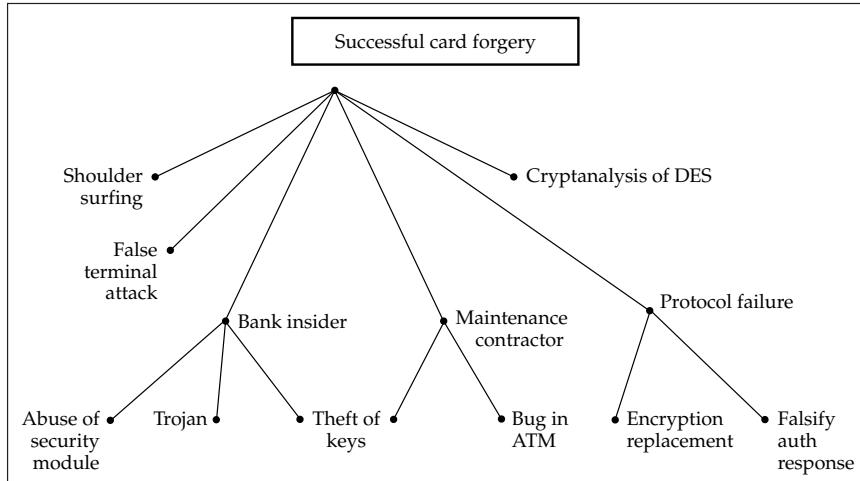


Figure 27.3: – a threat tree

banks directly, so SWIFT did not have the means to forge a valid transaction and its staff and systems had to be trusted less. In general, minimizing the trusted computing base is an exercise in hazard elimination. The same applies in privacy engineering too. For example, if you're designing a contact tracing app to monitor who might have infected whom in an epidemic, one approach is to have a central database of everyone's mobile phone location history. However that has obvious privacy hazards, which can be reduced by keeping a Bluetooth contact history on everyone's mobile phone instead, and uploading the contact history of anyone who calls in sick. You then have a policy decision to take between better privacy and better tracing.

### 27.3.3 Fault trees and threat trees

Once you have eliminated as many hazards as possible, the next step is to identify failures that could cause accidents. A common top-down way of identifying the things that can go wrong is *fault tree analysis* where a tree is constructed whose root is the undesired behavior and whose successive nodes are its possible causes. This top-down approach is natural where you have a complex system with a small number of well-known bad outcomes that you have to avoid. It carries over in a natural way to security engineering. Figure 27.3 shows an example of a fault tree (or *threat tree*, as it's often called in security engineering) for fraud from automatic teller machines.

Threat trees are used in the US Department of Defense. You start out from each undesirable outcome, and work backwards by writing down each possible immediate cause. You then recurse by adding each precursor condition. By working round the tree's leaves you should be able to see each combination of technical attack, operational blunder, physical penetration and so on which could break your security policy. The other nice thing you get from this is a visualisation of commonality between attack paths, which makes it easier to reason about how to disrupt the most attacks with the least effort. In some

variants, attack branches have countermeasure sub-branches, which may have counter-countermeasure attack branches, and so on, in different colours for emphasis. A threat tree can amount to an attack manual for the system, so it may be highly classified, but it's a DoD requirement – and if the system evaluators or accreditors can find significant extra attacks, they may fail the product.

#### 27.3.4 Failure modes and effects analysis

Returning to the safety-critical world, another way of doing hazard analysis is *failure modes and effects analysis* (FMEA), pioneered by NASA, which is bottom-up rather than top-down<sup>1</sup>. This involves tracing the consequences of a failure of each of the system's components all the way up to the effect on the mission. This is the natural approach in systems with a small number of well-understood critical components or subsystems, such as aircraft. For example, if you're going to fly a plane over an ocean or mountains where you can't glide to an airport in the case of engine failure, then engine power is critical. You therefore study the mean time to failure of your powerplant and its failure modes, from a broken connecting rod to running out of fuel. You insist that single-engine aircraft use reliable engines and you regulate the maintenance schedules; planes have more than one fuel tank. When carrying a lot of passengers, you insist on multi-engine aircraft and drill the crews to deal with engine failure.

An aerospace example of people missing a failure mode that turned out to be critical is the 1986 loss of the space shuttle Challenger. The O-rings in the booster rockets were known to be a risk by the NASA project manager, and damage had been found on previous flights; meanwhile the contractor knew that low temperatures increased the risk; but the concerns did not come together or get through to NASA's top management. An O-ring, made brittle by the cold, failed – causing the loss of the shuttle and seven crew. On the resulting board of inquiry, the physicist Richard Feynman famously demonstrated this on TV by putting a sample of O-ring in a clamp, freezing it in iced water and then showing that when he released it, it remained dented and did not spring back [1612]. This illustrates that failures are often not just technical but also involve how people behave in organisations: when protection mechanisms cross institutional boundaries, as for example with cars, you need to think of the law and economics as well as just the engineering. Such problems will become much more complex as we move towards autonomous vehicles, which will rely on all sorts of third-party services and infrastructure.

#### 27.3.5 Threat modelling

Both fault trees and FMEA depend on the analyst understanding the system really well; they are hard to automate, not fully repeatable and can be up-ended by a subtle change to a subsystem. So a thorough analysis of failure modes will often combine top-down and bottom-up approaches with some methods specific

---

<sup>1</sup>FMEA is bottom-up in the technical sense that the analysis works up from individual components, but its actual management often has a top-down flavour as you start work on the safety case once you have an outline design and refine it progressively as the design is evolved into a product.

### **27.3. LESSONS FROM SAFETY-CRITICAL SYSTEMS**

---

to the application that people have learned over time. Many industries now have to rethink their traditional safety analysis methods to incorporate security.

In car safety, complex supply chains mean we have to do multiple interlocking analyses of vehicles and their subsystems. A traditional subsystem analysis might work through the failure modes of headlamps, since losing them while driving at night can lead to an accident. As well as mitigating the risk of a lamp failure by having two or more lamps, you worry about switch failure, and when the switch becomes electronic you build a fault tree of possible hardware and software faults. When we extend this from safety to security, we think about whether an attacker might take over the entertainment system in a car, and use it to send a malicious ‘lamp off’ message on the CAN bus once the car is moving quickly enough for this to be dangerous. This analysis may lead to a design decision to have a firewall between the cabin CAN bus and the powertrain CAN bus. (This is the worked example in the new draft ISO 21434 standard for cybersecurity in road vehicles [962].)

More generally, the shift from safety to security means having to think systematically about insiders. Just as double-entry bookkeeping was designed to be resilient against a single dishonest clerk and has been re-engineered against the similar threat of a clerk with malware on their PC, so modern large-scale systems are typically designed to limit the damage if a single component is compromised. So how can you incorporate malicious insiders into a threat model? If you’re using FMEA, you can just add an opponent at various locations, as with our malicious ‘lamp off’ message. As for more complex systems, the methodology adopted by Microsoft following its big push in 2003 to make Windows and Office more secure is described by Frank Swiderski and Window Snyder [1851]. Rather than being purely top-down or bottom-up, this is a meet-in-the-middle approach. The basic idea is that you list not just the assets you’re trying to protect (ability to do transactions, access to confidential data, whatever) but also the assets available to an attacker (perhaps the ability to subscribe to your system, or to manipulate inputs to the smartcard you supply him, or to get a job at your call center). You then trace the attack paths through the system, from one module to another. You try to figure out what the trust levels might be; where the barriers are; and what techniques, such as spoofing, tampering, repudiation, information disclosure, service denial and elevation of privilege, might be used to overcome particular barriers. The threat model can be used for various purposes at different points in the security development lifecycle, from architecture reviews through targeting code reviews and penetration tests.

There are various ways to manage the resulting mass of data. An elementary approach is to construct a matrix of hazards against safety mechanisms, and if the safety policy is that each serious hazard must be constrained by at least two independent mechanisms, then you can check for two entries in each of the relevant columns. So you can demonstrate graphically that in the presence of the hazard in question, at least two failures will be required to cause an accident. An alternative approach, *system theoretic process analysis* (STPA), starts off with the hazards and then designs controls in a top-down process, leading to an architectural design for the system; this can be helpful in teasing apart interacting control loops [1150]. Such methodologies go across to security engineering [1556]. One way or another, in order to make the complexity

manageable, you may have to organise a hierarchy of safety and security goals. The security policies discussed in Part II of this book may give you the beginnings of an answer for the applications we discussed there, and some inspiration for others. This hierarchy can then drive a risk matrix or risk treatment plan depending on the terminology in use in your industry.

### 27.3.6 Quantifying risks

The safety-critical systems community has a number of techniques for dealing with failure and error rates. Component failure rates can be measured statistically; the number of bugs in software can be tracked by techniques I'll discuss in the next chapter; and there is a lot of experience with the probability of operator error at different types of activity. The bible for human-factors engineering in safety-critical systems is James Reason's book '*Human Error*'; I discussed in Chapter 3 the rising tide of research in security usability through the 2010s as the lessons from the safety world have started to percolate into our field.

The error rate in a task depends on its familiarity and complexity, the amount of pressure and the number of cues to success. Where a task is simple, performed often and there are strong cues to success, the error rate might be 1 in 100,000 operations. However, when a task is performed for the first time in a confusing environment where logical thought is required and the operator is under pressure, then the odds can be against successful completion. Three Mile Island and Chernobyl taught nuclear engineers that no matter how many design walkthroughs you do, it's when the red lights go on for real that the worst mistakes get made. The same lesson has come out of one air accident investigation after another. When dozens of alarms go off at once, there's a fair chance that someone will push the wrong button. One guiding principle is to default to a safe state: to damp down a nuclear reaction, to return an aircraft to straight and level flight, or to bring an autonomous vehicle to a stop at the side of the road. No principle is foolproof, and a safe state may be hard to measure. A vehicle can find it hard to tell where the side of the road is if there's a grass verge; and in the Boeing 737Max crashes (which I describe in detail in section 28.2.4) the flight control computer tried to keep the plane level but was confused by a faulty angle-of-attack sensor and dived the plane into the ground instead.

Another principle of safety usability in an emergency is to keep the information given to operators, and the controls available for them to use, both simple and intuitive. In the old days, each feed went to a single gauge or dial and there was only so much space for them. The temptation nowadays is to give the operator everything, because you can. In the old days, designers knew that an emergency would give the pilots tunnel vision so they put the six instruments they really needed right in the middle. Nowadays there can be fifty alarms rather than two and pilots struggle to work out which screen on which menu of the electronic flight information system to look at. It is much broader than aviation. A naval example is the 2017 collision of the USS McCain in the Straits of Singapore, where UI confusion was a major factor. Steering control was shifted to the wrong helm station and an engine was not throttled back in time, resulting in an uncommanded turn to port across a busy shipping lane,

### **27.3. LESSONS FROM SAFETY-CRITICAL SYSTEMS**

---

impact with a chemical tanker, and the death of ten sailors [1929].

So systems that are not fully autonomous must remain controllable, and for that the likely human errors need to be understood. Quite a lot is known about the cognitive biases and other psychological factors that make particular types of error more common; we discussed them in Chapter 3, and a prudent engineer will study how they work out in their field. Errors are rare in frequently-performed tasks at which the operator has developed some skill, and are more likely when operators are stressed and surprised. This starts to get us out of the territory of risk, where the odds are known, and into that of uncertainty, where they're not.

In security systems, too, the most egregious blunders can be expected in important but rarely performed tasks. Security usability isn't just about presenting a nice intuitive interface to the end-user. It should present the risks in a way that accords with common mental models of threat and protection, and the likely user reactions to stress should lead to safe outcomes.

It is important to be realistic about the skill level of the people who will perform each critical task and any known estimates of the likelihood of error. An airplane designer can rely on a predictable skill level from anyone with a commercial pilot's license, and a shipbuilder knows the strengths and weaknesses of an officer in the merchant marine. Cars can and do get operated by drivers who are old and frail, young and inexperienced, distracted by passengers, or under the influence of alcohol. At the professional end of things, usability testing can be profitably integrated with staff training: when pilots go for their refresher courses in the simulator, instructors throw all sorts of combinations of equipment failure, bad weather, cabin crisis and air-traffic-control confusion at them. They observe what combinations of stress result in fatal accidents, and how these differ across cockpit types. Such data are valuable feedback to cockpit designers. In aviation, the incentives for safe operation are sufficiently strong and well aligned, and the scale is large enough, to support a learning system. Even so, there are expensive disasters, such as the Boeing 737Max flight control software. This not only had at least one serious bug, but escaped a proper failure modes and effects analysis because the engineers responsible – under pressure from their managers to complete the project on time – wrongly assumed that pilots would be able to cope with any failure [89]. As a result, the software relied on a single angle-of-attack sensor rather than using the two sensors with which the aircraft was fitted, and sensor failure led to fatal accidents<sup>2</sup>.

When testing the usability of redundant systems, you need to pay attention to *fault masking*: if the output is determined by majority voting between three processors, and one of them fails, then the system will continue to work fine – but its safety margin will have been eroded, perhaps in ways the operators won't understand properly. Several air crashes have resulted from flying an airliner with one of the cockpit systems out of action; although pilots may be intellectually aware that one of the data feeds to the cockpit displays is unreliable, they may rely on it under pressure by reflex rather than checking with other instruments. So you have to think hard about how faults can remain

---

<sup>2</sup>Aviation safety standards such as DO178 and DO254 generally require diversity in measurement type, physics, processing characteristics in addition to redundancy to mitigate common-mode failures.

visible and testable even when their immediate effects are mitigated.

Another lesson from safety-critical systems is that although a safety requirements specification and test criteria will be needed as part of the safety case for the lawyers and regulators, it is good practice to integrate both of them with the mainstream product documentation. If the safety case is separate, then it's easy to sideline it after approval and fail to maintain it properly. (This was a factor in the Boeing 737Max disaster as the usability assumptions underlying the safety case for the flight control software were not updated from the previous model of 737.) The move from project-based software management to agile methodologies, and via DevOps to DevSecOps, is finally starting to embed security management into the way products evolve. We will discuss this in the next section.

Finally, safety is like security in that it really has to be built in as a system is developed, rather than retrofitted. The main difference between the two is in the failure model. Safety deals with the effects of random failure, while in security we assume a hostile opponent who can cause some of the components of our system to fail at the least convenient time and in the most damaging way possible. People are naturally more risk-averse in the presence of an adversary; I will discuss this in section 28.4. A safety engineer will certify a critical flight-control system with an MTBF of  $10^9$  hours; a security engineer has to worry whether an adversary can force the preconditions for that one-in-a-billion failure and crash the plane on demand.

In effect, our task is to program a computer that gives answers which are subtly and maliciously wrong at the most inconvenient moment possible. I've described this as ‘programming Satan’s computer’ to distinguish it from the more common problem of programming Murphy’s [113]. This is one of the reasons security engineering is hard: Satan’s computer is harder to test [1668].

## **27.4 Prioritising protection goals**

If you’ve a project to create an entirely new product, or to radically change an existing one, it’s an idea to spend some time thinking through the protection priorities from first principles. A careful safety analysis or threat modelling exercise can provide some numbers to inform this. When developing a safety case or a security policy in detail, it’s essential to understand the context, and much of this book has been about the threat models relevant to a wide range of applications. You should try to refine numerical estimates of risk from the environment or context as well.

In the case of a business system, analysis will hinge on the tradeoff between risk and reward. Security people often focus too much on the former. If your firm has a turnover of \$10m, gross profits of \$1m and theft losses of \$150,000, you might make a loss-reduction pitch about ‘how to increase profits by 15% by stopping theft’; but if you could double the turnover to \$20m, then the shareholders would prefer that even if it triples the losses to \$450,000. Profit is now \$1.55m, up 85%, rather than 15%. This is borne out by the experience of online fraud engines. When discussing fraud management strategies with a

## 27.4. PRIORITISING PROTECTION GOALS

number of retailers, I noticed that the firms who got the best results were those where the fraud management team reported to sales rather than finance. A typical bricks-and-clicks retailer in the UK might decline something like 4% of offered shopping baskets because the fraud engine alerts at the combination of goods, delivery address and payment details. So if you can improve the fraud engine and reject only 3%, that's 1% more sales – a prospect to light up your Chief Marketing Officer's eyes. But if the fraud team reports instead to the Chief Financial Officer, they're likely to be seen as a cost rather than as an opportunity.

Similarly, the site reliability engineers of online services have learned not to make a system too reliable. If local Internet availability is only 99%, then a service that's up 99.9% of the time will be fine; there's no point spending millions more to hit 99.99% if none of your users will notice the difference. You're better off deliberately setting an 0.1% *error budget* which you can use productively, such as by causing occasional deliberate failures to exercise your resilience mechanisms [236]. This brings me to one of the open debates in security management: should one aim at having no CVEs open in any of the software on which one relies? The tick-box approach is to say 'Of course there must be no open CVEs', but that may impose a rather high compliance cost. If you're Google, and wrote all your own infrastructure, maybe you can aim at that; many firms can't and have to prioritise. I'll discuss CVEs in more detail in section 27.5.7.1 later.

So don't trust people who can only talk about 'tightening security'. Often it's too tight already, and what you really need to do is just focus it slightly differently. In the first edition of this book, I presented a case study of self-service checkout at supermarkets. Twenty years ago, a number of supermarkets started to introduce self-checkout lanes. Some started to obsess about losses, and let security get in the way of usability by aggressively challenging customers about product weight. One of the stores that got an advantage started with a more forgiving approach which they tuned up gradually in the light of experience. Eventually the industry figured out how to operate self-checkout lanes, but the quality of the implementation still varies significantly. By early 2020, the pioneers are small convenience stores like Lifvs in Sweden which have no staff; you open the store's door with an app, scan your purchases and pay online. Amazon was also experimenting with fully self-service food stores. We saw the next 20 years of innovation crammed into the few months of the 2020 coronavirus lockdown; by June, other supermarkets have been urging us to download their scanning app, scan our purchases as we pick them, charge them to a card, and just go.

Many modern business models were once considered too risky, starting with the self-service supermarket itself back in the days when grocers kept all the goods behind the counter. Everyone thought Richard Sears would go bust when he adopted the slogan 'Satisfaction guaranteed or your money back' in the 1880s, yet he invented the modern mail-order business. In business, profit is the reward for risk. But entrepreneurs who succeed may have to improve security quickly. One recent example is the videoconferencing platform Zoom – which grew from 20 million users to 200 million in March 2020, and changed in the process from an enterprise platform into something more like a public utility – forcing them

into a major security engineering effort [1763].

Trade-offs in safety are harder. Logically, the value of a human life in a developed country might be a few million dollars, that being an average person's lifetime earnings. However our actual valuation of a human life as revealed by safety behaviour varies from about \$50,000 for improvements to road junctions, up to over \$500m for train protection systems – and that's just in the context of transport policy. The variance in health policy is even greater, with costs per life saved ranging from a few hundred dollars for flu jabs and some cancer screening to billions for the least effective interventions [1869]; in other safety contexts, domestic smoke alarms cost a few hundred dollars per life saved while the number for the “war on terror” is in the billions [1350]. The reasons for this irrationality are fairly well understood – I discussed the psychology in section 3.2.5 and the policy aspects in 26.3.3. Safety preferences can be changed very sharply by the threat of hostile action; people may completely ignore a 1-in-10,000 risk of being killed by poorly-designed medical devices until there's a possibility that the devices might be hacked, at which point even a 1-in-10,000,000 risk becomes scary. I discuss this phenomenon in section 28.4.

## 27.5 Methodology

Software projects usually take longer than planned, cost more than budgeted and have more bugs than expected<sup>3</sup>. By the 1960s, this had become known as the *software crisis*, although the word ‘crisis’ may be inappropriate for a state of affairs that has now lasted, like computer insecurity, for two generations. Anyway, the term *software engineering* was proposed by Brian Randall in 1968 and defined to be:

Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

The pioneers hoped that the problem could be solved in the same way we build ships and aircraft, with a foundation in basic science and a framework of design rules [1420]. Since then there's been a lot of progress, but the results have been unexpected. Back in the late 1960s, people hoped that we'd cut the number of large software projects failing from the 30% or so that was observed at the time. But we still see about 30% of large projects failing – the difference is that the failures are much bigger. Modern tools get us farther up the complexity mountain before we fall off, but the rate of failure is set by company managers' appetite for risk. We'll discuss this further in the section on organisational behaviour at the end of this chapter.

Software engineering is about managing complexity, of which there are two kinds. There is the *incidental complexity* involved in programming using inappropriate tools, such as the assembly languages which were all that some early machines supported; programming a modern application with a graphical user interface in such a language would be impossibly tedious and error-prone. There

---

<sup>3</sup>This is sometimes known as “Cheops' law” after the builder of the Great Pyramid.

is also the *intrinsic complexity* of dealing with large and complicated problems. A bank's core systems, for example, may involve tens of millions of lines of code that implement hundreds of different products sold through several different delivery channels, and are just too much for any one person to understand.

Incidental complexity is largely dealt with using technical tools. The most important are high-level languages that hide much of the drudgery of dealing with machine-specific detail and enable the programmer to develop code at an appropriate level of abstraction. They bring their own costs; many vulnerabilities are the result of the properties of the C language, and if we were rerunning history we'd surely use something like Rust instead. There are also formal methods such as static analysis tools, that enable particularly error-prone design and programming tasks to be checked.

Intrinsic complexity requires something subtly different: methodologies that help us divide up a problem into manageable subproblems and restrict the extent to which these subproblems can interact. These in turn are supported by their own sets of tools. There are basically two approaches – top-down and iterative.

### 27.5.1 Top-down design

The classical model of system development is the *waterfall model* formalised by Win Royce in the 1960s for the US Air Force [1628]. The idea is that you start from a concise statement of the system's requirements; elaborate this into a specification; implement and test the system's components; then integrate them together and test them as a system; then roll out the system for live operation. From the 1970s until the mid-2000s, this was how all systems for the US Department of Defense were supposed to be developed, and their lead was followed by many governments worldwide, including not just in defence but in administration and healthcare. When I worked in banking in the 1980s, it was the approved process there too, promoted assiduously by IBM, by governments and by the big accountancy firms.

The idea is that the requirements are written in the user language, the specification is written in technical language, the unit testing checks the units against the specification and the system testing checks whether the requirements are met. At the first two steps in this chain there is feedback on whether we're building the right system (*validation*) and at the next two on whether we're building it right (*verification*). There may be more than four steps: a common elaboration is to have a sequence of *refinement* steps as the requirements are developed into ever more detailed specifications. But that's by the way.

The defining feature of the waterfall model is that development flows inexorably downwards from the first statement of the requirements to the deployment of the system in the field. Although there is feedback from each stage to its predecessor, there is no system-level feedback from (say) system testing to the requirements.

There is a version used in safety-critical systems development called the V model, where the system flows down to implementation, then climbs back up a hill of verification and validation on the other side, where it's tested successively against the implementation, the specification and the requirements. This is a

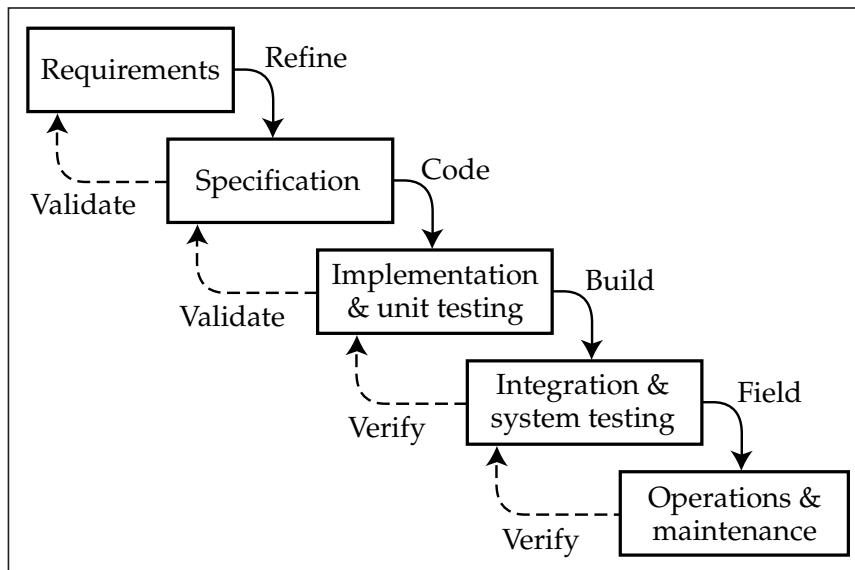


Figure 27.4: – the waterfall model

German government standard, and also used in the aerospace industry worldwide; it's found in the ISO 26262 standard for car software safety. But although it's written from left to right rather than top-down, it's still a one-way process where the requirements drive the system and the acceptance test ensures that the requirements were met, rather than a mechanism for evolving the requirements in the light of experience. It's more a different diagram than a different animal.

The waterfall model had a precursor in a methodology developed by Gerhard Pahl and Wolfgang Beitz in Germany just after World War 2 for the design and construction of mechanical equipment such as machine tools [1490]; apparently one of Pahl's students later recounted that it was originally designed as a means of getting the engineering student started, rather than as an accurate description of what experienced designers actually do. Win Royce also saw his model as a means of starting to get order out of chaos, rather than as the prescriptive system it developed into.

The strengths of the waterfall model are that it compels early clarification of system goals, architecture, and interfaces; it makes the project manager's task easier by providing definite milestones to aim at; it may increase cost transparency by enabling separate charges to be made for each step, and for any late specification changes; and it's compatible with a wide range of tools. Where it can be made to work, it's often the best approach. The critical question is whether the requirements are known in detail in advance of any development or prototyping work. Sometimes this is the case, such as when writing a compiler or (in the security world) designing a cryptographic processor to implement a known transaction set and pass a certain level of evaluation. Sometimes a top-down approach is necessary for external reasons, as with an interplanetary space probe where you'll only get one shot at it.

But very often the detailed requirements aren't known in advance and an iterative approach is necessary. The technology may be changing; the environment could be changing; or a critical part of the project may be the design of a human-computer interface, which will probably involve testing several prototypes. Very often the designer's most important task is to help the customer decide what they want, and although this can sometimes be done by discussion, there will often be a need for some prototyping.

Sometimes a formal project is just too slow. Reginald Jones attributes much of the UK's relative success in electronic warfare in World War 2 to the fact that British scientists hacked stuff together quickly, while the Germans used a rigid top-down development methodology, getting beautifully engineered equipment but always six months too late [990].

But the most common reason for using iterative development is that we're starting from an existing product that we want to improve. Even in the early days of computing, most programmer effort was always expended on maintaining and enhancing existing programs rather than developing new ones; surveys suggest that 70–80% of the total cost of ownership of a successful IT product is incurred after it first goes into service, even when a waterfall methodology was used [2060]. Nowadays, as software becomes a matter of embedded code, apps and cloud services which all become ever more complex, the reality in many firms is that 'the maintenance is the product'.

Even in the late 1990s, when the most complex human artefacts were software packages such as Microsoft Office, the only way to write such a thing was to start off from the existing version and enhance it. That does not make the waterfall model obsolete; on the contrary, it is often used to manage a project to develop a major new feature, or to refactor existing code. However, the overall management of a major product nowadays is likely to be based on iteration.

### 27.5.2 Iterative design: from spiral to agile

There are different flavours of iterative development, ranging from a rapid prototyping exercise to firm up the specification of a new product, through to a managed process for fixing or enhancing an existing system.

In the first case, one approach is the *spiral model* in which development proceeds through a pre-agreed number of iterations in which a prototype is built and tested, with managers being able to evaluate the risk at each stage so they can decide whether to proceed with the next iteration or to cut their losses. Devised by Barry Boehm, it's called the spiral model because the process is often depicted as in Figure 27.5. There are many applications where an initial prototype is the key first step; from a startup aiming to produce a demo to show to investors, through a company building a mockup of a new product to show a focus group, to DARPA seedling projects that aim to establish that some proposed technology isn't completely impossible. Prototype applications for the security engineer range from security usability testbeds to proof-of-concept attack code. The key is to solve the worst problem you're facing, so as to reduce the project risk as much as possible.

The second case we now describe as *agile development*, which may be summed

## 27.5. METHODOLOGY

---

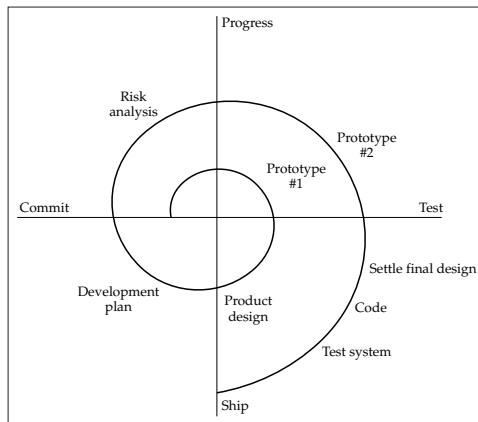


Figure 27.5: – the spiral model

up in the slogan: “Solve your worst problem. Repeat”.

An early advocate for an evolutionary approach was Harlan Mills, who taught that you should build the smallest system that works, try it out on real users, and then add functionality in small increments. This is how the packaged software industry had learned to work by the 1990s: as PCs became more capable, software products became so complex that they could not be economically developed (or redeveloped) from scratch. Indeed, Microsoft tried more than once to rewrite Word, but gave up each time. A landmark early book on evolutionary development was ‘Debugging the Development Process’ by Steve Maguire of Microsoft in 1994 [1209]. In this view of the world, products aren’t the result of a project but of a process that involves continually modifying previous versions. Microsoft contrasted its approach with that of IBM, then still the largest IT company; in the IBM ecosystem, the waterfall approach was dominant. (IBMer for their part decried Microsoft as a bunch of undisciplined hackers who produced buggy, unreliable code; but IBM’s near-death experience after Microsoft stole their main business markets has been ascribed to the rigidity of the IBM approach to development [390].) Professional practice has evolved in the quarter century since then, and evolutionary development is now known as ‘agile’, but it is recognisably the same beast.

A key insight about evolutionary development is that just as each generation of a biological species has to be viable for the species to continue, so each generation of an evolving software product must be viable. The core technology is *regression testing*. At regular intervals – typically once a day – all the teams working on different features of a product check in their code, which gets compiled to a *build* that is then tested automatically against a large set of inputs. The regression test checks whether things that used to work still work, and that old bugs haven’t found their way back. It’s always possible that someone’s code broke the build, so we consider the current ‘generation’ to be the last build that worked. Things are slightly more complex when systems have to work together, as when an app has to talk to a cloud service, or when several electronic components in a vehicle have to work together, or where a single vehicle component has to be customised to work in several different vehicles. You can end up with

a hierarchy of builds and test regimes. But one way or another, we always have viable code that we can ship out for beta testing, or whatever the next stage of our process might be.

The technology of testing was probably the biggest practical improvement in software engineering during the 1990s and early 2000s. Before automated regression tests were widely used, IBM engineers used to reckon that 15% of bug fixes either introduced new bugs or reintroduced old ones [18]. The move to evolutionary development was associated with a number of other changes. For example, IBM had separated the roles of system analyst, programmer and tester; the analyst spoke to the customer and produced a design, which the programmer coded, and then the tester looked for bugs in the code. The incentives weren't quite right, as the programmer could throw lots of buggy code over the fence and hope that someone else would fix it. This was slow and led to bloated code. Microsoft abolished the distinction between analysts, programmers and testers; it had only developers, who spoke to the customer and were also responsible for fixing their own bugs. This held up the bad programmers who wrote lots of bugs, so that more of the code was produced by the more skilful and careful developers. According to Steve Maguire, this is what enabled Microsoft to win the battle to rule the world of 32-bit operating systems; their better development methodology let them take a \$100bn business-software market from IBM [1209].

### 27.5.3 The secure development lifecycle

By the early 2000s, Microsoft had overtaken IBM as the leading tech company, but it was facing ever more criticism for security vulnerabilities in Windows and Office that led to more and more malware. Servers were moving to Linux and individual users were starting to buy Macs. Eventually in January 2002 Bill Gates sent all staff a 'trustworthy computing' memo ordering them to prioritise security over features, and stopping all development while engineers got security training. Their internal training materials became books and papers that helped drive change in the broader ecosystem. I already discussed their threat modelling in section 27.3.5; their first take on secure development appeared in 2002 in Michael Howard and David LeBlanc's 'Writing Secure Code' [927], which sets out the early Microsoft approach to managing the security lifecycle, and which I discussed in the second edition of this book. More appeared over time and their *security development lifecycle* (SDL) appeared in 2008, being adopted widely by Windows developers.

The widely used 2010 'simplified implementation' of SDL is essentially a waterfall process [1308]. It 'aims to reduce the number and severity of vulnerabilities in software' and 'introduces security and privacy throughout all phases of the development process'. The 'pre-SDL' component is security training; it's assumed that all the developers get a basic course, the contents of which will depend on whether they're building operating systems, web services or whatever. There are then five SDL components.

1. Requirements: this involves a risk assessment and the establishment of quality gates or 'bug bars' which will prevent code getting to the next stage if it contains certain types of flaw. The requirements themselves

## 27.5. METHODOLOGY

---

are reviewed regularly; at Microsoft, the reviews are never more than six months apart.

2. Design: this stage requires threat modelling and establishment of the attack surface, to feed into the detailed design of the product.
3. Implementation: here, developers have to use approved tools, avoid or deprecate unsafe functions, and perform static analysis on the code to check this has been done.
4. Verification: this step involves dynamic analysis, fuzz testing, and a review of the attack surface.
5. Release: this is predicated on an incident response plan and a final security review.

As well as providing some basic security training to all developers, there are some further organisational aspects. First, security needs a subject-matter expert (SME) from outside the dev team, and a security or privacy champion within the team itself to check that everything gets done.

Second, there is a maturity model. Starting in 1989, Watts Humphrey developed the *Capability Maturity Model* (CMM) at the Software Engineering Institute at Carnegie-Mellon University (CMU), based on the idea that competence is a function of teams rather than just individual developers. There's more to a band than just throwing together half-a-dozen competent musicians, and the same holds for software. Developers start off with different coding styles, different conventions for commenting and formatting code, different ways of managing APIs, and even different workflow rhythms. The CMU research showed that newly-formed teams tended to underestimate the amount of work in a project, and also had a high variance in the amount of time they took; the teams that worked best together were much better able to predict how long they'd take, in terms of the mean development time, but reduced the variance as well [1937]. This requires the self-discipline to sacrifice some efficiency in resource allocation in order to provide continuity for individual engineers and to maintain the team's collective expertise. Microsoft adapted this and defines four levels of security maturity for developer teams.

### 27.5.4 Gated development

It's telling that the biggest firm pushing evolutionary development reverted to a waterfall approach for security. Many of the security engineering approaches of the time were tied up with waterfall assumptions, and automated testing on its own is less useful for the security engineer for a number of reasons. Security properties are both emergent and diverse, we security engineers are fewer in number, and there hasn't been as much investment in tools. Specific attack types often need specific remedies, and many security flaws cross a system's levels of abstraction, such as when specification errors interact with user interface features – the sort of problem for which it's difficult to devise automated tests. But although regression testing is not sufficient, it is necessary, as it finds functionality that's been affected by a change. It's particularly important when

development sprints add lots of features that can interact with each other. For this reason, security patches to Windows are an example of *gated development*: at regular intervals, a pre-release version of the product is pushed through a whole series of additional tests and reviews and prepared for release. This is fairly common across systems with safety or security requirements. The preparation may involve testing with a wide variety of peripherals and applications in the case of Windows, or recertification in the case of software for a regulated product.

An issue many neglect is that security requirements evolve, and also have to be maintained and upgraded. They can be driven by changing environments, evolving threats, new dependencies on platforms old and new, and a bundle of other things. Some changes are implicit; for example, when you upgrade your static analysis tools you may find hundreds of ‘new’ bugs in your existing codebase, which you have to triage. Once more Microsoft was a pioneer here. When a vulnerability was found in Windows, it’s not enough to just patch it; whoever wrote it might have written a dozen similar ones that are now scattered throughout the codebase, and once you publish a patch, the bad guys study it and understand it. So rather than just fixing a single bug, you update your toolchain so you find and eliminate all similar bugs across your products. In order to manage the costs, both for Microsoft and its customers, the company started bundling patches together into a monthly update, the now famous ‘patch Tuesday’, in 2003. From then until 2015, all customers – from enterprises to the users of home PCs and tablets – had their software updated on the second Tuesday every month. And such patching creates further dependencies. Modern quality tools can help you check that no code has a CVE open, so all your customers should have to patch too, if they live by such tools. But many don’t: as many as 70% of apps on both phones and desktops have vulnerabilities in the open-source libraries they use, and which could usually be fixed by a simple update [1695]. Since 2015, Windows home users receive continuous updates<sup>4</sup>.

Much the same considerations apply to safety-critical systems, which are similar in many respects to secure systems. Safety, like security, is an emergent property of whole systems, and it doesn’t compose. Safety used to depend, in most applications, on extensive pre-market testing. But it’s hard for a connected device to have safety without security, and now that devices such as cars are connected to the Internet, they are acquiring patch cycles too. Yet ensuring that the latest version of a safety-critical system satisfies the safety case may require extensive and expensive testing. For example, a car may contain dozens of *electronic control units* (ECUs) from different component suppliers, and in addition to testing the individual ECUs you have to test how they work together. Firms in the car industry are mutually suspicious and won’t share source code with each other, even under NDA, so testing can be complex. The main test rig may be a ‘lab car’ containing all the electronics from a particular model of car, plus extra test systems that let you simulate various maneuvers and even accidents. These cost real money, and you also need to keep real vehicles for

---

<sup>4</sup>This also breaks things: we were once about to demonstrate an experiment using a body motion-capture suit to a TV crew when the Windows laptop we used to drive it updated itself, and suddenly the capture software wouldn’t work any more. There followed frantic phone calls to the software developer in the Netherlands and thankfully we got their update a few hours later, just in time for the show.

road testing. The cost of maintaining fleets of lab cars and real test cars is one of the reasons car companies dragged their heels when the EU decided to require them to patch car software for ten years after the last vehicle left the showroom.

This is one respect in which Tesla has a significant advantage; as a tech company with software at the core of its business, Tesla can test and ship changes in weeks which take the legacy car firms years, as they leave most of the software development to the component suppliers [404]. Traditionally, automotive software contracts involved ten years' support; now you need to support a product for three years' development, seven years in the showroom and a further ten after that. I'll discuss the sustainability aspects of this in the next chapter. Meanwhile, Tesla is forcing the legacy industry to raise its game, with VW announcing they've spent \$8bn to create a proper software division, just as their Tesla competitor project runs late [1686].

### 27.5.5 Software as a Service

Since the early 2010s, more and more software has been hosted on central servers, accessed by thin clients and paid for on a subscription basis, rather than being sold and distributed to users. The typical customer has many costs for running software beyond the license fee, including not just the cost of servers and operators but of deploying it, upgrading it regularly and managing it. If the vendor can take over these tasks from all their customers, many duplicated costs are removed, and they can manage things better because of their specialised knowledge. Software can be instrumented so that developers can monitor all aspects of its performance on a dashboard.

The key technical innovations behind *Software as a Service* (SaaS) are *continuous integration* and *continuous deployment*. Rather than having thousands of customers managing dozens of different versions of the software, the vendor can migrate a few customers to a new version to test it, and then migrate the rest. Upgrades become much more controllable, as they can be tested in a dry run against a snapshot of the real customer data, called a *staging environment*. Some companies now deploy several times a day, as their experience is that frequent small changes can be safer and have less risk of breaking something than a larger deployment, such as Microsoft's Patch Tuesday.

Deployment itself is tentative. A SaaS company will typically run its software on a number of service instances running on VMs behind a load balancer, which provides a point of indirection for managing running services. The separate instances also provide separate *failure domains* to improve robustness. To do a *rolling deployment* we configure a load balancer to send say 1% of the traffic to an instance with the new version, often called the 'canary' after the caged bird used by miners to detect carbon monoxide leaks. If the canary survives, deployment can be rolled forward progressively to new service instances. If the logging system detects any problems, developers are alerted. Some care needs to be taken that things don't go wrong if users flap between old and new versions of a design between transactions. If you make a change that breaks backwards compatibility, you typically build an intermediate stage that will work with both old and new systems (we were doing this in the world of bank mainframes back in the 1980s anyway).

The ability to manage risks through phased release and rolling deployment changes the economics of testing. The fact that you can fix bugs extremely quickly mean that you can achieve a target quality level with much less testing. You can also see everything the users do, so for the first time you can really understand how usability fails from the point of view of security, safety – and revenue. Of course it’s revenue that usually drives the exploitation of this. Analytics collectors write all behavioural events to a log, which is fed into a data pipeline for metrics, analytics and queries. This in turn supports experiment frameworks that can do extensive A/B testing of possible features. Ad-driven services can optimise by engagement metrics such as active users, time per user session and use of specific features. Controlled experiments are used to improve security too; for example, Google has tuned its browser warnings by measuring how millions of users react to different warnings of expired certificates. Such improvements are usually fairly small by themselves, so you really need controlled experiments to measure them; but when you do lots of them, they add up. The investment in building such frameworks into the phased deployment mechanisms gives an increasing return to scale; the more users you have, the faster you can achieve statistical significance. So large firms can optimise their products more quickly than their smaller competitors; SaaS, like a lot of other digital technology, not only cuts costs in the short term, but increases lock-in in the long term. Each time you access a service from a large SaaS firm, you may be an unwitting participant in tens or even hundreds of experiments. There are lots of fiddly details about running multiple concurrent experiments while also deploying system enhancements.

Things can get more complex still when you have services put together from multiple microservices. This brings us to the world of *infrastructure as code*, also known as cloud native development or DevOps, where everything is developed in containers, VMs etc, so all the infrastructure is based on code and can be replicated quickly. You can also use containers to simplify things, packaging as many security dependencies with the code as possible. New code can be deployed to a test infrastructure rapidly and tested realistically. You could if you wanted manage rolling deployment manually, but this is not scalable and prone to error. The solution is to write *deployment code*, as part of the application development process, that uses the cloud platform APIs to allow applications to deploy themselves and the associated infrastructure, and to hook into the monitoring mechanisms. In the last few years, some toolkits have become available that allow engineers to do this in a more declarative fashion.

The best guide to this I know is Google’s 2013 book ‘*Site Reliability Engineering*’; SRE is their term for DevOps [236]. Google led the industry in the art of building large dependable systems out of large fleets of low-cost PCs, building the necessary engineering for load balancing, replication, sharding and redundancy. As they operated at a larger scale than anybody else through the 2000s and early 2010s, they had to automate more tasks and became good at it. The goals of SRE are availability, latency, performance, efficiency, change management, monitoring, emergency response, and capacity planning. The core strategy is to apply software engineering techniques to automate system administration tasks so as to balance rapid innovation with availability.

As we already noted, there’s no point striving for 99.9999% availability if

ISPs only let users get to your servers 99% or 99.9% of the time. If you set a realistic error budget, say 0.1% or 0.01% unavailability, you can use that to achieve a number of things. First, most outages are due to live system changes, so you monitor latency, traffic, errors and saturation well and roll back quickly whenever anything goes wrong. You use the rest of the error budget to support your experimental framework, and doing controlled outages to flush dependencies. (This was pioneered by Netflix whose ‘chaos monkey’ would occasionally take down routers, servers, load balancers and other components, to check that the resilience mechanisms worked as intended; such ‘fire drills’ are now an industry standard and involve taking down whole data centres.)

In section 12.2.6.2, we mentioned *technical debt*. This concept, due to Ward Cunningham, encapsulates the observation that development shortcuts are like debt. Whenever we skimp on documentation, fix a problem with a quick-and-dirty kludge, don’t test a fix thoroughly, fail to build in security controls, or fail to work through the consequences of errors, we’re storing up problems that may have to be repaid with interest in the future [41]. Technical debt may make sense for a startup, or a system nearing the end of its life, but it’s more often a product of poor management or poorly-aligned incentives. Over time, systems can fall so deeply into debt that they become too hard to maintain or to use; they have to be refactored or replaced. For a bank to have to replace its core banking systems is hugely expensive and disruptive. So managing technical debt is really important; this is one of the changes in system management thinking since the second edition of this book. One important aspect of the philosophy of DevOps is to run debt-free.

### 27.5.6 From DevOps to DevSecOps

As I write, in 2020, the cutting edge is applying agile ideas and methodology not just to development and operations, but to security too. In theory this can mean a strategy of ‘everything as code’; in practice it means not just maintaining an existing security rating (and safety case if relevant) but responding to new threats, environmental changes, and surprising vulnerabilities. Bringing the two together involves real work, and sometimes things need to be reinvented. I mentioned for example in section 12.2.2 that DevOps undermines the separation between development and production on which banks have relied for years; where separation of duties is necessary, we have to reimagine it.

We see several different approaches in the companies with which we work. In what follows I will give two examples, which we might roughly call the Microsoft world and the Google world. There are of course many others.

#### 27.5.6.1 The Azure ecosystem

Most of the world’s largest commercial firms from banks and insurers through retail to shipping and mining have built their enterprise systems on Windows over the past 25 years and are now migrating them to Azure, often using systems integration and facilities management firms to do the actual work. The typical client has a mixture of on-premises and cloud systems with new developments mostly migrating from the former to the latter. Here policy is largely set by

the Big Four auditors who, in addition to their standard set of internal control features, follow Microsoft in requiring a secure development lifecycle. The several dozen tools used to do threat modelling, static analysis, dynamic analysis, fuzz testing, app and network monitoring, security orchestration and incident response impose a significant overhead with dozens of people copying data from one tool to another. The DevSecOps task here is to progressively integrate the tools by automating these administrative tasks.

To support this ecosystem, Microsoft has extended its SDL with further steps: defining metrics and compliance reporting; threat modelling; cryptography standards; managing the security risks of third-party components; penetration testing; and a standardised incident response. The firm now claims that 10% of its engineering investment is in cybersecurity. The capable system integration and facilities management firms have worked out ways of building these steps into their workflows; much of the actual work involves integrating the third-party security products that they or their customers have bought. Appropriate automation is vital for the security team to continue raising their game, extending their scope and increasing effectiveness; without it, they fall further and further behind, and burn out [1846].

The organising principles for DevSecOps in such a company will be to ‘shift left’ which can cover a number of things: the unifying theme is moving security, like software and infrastructure, into the codebase. One strategy is to cause things to ‘fail fast’ including engaging security experts early enough in the development process to avoid delays later: doing pre-commit static analysis of each developer’s code to minimise failed builds; buying or building specialist tools to detect errors such as incorrect authentication, mistakes in using crypto functions, and injection opportunities; both automated and manual security testing of new versions; and automated testing of configuration and deployment including scanning of the staging network and checks on credentials, encryption keys and so on. And while, back in 2010, Microsoft considered operational security to be separate from software security, a modern Azure shop will close the loop by following up deployment with continuous monitoring, manual penetration tests and finally bug bounties for third parties who spot something wrong. We will discuss these in more detail later.

### 27.5.6.2 The Google ecosystem

A second view comes from engineers working on infrastructure, and the best reference I know is a 2020 book by six Google engineers, ‘*Building Secure and Reliable Systems*’ [23]. The DevSecOps strategy is somewhat similar at Amazon, but optimised for their product offerings; it is described by their CTO Werner Vogels at [1966]. However the Google experience is described in much more detail. This section draws on their book, and on colleagues who have worked recently at the major service firms.

When building infrastructure systems on which hundreds of millions of people will rely, it is critical to automate support functions quickly, and to have really robust processes for threat identification, incident response, damage limitation and service recovery. So while a facilities-management firm might work at integrating support functions to save money and reduce errors, the emphasis

at major service firms is reliability. I already mentioned the Google approach to site reliability engineering: set a realistic target, of say 99.9% availability, and then use the residual error budget of 0.1% downtime by apportioning it between failure recovery, upgrades and experiments.

This in turn drives further principles such as design for recoverability, design for understandability, and a desire to stop humans touching production systems wherever possible. It's not enough to have automation for the incremental deployment of new binaries; you also want to stop sysadmins having to type complicated command lines into routers to configure networks; this is where most of the network outages come from, as we noted in section 21.2.1. You manage such risks by building suitable tool proxies. This can involve quite a lot of work to align the update of binary and config files and work out how to allocate support and recovery effort between SRE and security engineering teams. Further complexity arises with secure testing. How do you build test infrastructures to exercise least privilege? How do you test systems that contain large amounts of personal information? How do you test the break-glass mechanisms that give SRE teams emergency human access to live systems? Most of these are questions we already had to deal with in the mainframe world of the 1980s, but they arose only occasionally and were dealt with by human ingenuity and by trusting some key staff. Scaling everything up from thousands of users to billions means that a lot more has to be automated.

There are still tensions. In site reliability engineering, alarms should be as simple, predictable and reliable as possible; but in security, some randomisation is often a good idea.

At the application level, systems are increasingly compartmentalised into microservice components with defensible security boundaries and tamper-resistant security contexts, so that if Alice compromises a shopping system's catalogue, she still can't spend money as Bob as the payment service is separate. Each component will typically be implemented as a number of parallel copies or shards, giving still smaller failure domains. Such domains enable you to limit the blast radius of any compromise; ideally, you want to be able to deal with an intrusion without taking your whole system offline. Compartmentalised systems can be engineered for resilience too but this is not straightforward. When a failure domain fails, when do you just spin up a new one, and when do you do something different? What are the dependencies? Which components should fail open, and which should fail secure? What sort of degraded performance is acceptable under congestion, or under attack? What's the role of load shedding and throttling? And what sort of pain can you rationally inflict on users, and on business models? Do you ditch some of the ads, require extra CAPTCHAs for logons, or both? And how do you test and validate all these resilience mechanisms?

Large firms invest a lot of engineering time in building application frameworks for such services. There are also standard frameworks for web pages, which should not only prevent SQL injection and cross-site scripting attacks in the first place, but also provide support for dozens of different languages. Having a single front end to terminate all http(s) and TLS traffic means that if you have to update your certificate management mechanisms or ciphersuites you only need to do it once, not in all your different services. A single front end can also provide a single location for load balancing and DDoS protection, as well

as for many other functions such as supporting dozens of different languages.

Using type encapsulation to enforce properties of URLs, SQL and so on can reduce the amount of code you need to verify. If you have secure-by-construction APIs that are also understandable, that's best. Google has a crypto API called Tink that forces more correct use. It requires use of a key management service, whether in the Google cloud, AWS or the Android keystore. This fits into an overall framework for managing crypto termination, code provenance, integrity verification and workload isolation, called BeyondProd [998].

### 27.5.6.3 Creating a learning system

Whether you follow the Microsoft approach, the Google approach or your own, to tune such a process you need metrics, and suitable candidates include the numbers of security tickets opened to dev teams, the number of security-failed builds, and the time it takes for a new application to achieve compliance under the relevant regulation (whether SOX, GDPR or HIPAA). As Dev, Sec and Ops converge, the metrics and management processes converge with the network defence mechanisms discussed in section 21.4, from network monitoring to security incident and event management. But all this needs to be managed intelligently. A well-run firm can make the security process more visible to all the dev / ops staff via the sprints that you do to work up a privacy impact assessment, improve access controls, extend logging or whatever. A badly-run firm will manage to the metrics, which will create tensions: their security staff can end up with conflicting goals of keeping the bad guys out, and also of 'feeding the beast' by hitting all the metrics used to justify the team's own existence [1846]. It's important to understand where conflicts naturally arise as a function of the organisation's management structure, and somehow keep them constructive.

One of the big drivers in either case, though, will be the vulnerability lifecycle. The processes whereby bugs become exploits and then attacks, and these attacks are noticed leading to vulnerability reports, interim defences using devices such as firewalls, then definitive patches that are rolled out not just to direct users but along complex supply chains, is ever more central to security management.

### 27.5.7 The vulnerability cycle

Back in the 1970s and 1980s, people sometimes described the evolutionary procedure of finding security bugs in systems and then fixing them dismissively as *penetrate-and-patch*. It was hoped that some combination of an architecture that limited the attack surface and the application of formal methods would enable us to escape. As we've seen, that didn't really work, except in a few edge cases such as cryptographic equipment. By the early 2000s, we had come to the conclusion that we just had to manage the patch cycle better, and the modern approach of security breach disclosure laws, CERTs and responsible disclosure bedded down during this period.

The vulnerability cycle consists of the process whereby someone, the *researcher*, discovers a vulnerability in a system that is maintained by a *vendor*.

The researcher may be a *customer*, an academic, a contractor for a national intelligence agency or even a criminal. They may sell it in a market. The idea of vulnerability markets was first suggested by Jean Camp and Catherine Wolfram in 2000 [371]; firms were set up to buy vulnerabilities, and over time several markets emerged. Most of the big software and service firms now offer bug bounties, which can range from thousands to hundreds of thousands of dollars; at the other extreme are operators who buy up exploits for sale to *exploiters* such as cyber-arms manufacturers (who sell to military and intelligence agencies) and forensic firms (who sell to law enforcement). Such operators now offer millions of dollars for persistent remote exploits of Android and iOS.

The researcher may also disclose the bug to the vendor directly – nowadays many vendors have a *bug bounty program* that pays rewards for disclosed vulnerabilities that attempt to match market prices, at least in order of magnitude. As market prices for zero-day exploits against popular platforms have headed into six and even seven figures, so have bug bounties. Apple, for example, offers \$1M for anyone who can hack the iOS kernel without requiring any clicks by the user. In 2019, it emerged that at least six hackers have now earned over \$1M through the bug bounty platform HackerOne alone [2030]. A downside of large bug bounties is that while bugs used to occur naturally, we now see them being introduced deliberately, for example by contributors to open-source projects whose code ends up in significant platforms. Such *supply-chain attacks* used to be the preserve of nation states; now they're opening up [890].

If an exploit is used in the wild before the vendor issues a patch, it is called a *zero day*, and is typically used for targeted attacks. If it's used enough, then eventually someone will notice; the attack gets reported, and then vendor issues a patch, which may then be reverse engineered so that many other actors now have exploit code. Customers who fail to patch their systems are now vulnerable to multiple exploits that can be deployed at scale by crime gangs.

Getting the patching cycle right is a problem in the economics of information security as much as anything else, because the interests of the various stakeholders can diverge quite radically.

1. The vendor would prefer that bugs weren't found at all, to spare the expense of patching. They'll patch if they have to but want to minimise the cost, which may include a lot of testing if their code appears in lots of product versions. Indeed, if their code is used in customer devices that now need patching (like cars) they may have to pay an indemnity to cover their customer's costs; so in such industries there's an even more acute incentive for foot-dragging and denial.
2. The average customer might prefer that bugs weren't found, to avoid the hassle of patching. Lazy customers may fail to patch, and get infected as a result. (If all the infected machines do is send a bit of spam, their owners may not notice or care.)
3. The typical security researcher wants some reward for their discoveries, whether fame, cash or getting a fix for a system they rely on.
4. The intelligence agencies want to learn of vulnerabilities quickly, so they can be used in zero-day exploits before a patch is shipped.

## 27.5. METHODOLOGY

---

5. The security software firms benefit from unpatched vulnerabilities as their firewalls and AV software can look for their indicators of compromise to block attacks that exploit them.
6. Large companies don't like patches, and neither do government departments, as the process of testing a new patch against the enterprise's critical systems and rolling it out is expensive. The better ones have built automation to deal with regular events like Microsoft's Patch Tuesday, but updating or risk-assessing the millions of IoT devices in their offices and factories will be a headache for years to come. Most firms just don't have a good enough asset inventory system to cope.

During the 1990s, the debate was driven by people who were frustrated at software vendors for leaving products unpatched for months or even years. The bugtraq mailing list was set up to provide a way for people to disclose bugs anonymously; but this meant that a product might be completely vulnerable for a month or two until a patch was written, tested and shipped, and until customer firms had tested it and installed it. This led to a debate on 'responsible disclosure' with various proposals about how long a breathing space the researcher should give the vendor [1572].

The consensus that emerged was that researchers should disclose vulnerabilities to a computer emergency response team (CERT)<sup>5</sup> and the global network of CERTs would inform the vendor, with a delay for a patch to be issued before the vulnerability was published. The threat of eventual disclosure got vendors off their butts; the delay gave them enough time to test a fix properly before releasing it; researchers got credit to put on their CVs; customers got bug fixes at the same time as bug reports; and the big companies organised regular updates for which their corporate customers can plan. Oh, and the agencies had a hot line into their local CERT, so they learned of naturally occurring exploits in advance and could exploit them. This was part of the deal described in section 26.2.7.3 that ended Crypto War 1 back in 2000.

### 27.5.7.1 The CVE system

An industrial aspect is the *Common Vulnerabilities and Exposures* (CVE) system, launched in 1999, which assigns numbers to reported vulnerabilities in publicly released software packages. This is maintained by Mitre, but it delegates the assignment of CVEs to large vendors. CVE IDs are commonly included in security advisories, enabling you to search for details of the reporting date, affected products, available remedies and other relevant information. There is a Common Vulnerability Scoring System (CVSS) which provides a numerical representation of the severity of a vulnerability. The method for calculating this has become steadily more complex over time and now depends on whether the attack requires local access, its complexity, the effort required, its effects, the availability of exploit code and of patches, the number of targets and the potential for damage.

---

<sup>5</sup>The EU is renaming these CSIRTs – computer security incident response teams.

NIST's *National Vulnerability Database* (NVD), described as a "comprehensive cybersecurity vulnerability database that integrates all publicly available U.S. Government vulnerability resources and provides references to industry resources" is based on the CVE List. These resources are critical for automating the tracking of vulnerabilities and updates. There are now so many thousands of vulnerabilities reported, and so many hundreds of patches shipped, that automation is essential.

As the system was bedding down, it became a subject of study by security economists. Traditionalists argued that since bugs are many and uncorrelated, and since most exploits use vulnerabilities reverse-engineered from existing patches, there should be minimal disclosure. Pragmatists argued that, from both theoretical and empirical perspectives, the threat of disclosure was needed to get vendors to patch. I discussed this argument in section 8.6.2. Since then we have seen the introduction of automatic upgrades for mass-market users, the establishment of firms that make markets in vulnerabilities, and empirical research on the extent to which bugs are correlated. Modulo some tuning, the current computer industry way of doing things has been stable for over a decade.

### 27.5.7.2 Coordinated disclosure

Yet some industries are lagging well behind. In section 4.3.1 I described how Volkswagen sued academics at Birmingham and Nijmegen universities after they discovered, and responsibly disclosed, vulnerabilities in Volkswagen's remote key entry system that were already being exploited in car-theft tools that were available online. This was a mistake, as it drew attention to the vulnerability, and Volkswagen duly lost in court. Companies like Microsoft and Google have had twenty years to learn that running bug bounty programs and monthly patching works better than threatening to sue people, but a lot of firms in legacy industries still haven't worked this out even though their products contain more and more software.

One of the problems in the Volkswagen case was that the researchers initially disclosed the vulnerability to the supplier of its key entry system, which in turn told Volkswagen only at the last minute. As a result of supply chain problems like this, responsible disclosure has given way to *coordinated disclosure*. Few firms build all their own tools any more, and even a child's toy may have multiple software dependencies. If it does speech and gesture recognition, it probably contains an Arm chip running some flavour of Linux or FreeBSD, communicates with a cloud service running another flavour of Linux, and can be controlled by an app that may run on Android or iOS. The safety of the toy will depend on secure communications; for example, it was discovered in February 2019 that the communications between Enox's 'Safe-KID-One' toy watch and its backend server were unencrypted, so that hackers could in theory track and call kids. The response was an immediate EU-wide safety recall [654]. Getting this sort of thing wrong can be sudden death for your product, and your company.

Now what happens when someone discovers an exploitable bug in a platform used in dozens of embedded products? This can be traumatic, as with the Shellshock bug in Linux and the Heartbleed bug in OpenSSL (which also affected Linux). If Linux gets an emergency patch, coordinating the disclosure is a

nightmare: the Linux maintainers may be able to work in private with the main Linux distributions, and with derivatives like Android whose developers keep in close contact with them. But there are the thousands of products that incorporate Linux, from alarm clocks to TVs and from kids' toys to land mines. You may suddenly find that the CCTV cameras in your building security system have all become hackable, and the vendor can't fix them quickly or at all. Coordinating disclosure on platforms is one of the seriously hard problems. There is no silver bullet but there are still many things you can do, ranging from documenting your upstream and downstream dependencies, through aggressive testing of software you depend on so you get to exercise and understand the bug reporting mechanisms, to becoming part of its developer community.

Dealing with such shocks is just one aspect of a process that in the late 2010s became a speciality of its own, namely security incident and event management.

### 27.5.7.3 Security incident and event management

You need an incident response plan for what you'll do when you learn of a vulnerability or an attack. In the old days, vendors could take months to respond with a new version of the product, and would often do nothing at all but issue a warning (or even a denial). Nowadays, breach-notification laws in both the USA and Europe oblige firms to disclose attacks where individuals' privacy could have been compromised, and people expect that problems will be fixed quickly. Your plan needs four components: monitoring, repair, distribution and reassurance.

First, make sure you learn of vulnerabilities as soon as you can – and preferably no later than the bad guys (or the press) do. This means building a threat intelligence team. In some applications you can just acquire threat intelligence data from specialist firms, while if you're an IoT vendor it may be prudent to operate your own honeypots so you get immediate warning of people attacking your products. Listening to customers is important: you need an efficient way for them to report bugs. It may be an idea to provide some incentive, such as points towards their next upgrade, lottery tickets or even cash. You absolutely need to engage with the larger technical ecosystem of bug bounties, vulnerability markets, CERTs and CVEs described in section 27.5.7.

Second, you need to be able to repair the problem. Twenty years ago, that meant having one member of each product team 'on call' with a pager in case something needed fixing at three in the morning. Nowadays it means preparing an orchestrated response to anything from a vulnerability report to a major breach. This will extend from the intrusion-detection and network monitoring functions we discussed in section 21.4.2.3 and the threat intelligence team through to identifying the dev teams responsible and notifying both your suppliers upstream and your customers downstream. Responder teams may also need alternative means of communication. Did you ever stop to think whether you need satellite phones?

Third, you need to be able to deploy the patch rapidly: if all the software runs on your own servers, then it may be easy, but if it involves patching code in millions of consumer devices then advance planning is needed. It may seem easy to get your customers to visit your website once a day and check for upgrades,

but if their own systems depend on your devices and they need to test any dependencies, there's a tension [195]: pioneers who apply patches quickly can discover problems that break their systems, while people who take time to test will be more vulnerable to attack. The longer the supply chains get, the harder the conflicts of interest are to manage. Operations matter hugely: an emergency patch process that isn't tested may do more harm than good, and experience teaches that in an emergency you just run your normal patch process as fast as possible [23].

Finally, you need to educate your CEO and main board directors in advance about the need to deal quickly and honestly with a security breach in order to keep confidence and limit damage, by giving them compelling examples of firms that did well and others that did badly. You need to have a mechanism to get through to your CEO and brief them immediately so they can show the thing's under control and reassure your key customers. So you need to know the mobile and home phone numbers of everyone who might be needed urgently. And you need a plan to deal with the press. The last thing you need is for dozens of journalists to phone up and be stonewalled by your PR person or even your switchboard operator as you struggle madly to fix the bug. Have a set of press releases ready for incidents of varying severity, so that your CEO only has to pick the right one and fill in the details. This can then ship as soon as the first (or perhaps the second) journalist calls.

Remind your CEO that both the USA and Europe have security-breach disclosure laws, so if your systems are hacked and millions of customer card numbers compromised, you have to notify all current and former customers, which costs real money. You can expect to be sued. If you have 10 million customers' personal data compromised, that might mean 10 million letters at \$5 each and 3 million reissued credit cards at \$10 each, even if you don't get claims from banks for actual fraud losses on those accounts. (That may well happen; you might expect that of 3 million accounts, a few tens of thousands would suffer some fraud in each year anyway, and the banks will sue you for all of it.) The financial loss from a significant breach can easily hit nine figures. If it happens to you more than once, you can expect to lose customers: customer churn might only be 2% after one notified breach, but 30% after two and even more after three [2037]. Since some CEOs have been fired after large breaches, information security has become a CEO issue.

### 27.5.8 Organizational mismanagement of risk

Organizational issues are not just a contributory factor in system failure, as with the loss of organizational memory and the lack of mechanisms for monitoring changing environments. They can often be a primary cause. There's a large literature on how people behave in organisations, which I touched on in section 8.6.7, and I've given a number of further examples in various chapters. However, the importance of organisational factors increases as projects get bigger. Bezos' law says you can't run a dev project with more people than can be fed from two pizzas. A team of eight people is just about manageable, but you can't go six times as fast by having six such teams in parallel. If a project involves multiple teams the members can't talk to each other at random, or you

get chaos; and they can't route all their communications through the lowest common manager as there isn't the bandwidth. As you scale up, the coordination will start to involve a proliferation of middle managers, staff departments and committees. The communications complexity of a clean military chain of command, for  $N$  people with no lateral interaction, is  $\log N$ ; where everybody has to consult everybody else, it's  $N^2$ ; and where any subset can form a committee to think about the problem, it can head towards  $2^N$ . Business school people have written extensively about this, and their methodology is generally based on case studies.

Many large development projects have crashed and burned. The problems appear to be much the same whether the disaster is a matter of safety, of security or of the software simply never working at all; so security people can learn a lot from studying project failures documented in the general engineering literature.

A classic study of large software project disasters was written by Bill Curtis, Herb Krasner, and Neil Iscoe [504]. They found that failure to understand the requirements was mostly to blame: a thin spread of application domain knowledge typically led to fluctuating and conflicting requirements which in turn caused a breakdown in communication. The example I give in my undergraduate lectures is the meltdown of a new dispatch system for the London Ambulance Service where a combination of an overly ambitious project, an inadequate specification and no real testing led to the city being without ambulance cover for a day. There are all too many such examples; I use the London Ambulance Service case because the subsequent inquiry documented the causes rather well [1805]. I also happened to be in London that day, so I remember it. If you haven't ever read the inquiry report, I recommend you do so. (In fact I strongly recommend that you read lots of case studies of project failure.)

The millennium bug gives another useful data point. If one accepts that many large commercial and government systems needed extensive repair work to change two-digit dates into four-digit ones in preparation for the year 2000, and the conventional experience that a significant proportion of large development projects are late or never delivered at all, many people naturally assumed that a significant number of systems would fail at the end of 1999, and predicted widespread chaos. But this didn't happen. Certainly, the risks to the systems used by small and medium-sized firms were overstated; we did a thorough check of all our systems at the university, and found nothing much that couldn't be fixed fairly easily [69]. Nevertheless, the systems of some large firms whose operations are critical to the economy, such as banks and utilities, did need substantial repairs. Yet there were no reports of high-consequence failures. This appears to support Curtis, Krasner, and Iscoe's thesis. The requirement for Y2K bug fixes was known completely: "I want this system to keep on working, just as it is now, through into 2000 and beyond".

This is one of the reasons I chose the quote from Rick Smith to head this chapter: "My own experience is that developers with a clean, expressive set of specific security requirements can build a very tight machine. They don't have to be security gurus, but they have to understand what they're trying to build and how it should work."

Organisations have difficulty dealing with uncertainty, as it gets in the way of

## 27.5. METHODOLOGY

---

setting objectives and planning to meet them. So capable teams tackle the hard problem first, to reduce uncertainty; that was DARPA’s mission, and the core of the spiral model. There’s a significant business-school literature on how to manage uncertainty in projects [1179]. But it’s easy to get this wrong, even in a fairly well-defined project. Faced with a hard problem, it is common for people to furiously attack a related but easier one; we’ve seen a number of examples, such as in section 26.2.8.

Risk management can be even worse in security where the problem is open-ended. We really have no idea where the next shitstorm will come from. In the late 1990s, we thought we’d got secure smartcards; then along came differential power analysis. In the mid-2010s we thought we had secure enough CPUs for competitor firms to run their workloads on the same machines in Amazon data centres; then along came Spectre. We also used to think that Apple products couldn’t get malware and that face recognition would never be good enough to be a real privacy threat. Even though Moore’s law is slowing down, there will be more surprises.

Middle managers prefer approaches that they can implement by box-ticking their way down a checklist, but to deal with uncertainties and open-ended risks, you need a process of open learning, with people paying attention to the alerts, or the frauds, or the safety incidents, or the customer complaints – whatever you can learn from. But checklists demand less management attention and effort, and the quality bureaucracy loves them. I noted in section 9.6.6 that certified processes had a strong tendency to displace critical thought; instead of constantly reviewing a system’s protection requirements, designers just reach for their checklists. The result is often perverse. By not tackling the hard problem first, you hide the uncertainty and it’s worse later<sup>6</sup>. Also, people rapidly learn how to game checklists. There is the eternal tension between us security experts telling firms to pay smart people to anticipate what might go wrong, and boards telling managers to deliver product faster using fewer and cheaper engineers.

When the threat model is politically sensitive, things get more complicated. The classic question is whether attacks come from insiders or outsiders. Insiders are often the biggest security risk, whether because some of them are malicious or because most of them are careless. But you can’t just train all your staff to be unhelpful to each other and to customers, unless perhaps you are a government department or other monopoly. You have to find the sweet spot for control, and that often means working out how to embed it in the culture. For example, bank managers know that dual-control safe locks reduce the risk of their families being taken hostage, and requiring two signatures on large transactions means extra shoulders to take the burden when something goes wrong.

Getting the risk ecosystem right in an organisation can take both subtlety and persistence. The cultural embedding of controls and other protective measures is hard work; if you come into contact with multiple firms then it’s interesting to observe how they manage their rules around everything from code audits (which the tech majors insist on) to tailgating (which semiconductor firms are at pains to prevent) and whether people are expected to keep one hand on a

---

<sup>6</sup>I will discuss ISO 27001 in the next chapter. The executive summary for now is that almost every firm hit by a big data breach had ISO 27001 certification, but it failed because their auditors said something was OK that wasn’t.

banister as they walk up and down the stairs (a favourite of energy companies). Where do these risk cultures come from, how are they promoted, and why do they cluster by sector? Their transactional internal control structures may be heavily influenced by their auditors, as we discussed in section 12.2.6.3, but the broader security culture varies a lot – and matters.

A further factor is that good CISOs are almost as rare as hens' teeth. There are some stars at the top tech and fintech firms, but being a CISO can be a thankless job. Good engineers often don't want it, or don't have the people skills to cope, while ambitious managers tend to avoid the job. In many organisations, promotions are a matter of seniority and contacts; so if you want to be the CEO you'll have to spend 20 years climbing up the hierarchy without offending too many people on the way. Being CISO will mean saying no to people all the time, and a generalist with no tech background can't hack it anyway. The job also brings a lot of stress, and the risk of burnout; a CISO's average tenure is about two years [430]. In any case, embedding an appropriate culture around risk and security is for the CEO and the board. If they don't think it's important, the CISO has no chance. But breaches have now led to enough CEOs being fired, or losing millions on their stock, that other members of that tribe are starting to pay attention.

One way the risk ecosystem can be skewed is that if a company manages to arrange things so that some of the risks of the systems it operates get dumped on third parties. This creates a moral hazard by removing the incentives to take care. We discussed this in section 12.5.2 in the context of banks trying to shift fraud liability in payment systems to cardholders, merchants or both. Staff can get lazy or even crooked if they know that customer complaints will be brushed off. Another example is Henry Ford, who took the view that if you were injured by one of his cars, you should sue the driver, not him; it took decades for courts and lawmakers to nail down product liability.

Companies may also swing from being risk takers to being too risk averse, and back again. The personality of key executives does matter. My own university has been gung-ho when we hired an engineer to be Vice-Chancellor, timorous when we hired a lawyer, and in the middle when we hired a medic.

Another source of problems is when system design decisions are taken by people who are unlikely to be held accountable for them. This can happen for many reasons. IT staff turnover could be high, with much reliance placed on contract staff; fear of redundancy can turn loyal staff into surreptitious job-seekers. This can be a particular problem in big public-sector IT projects: none of the ministers or civil servants involved expect to be around when the thing is delivered seven years from now. So when working on a big system project, don't forget to look round and ask yourself who'll take the blame later when things go wrong.

Yet another is that when hiring security or safety consultants to help with product design, firms have an incentive to go for a firm that is ‘good enough’ but will not be too demanding; a gentle review from a Big Four firm will be much more useful than a detailed review from an expert who might recommend much more expensive design changes. Indeed, if a firm was determined to get a completely secure product, then they should hire multiple experts. We described

in section 14.2.3 how this helped with the design of prepayment electricity meters, and a later experiment with students confirmed that the more people you got to think about a proposed system design, the more potential hazards and vulnerabilities they could spot [68]. Of course, this rarely happens.

## 27.6 Managing the Team

To develop secure and reliable code, you need to build a team with the right culture, the right mix of skills, and the right incentives.

Many modern systems are already so complex that few developers can cope with all aspects of them. So how do you build strong development teams with complementary skills? This has been a subject of vigorous debate for over fifty years now, with different writers reflecting their personal style or company culture. It has long been entangled with cultural issues such as diversity, although these have only got serious attention since the mid-2010s.

### 27.6.1 Elite engineers

Going back to the 1960s, Fred Brooks's famous book, 'The Mythical Man-Month', describes the lessons learned from developing the world's first large software product, the operating system for the IBM S/360 mainframe [328]. He describes the 'chief programmer team', a concept evolved by his colleague Harlan Mills, in which a chief programmer – a development lead, in today's language – is supported by a toolsmith, a tester and a language lawyer. The thinking was that some programmers are much more productive than others, so rather than promoting them to management and 'losing' them you create posts for them with the salary and esteem of senior managers. The same approach was found in other tech companies in the 1960s through the 1980s, and even in bank IT departments where I worked in the late 1980s.

The view taken by more modern companies such as Microsoft, Google and Facebook is that you only want to hire the ultra-productive engineers in the first place – especially if you get a million CVs a year but plan to hire only 20,000 new engineers. One approach is to hire people as contractors for a few months to see how they do; but that's harder with fresh graduates, as even bright students from elite schools can take a few months to become productive in a commercial team. Productivity is also a matter of culture; engineers who thrive at one company may do much less well at another. A related issue is that if you have each candidate interviewed by a number of your engineers, that's not just a drain on engineer time, but can also perpetuate a culture that's not very welcoming to women engineers. Elite universities are in a similar situation to the tech majors, with dozens of applicants for each place; over the years we've learned to have mechanisms to monitor diversity in hiring and admissions.

The two approaches are not in conflict. Modern tech firms employ multiple tech superstars from internationally known designers to Turing-award winning computer scientists. The view at one such firm is that you cannot expect to write good software if you don't have a career structure for programmers. People who

want to spend their lives writing software, and are good at it, have to get respect, however your organisation signals that – whether it's salary, bonuses, stock or fripperies like access to the executive dining room. Universities get this; we professors run the place. Tech companies get it too, and one or two banks have started to. But governments are generally appalling. In the UK civil service, the motto is that “scientists should be on tap but not on top.” And more than one car company I know of has real problems hiring and retaining decent software engineers. In one of them, software engineers are expected to become managers after five years or remain on a junior pay grade, while in another all engineers are expected to wear business suits to work (and still paid lousy money). I'll return to this in section 27.6.6.

### 27.6.2 Diversity

At the beginning of computing, there were plenty of women programmers – they were the majority until the late 1960s, and included pioneers such as Grace Hopper and Dame Stephanie Shirley (who ran her company for years as ‘Steve Shirley’). When I started in the early 1970s there was still a much better gender balance than today. There were minorities too; the orbital calculations for the Mercury, Gemini and Apollo missions were led by an African-American woman, Katharine Johnson. But things have become male-dominated in the USA and the UK. Since I became an academic in the 1990s, about a sixth of local computer science students have been women, despite significant efforts to recruit more women students. However, in the formerly communist countries of Eastern Europe, the ratio is about a third. (We've improved our gender balance by admitting lots of students from southern and eastern Europe.) In India there's close to gender balance. So this is a cultural issue, and there's a lot of debate on how it came about. Is it a lack of role models, or is it the fault of careers advisers in schools, or are many IT shops just an unpleasant working environment for women? That has certainly been an issue: the Gamergate scandal, which I discussed in section 2.5.1, exposed deep misogyny in some gaming communities, while the #MeToo movement has highlighted many cases of sexism in Silicon Valley.

Even within computer science we see a lot of subcultural variation. The last time I went to a hardware conference – an Arm developer event – I saw about 500 men but only three women (all of them Indian). In the security field, we were overwhelmingly male in the 1990s when the emphasis was cryptology and operating system internals, but are much more balanced now we have embraced the importance of design, usability and psychology. Role models and history do matter. Research groups with a woman faculty member get more applications from able women<sup>7</sup>.

More diverse teams are more effective, and the real change doesn't come with the first woman you hire, but when you have enough to change the team culture. That might mean three or more. It also means getting more enlightened managers. Clearly it's a bad idea to hire misogynistic bullies, though it can be hard to spot them in advance. More subtly, if you want to attract more women

---

<sup>7</sup>We have gender balance in our natural language processing group, started in the 1960s by the late Karen Spärck Jones.

and retain them, it can be an idea to manage the people rather than the work. You have to protect your staff and give them space to do what they're good at. Bullies are often creeps too; as well as bossing the people under them they suck up to the people above them. Very often such people don't understand what's going on technically so they have no idea who's productive and have to judge people by timekeeping or by how much they ingratiate themselves. If this management style spreads through an organisation, my advice would be to go somewhere else.

### 27.6.3 Nurturing skills and attitudes

Modern development has a tension between the desire to keep teams together, so that they get more efficient and predictable, and moving people around to develop their skills, stop them going stale, and ensure that there's more than one person able to maintain everything that matters.

You will also need a diversity of skills. If you're writing an app, for example, you may want a couple of people to write the Android code, a couple for the Apple code and a couple for the server. Depending on the task, there may be a user advocate who leads usability testing; advocates for safety or security; an architect whose job is to keep the overall design clean and efficient; a language lawyer who worries about APIs, a test engineer who runs the regression testing machinery and a toolsmith who maintains the static and dynamic analysis tools. If you're doing continuous integration you'll have an engineer specialising in A/B testing while if you have a gated approach the test emphasis might be on compatibility with third-party products or with security certification. You'll need to give some thought to how many of these skills you try to get in each dev, and how many are subject matter experts who work across teams or come in as consultants. And as you can't run a project with more people than you can feed from two pizzas, you want some of your people to have two or more of these skills. Good tech firms rotate engineers slowly through the company to acquire a range of skills that maximises their value to the firm (even though it also maximises their value to others, and makes it easier for them to leave) [1209].

But skills are not enough: you need to get people to work together. Here, too, working practices have evolved over the years. By about 2010, agile developers had adopted the 'scrum' where the whole dev team has a stand-up meeting for five minutes each day, at which the only people allowed to speak are the developers. They describe what they've done, what they're about to do and what the problems are. Some firms have moved teams to collaboration tools such as Jira. In our team we combined daily lunches together with a formal progress meeting once a week. (Since the coronavirus lockdown the formal meeting has become more important and we've worked to complement it with other online activities.)

It's bad practice if people who find bugs (even bugs that they coded themselves) just fix them quietly; as bugs are correlated, there are likely to be more. Bug tracking matters, and a ticketing system that enables good statistics to be kept is an important tool in improving quality. As an example of good practice, in air traffic control it's expected that controllers making an error should not only fix it but declare it at once by open outcry: "I have Speedbird 123 at

flight level eight zero in the terminal control area by mistake, am instructing to descend to six zero.” That way any other controller with potentially conflicting traffic can notice, shout out, and coordinate. Software is less dramatic, but is no different: you need to get your devs comfortable with sharing their experiences, including their errors.

Another factor in team building is the adoption of a standard style. One signal of a poorly-managed team is that the codebase is in a chaotic mixture of styles, with everybody doing their own thing. When a programmer checks out some code to work on it, they may spend half an hour formatting it and tweaking it into their own style. Apart from the wasted time, reformatted code can trip up your analysis tools. You also want comments in the code, as people typically spend more time reading code than writing it. You want to know what a programmer who wrote a vulnerability thought they were doing: was it a design error, or a coding blunder? But teams can easily fight about the ‘right’ quantity and style of comments. So when you start a project, sit everyone down and let them spend an afternoon hammering out what your house style will be. Provided it’s enough for reading the code later and understanding bugs, it doesn’t matter hugely what the style is: but it does matter that there is a consistent style that people accept and that is fit for purpose. Creating this style is a better team-building activity than spending the afternoon paintballing, or whatever the latest corporate team-building fad happens to be.

#### **27.6.4 Emergent properties**

One debate is whether you make everyone responsible for securing their own code, or have a security guru on whom everyone relies. The same question applies to safety in fields such as avionics. The answer, as the leading firms have discovered, is ‘both’. We already noted that Microsoft found it more effective to have developers responsible for evolving their own designs and fixing their own bugs, rather than splitting these functions between analysts, programmers and testers, as IBM did in the last century. Both Microsoft and Google now put rookie engineers through a security ‘boot camp’, so that everyone knows the basics, and also have subject matter experts at a number of levels. These range from working security consultants with a masters degree or the equivalent internal qualification, to people with PhDs in the intricate details of cryptography or virtualisation.

The trick lies in managing the amount of specialisation in the team, and the way in which the specialists (such as the security architect and the testing guru) interact with the other developers.

#### **27.6.5 Evolving your workflow**

You also need to think hard about the tools you’ll use. Professional development teams avoid a large number of the problems described in this book by using appropriate tools. You avoid buffer overflows by using a modern language such as Rust, or if you must use C or C++ then have strict coding conventions and enforce them using static-analysis tools such as SonarQube and Coverity.

You avoid crypto problems, such as timing attacks and weak random number generators, by using well-maintained libraries. But you need to understand the limitations of your tools. In the case of Coverity, for example, its authors explain that while it's great if you use it from the start of a project, adopting it in midstream imposes real costs, as you suddenly have 20,000 more bug reports to triage, and your ship date slips by a few months [235]. Improvements in static analysis tools, say in response to a new kind of attack, can also throw up a lot of alarms in an existing codebase. In the case of crypto libraries, we discussed in Chapter 6 how they tend to offer weak modes of operation such as ECB as defaults, so you need to ensure your team uses GCM instead. (Crypto is one of the areas where you need to talk to a subject matter expert.)

You'll be constantly adding new tools, whether to avoid cross-site scripting vulnerabilities and SQL injection as you update your website, or to make sure you don't leave your client data world-readable in an S3 bucket. If you don't follow the security news you may not be aware of the latest exploits and attacks, so you may not realise when you have to either grow your own expertise or buy it in. However you can't just buy everything in; the security industry has lots of unscrupulous operators who exploit ignorant customers. You need to understand what you need to buy, and why, and then you will need to integrate it with your existing tools, or your security ops people will spend ever more of their time copying IP addresses from one tool to another. Doing some of your own automation helps empower your staff as well as saving time.

Your tools and libraries have to support your architecture. One critical thing here is that you need to be able to evolve APIs safely. A system's architecture is defined more than anything else by its interfaces, and it decays by a thousand small cuts: by a programmer needing a file handling routine that uses two more parameters than the existing one, and who therefore writes a new routine – which may be dangerous in itself, or may just add to complexity and thus contribute indirectly to an eventual failure. In an ideal world, you'd rely on your programming language to prevent API problems using type safety mechanisms.

But the cross-system fan-out of dependencies is a real hazard to safe APIs. We saw in section 20.5 how the APIs of cryptographic hardware security modules were extended to support hundreds of banks' legacy ATM systems until we suddenly realised that the resulting feature interactions made them completely insecure. There are similar tensions in many other application areas, from mobile phone baseband software used in over a hundred different models of phone, to vehicle components used in over a hundred different cars. There must be better ways of managing this; I expect that applications with high fan-out will move in the direction of a microservices architecture with a common core and pluggable proxies for different calling applications.

### 27.6.6 And finally...

You also need to understand how to manage people, and the HR department can't do this for you<sup>8</sup>. Tech management cannot be done by generalists as

---

<sup>8</sup>The main job of HR is damage limitation – stopping leavers from suing you.

## 27.7. SUMMARY

---

they're unlikely to win the trust of their staff<sup>9</sup>. It also cannot be done well by engineers who are too introverted to engage and motivate others. Far too many managers went for the job not because they thought they might be good at it, but because it was the only way to get a decent salary. Successful managers in tech have to love and understand tech; they also have to love and understand people.

For your star engineers, you need to create other leadership roles. They may be innovators who will be most productive in an R&D lab. They may be the custodians of your institutional memory: old-timers who know the thirty years of history behind your product and can stop people repeating the mistakes of the past. They may provide moral leadership to your engineering staff and reassurance to your customers. They can help attract bright young recruits who want to work with them. But the key, I feel, is this: that you have one or more engineering professions in your firm. What's their shape? Who leads them? How do they compare to those in your competitors? How do you grow and develop them? If you realise that all of a sudden you have to unify the safety engineering and security engineering professions in your company, who is going to do that, and how?

## 27.7 Summary

Managing a project to build, or enhance, a system that has to meet critical requirements for security, safety or both, is a hard problem. As more and more devices acquire CPUs and communications, we need to build things that do real work while keeping out any vulnerabilities that would make them a target for attack. In other words, you want software security – together with other functionality, and other emergent properties such as safety and real-time performance.

If you're building something entirely new, or a major functional enhancement of an existing system, then understanding the requirements is often the hardest part of the process. More gentle system evolution can involve subtler changes to requirements. Larger changes can be forced externally; systems that succeed and get popular, can expect to get attacked.

Writing secure code is hard because of this dynamic context: the first problem is to figure out what you're trying to do. However, even given a tight specification, or constant feedback from people hacking your product, you're not home and dry. There are a number of challenges in hiring the right people, giving them the right tools, helping them develop the right ways of working, backing them up with expertise in the right way, and above all creating an environment in which they work to improve their security capability.

---

<sup>9</sup>As a math geek I always tended to see the MBA types and other corporate politicians much as the Earl of Rochester saw King Charles II: "Here lies our sovereign lord the king, Whose word no man relies on; He never says a foolish thing, Nor ever does a wise one."

## Research Problems

The issues discussed in this chapter are among the hardest and the most important of any in our field. However, they receive little attention because they lie at the boundaries with software engineering, applied psychology, economics and management. Each of these interfaces could be a productive area of research. Security economics and security psychology have made great strides in the last few years, and we now know we need to do a lot more work on making security tools easier for developers to use. One logical next step is integrating what we know with safety economics and safe usability.

Yet many failures are due to organisational behaviour. Every experienced developer or security consultant has their share of horror stories about firms with perverse incentives, toxic cultures, high staff turnover, incompetent management and all the rest of the things we see in the Dilbert cartoons. It could be useful if someone were to collect a library of case histories of security failures caused by unsatisfactory incentives in organisations, such as [876, ?]. What might follow given a decent empirical foundation?

The late Jack Hirshleifer took the view that we should try to design organizations in which managers were forced to learn from their mistakes: how could we do that? How might you set up institutional structures to monitor changes in the threat environment and feed them through into not just systems development but into supporting activities such as internal control? Maybe we need something like Management as Code? How can you design an organization that is ‘safety-incentive-compatible’ in the sense that staff behave with an appropriate level of care? And what might the cultural anthropology of organisations have to say? We saw in the last chapter how the response of governments to the apparently novel threats posed by Al-Qaida was maladaptive in many ways: far too much of our social resilience budget was spent on anti-terror theatre, at the expense of preparedness for other societal risks such as pandemics. Similarly, far too much of the typical firm’s resilience budget has been captured by compliance, safety theatre and security theatre. As a result, too much of the security development effort is aimed at compliance rather than managing security and safety risks properly. How can we design feedback mechanisms that will enable us to put the right amount of effort in the right place? Or do we need broader structural change, such as the breakup of the Big Four accountancy firms?

## Further Reading

Managing the development of information systems has a large, diffuse and multidisciplinary literature. There are classics everyone should read, such as Fred Brooks’s ‘Mythical Man Month’ [328] and Nancy Leveson’s ‘Safeware’ [1149]. The economics of the software life cycle are discussed by Brooks and by Barry Boehm [272]. The modern books everyone should read, as of 2020, are probably the Google books on SRE [236] and on ‘Building Secure and Reliable Systems’ [23]. The Microsoft approach to the security development lifecycle has many online resources; their doctrine on threat modelling is discussed by Frank Swiderski and Window Snyder [1851]; and their security VP Mike Nash de-

## 27.7. SUMMARY

---

scribes the background to the big security push and the adoption of the security development lifecycle at [1385]. The most general set of standards on safety functional and integrity requirements, and the associated engineering processes, is IEC 61508; there are further sets of industry-specific standards. For example, there's IEC 61511 for process plant control systems, IEC 62061 for safety of machinery, and the EN 5012x series for railways. In aviation it's RTCA DO-254 for electronic hardware and RTCA DO-178C for software, while in the motor industry it's ISO 26262 for safety and ISO 21434 for security – though at the time of writing this is still just a draft. Standards for the Internet of Things are also a work in progress, and the current draft is ETSI EN 303 645 V2.1.

We can learn a lot from other engineering disciplines. Henry Petroski discusses the history of bridge building, why bridges fall down, and how civil engineers learned to learn from the collapses: what tends to happen is that an established design paradigm is stretched and stretched until it suddenly fails for some unforeseen reason [1518]. IT project failures are another necessary subject of study; there's a casebook on how to manage uncertainty in projects by Christoph Loch, Arnoud DeMeyer and Michael Pich [1179]. For security failures, it's important to follow the leading security blogs such as Schneier on Security, Krebs on Security and SANS, as well as the trade press.

Organizational aspects are discussed at length in the business school literature, but this can be bewildering to the outsider. Many business academics praise business, which is fine for selling airport books, but what we need is a more critical understanding of how organisations fail. If you're only going to read one book, make it Lewis Pinault's 'Consulting Demons' – the confessions of a former insider about how the big consulting firms rip off their customers [1527]. Organisational theorists such as Charles Handy talk of firms having cultures based on power, roles, tasks or people, or some combination. It's not just who has access to whom, but who's prepared to listen to whom and who will just ignore orders from whom. Perhaps such insights might help us design more effective tools and workflows that support how people actually work best.