**Chapter 20**

**Advanced Cryptographic**  
 **Engineering**

**Give me a rock on which to stand, and I will move the world.**

– Archimedes

**Whoever thinks his problem can be solved using cryptography,**

**doesn’t understand his problem and doesn’t understand**

**cryptography**

– Attributed by Roger Needham and Butler Lampson to each other

**20.1** **Introduction**

Cryptography is often used to build a trustworthy component on which more  
 complex designs can rely. Such designs come from three rather di↵erent back-  
 grounds. The ﬁrst is the government systems world we described in Chapter  
 9, where the philosophy is to minimise the trusted computing base using mech-  
 anisms like data diodes and multilevel secure encryption devices. The second  
 is the world of banking described in Chapter 12 where smartcards are used  
 as authentication tokens while HSMs are used to protect PINs and keys. The  
 third is the world of cryptography research in the 1980s and 1990s where people  
 dreamed of solving social problems using mathematics: of creating anonymous  
 communications so that oppressed groups could evade state surveillance, lead-  
 ing to censorship-resistant publishing, untraceable digital cash and electronic  
 elections that would be impossible to rig. In all these cases, real life turned out  
 to be somewhat messier than we anticipated.

There are even more complex cryptographic components that we use as

platforms. But the engineering isn’t just about reducing the attack surface, or  
 simplifying our fault tree analysis. In most cases there’s a signiﬁcant interaction  
 with policy, liability and other complicating factors.

In this chapter I’m going to discuss six examples of cryptographic engineering

606

*20.2. FULL-DISK ENCRYPTION*

– full disk encryption, the Signal protocol, Tor, hardware security modules,  
 enclaves and blockchains. The ﬁrst is a simple example to set the scene; the other  
 ﬁve use crypto in more complex ways to support a wide range of applications,  
 including payments in the case of the last three. All but HSMs are used by  
 cybercriminals.

Hard disk encryption has been around since the 1980s and is one of the

simplest security products, at least conceptually. By encrypting the data on  
 your hard disk when the machine’s in use, you ensure that a thief can only steal  
 the hardware, not the data.

Signal is a protocol for secure messaging between phones. It is perhaps the

next level up in complexity and is about enabling people to manage a social  
 network as securely as possible in the face of equipment compromise. Signal  
 does private contact discovery by means of enclaves.

Tor takes this to the next level by providing anonymity, when you don’t want

someone observing your tra�c to know who you’re talking to or which websites  
 you’re visiting.

HSMs have provided a trust platform for payment services since the 1980s.

But the crypto apps that run on them can su↵er from attacks on their *application*  
 *programming interfaces* that are so deeply entangled with payment applications  
 that they are very hard to ﬁx.

Enclaves are an attempt by CPU vendors to provide a general purpose crypto

platform: we’ve had Arm’s TrustZone since 2004 and Intel’s SGX since 2015.  
 They are starting to replace HSMs in payment applications, and also support  
 private contact discovery in Signal. But they have been plagued with problems  
 from side-channel attacks to class breaks. For example, if you can extract the  
 master secret key from an SGX chip, you can break the whole ecosystem.

Finally, for a quite di↵erent kind of trusted computer, we look at Bitcoin.

This is a project, since 2009, to create a digital currency based on a shared ledger  
 that emerges using cryptographic mechanisms from the cooperation of mutually  
 mistrustful parties. Many of the stakeholders are far from trustworthy, and there  
 are dominant players at several levels in the technology stack. Yet a trusted  
 computer has somehow emerged, thanks to a combination of cryptography and  
 economic incentives, and has kept going despite the huge amounts of money  
 that could be taken in a successful attack.

It may be useful to bring together in one chapter the trusted platforms

of both bankers and gangsters, so we can contrast them. Some striking facts  
 emerge. For example, the best attempts of the top technology companies to  
 produce trusted computers have produced ﬂawed products, while the gangsters  
 seem to have created something that works – at least for now.

**20.2** **Full-disk encryption**

The idea behind *full-disk encryption* (FDE) is simple. You encrypt data as

it’s written to disk, and do decryption as it’s read again. The key depends

on an initial authentication step such as a password, which is forgotten when

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 607 | Ross Anderson |

*20.2. FULL-DISK ENCRYPTION*

the machine sleeps or is switched o↵. So if a doctor leaves their laptop on a  
 train, only the hardware is lost; the medical records are not. FDE has become  
 a regulatory requirement in many industries. In Europe, privacy regulators

generally see the loss of machines with FDE as not serious enough to attract  
 a ﬁne or to need mandatory notiﬁcation of data subjects. Many phones and  
 laptops come with FDE; with some it’s enabled by default (Android) while with  
 others it just takes a click (Mac).

Scratch a little under the surface, though, and there’s a wide variance in

quality. From the early days of hard disks in 1980s, software FDE products  
 were available but imposed a performance penalty, while hardware products  
 cost more and were export-controlled. The engineering isn’t trivial, as you

need a platform on which to run the initial authentication step. Early products  
 o↵ered an extra encrypted volume but did not protect the host operating system  
 and could be defeated by malware. The initial authentication is tricky in other  
 ways. If you derive the disk key from a user password, then a thief can try  
 zillions of them o✏ine, as we discussed in 3.4.4.1, and guess anything a normal  
 user sets up. A hardware TPM chip can limit password guessing, and from  
 2007 this became available for Windows with Bitlocker. Integrating FDE into a  
 platform enables the vendor to design coherent mechanisms for trusted boot of  
 an authentic copy of the operating system, setting up and managing recovery  
 keys, and coping with quite complex interactions with software upgrade, swap  
 space, device repairs, the backup and recovery of user data, and factory reset  
 when the device is sold.

Third-party o↵erings started to o↵er some extra features: TrueCrypt, for

example, o↵ered a steganographic ﬁle system where the very existence of a disk  
 volume would remain hidden unless the user knew the right password [114]1.  
 A crypto phone sold to criminals, EncroChat, had a whole hidden partition  
 containing encrypted chat and VOIP apps; I’ll discuss such products in more  
 detail in section 25.4.1. However most people now use the FDE facility provided  
 by the vendor of their phone or laptop, as proper integration involves quite a lot  
 of the platform. Since 2010 we’ve had a special mode of operation, XTS-AES,  
 designed for FDE; it encrypts each block salted with the sector number, and has  
 a mechanism to ﬁt disc blocks to block ciphers. O↵erings such as Microsoft’s  
 BitLocker and Apple’s FileVault have an overhead of only a few percent, when  
 run on CPUs with AES support.

Yet attacks continue. In 2008, Alex Halderman and colleagues at Princeton

came up with *cold boot attacks*, which defeated the principal FDE products then  
 on the market and can still present a problem for many machines [854]. As I  
 described in section 18.3, you freeze a computer’s DRAM in which the transient  
 encryption key is stored, then reboot the device with a lightweight operating  
 system and acquire a memory image, from which the key can be read. In 2015,  
 we found that most Androids were insecure: the factory reset function was so  
 badly engineered by most OEMs that credentials, including FDE keys, could be  
 recovered from second-hand devices [1757]. And most Android phones don’t get

1That product was suddenly discontinued and its anonymous developers recommended

that users migrate to other products because of an unspeciﬁed vulnerability; some suspect  
 that this was a ‘warrant canary’, a pre-planned warning message whose transmission the  
 developers suppress by certifying regularly that they are not subject to coercion, but which  
 ﬁres o↵ a warning once they’re served with a subpoena or warrant [61].

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 608 | Ross Anderson |

*20.3. SIGNAL*

patched once they’re no longer on sale. And in 2019, Carlo Meijer and Bernard  
 van Gastel found that the three third-party FDE products that held 60% of the  
 market were insecure, that open-source software encryption would have been  
 better, and that Bitlocker turned itself o↵ if one of these hardware products  
 appeared to be present; thanks to their work, it no longer does so [1285]. And  
 then there’s the collateral damage. Now that lots of sensitive data are kept not  
 on hard disks but in Amazon S3 buckets, auditors routinely demand that these  
 buckets are encrypted; but as the failure mode of an S3 bucket isn’t a burglar  
 in Amazon’s data centre but negligence over access controls, it’s unclear that  
 S3 bucket encryption achieves anything other than tick-box compliance.

And ﬁnally one has to consider abusability, of which there are at least two

signiﬁcant kinds. First, the wide availability of FDE code is one of the two  
 components that led to the recent wave of ransomware attacks, where a gang  
 penetrates your systems, installs FDE, lets it run until you’ve encrypted enough  
 backups to make recovery painful, then demands a ransom for the key. (The  
 other component is cryptocurrency which I’ll discuss later in this chapter.) Sec-  
 ond, many people consider FDE to be magic insurance against compromise, and  
 won’t report a laptop left on a train if it had FDE enabled (or was supposed  
 to), even if the ﬁnder might have seen the password, or be able to easily guess  
 it.

So even the simplest of encryption products has a signiﬁcant entanglement

with compliance, is much more complex under the hood than you might think at  
 ﬁrst glance, usually imposes some performance penalty, and can be vulnerable to  
 a capable opponent – even years after the relevant attacks have been published.

**20.3** **Signal**

As smartphones spread round the world, people switched from SMS to messaging  
 apps such as WhatsApp, Telegram and Signal as they’re cheaper and more  
 ﬂexible, allowing you to create groups of families and friends. Pretty soon they  
 started supporting voice and video calls too, and o↵ering end-to-end encryption.  
 It had previously been possible to encrypt email using programs like PGP, but  
 it was rather ﬁddly (as we discussed in section 3.2.1) and remained a niche  
 activity. The arrival of new platforms meant that message encryption could  
 be made universal, shipped as a default with the app; the Snowden disclosures  
 helped stoke the public demand.

Signal is a free messaging app, initially developed by a man who uses the

name of Moxie Marlinspike. It set the standard for end-to-end encryption of  
 messaging, and its mechanisms have been adopted by competing products in-  
 cluding WhatsApp. Mobile messages can be highly sensitive, with everything  
 from lovers’ assignations through business deals to political intrigues at diplo-  
 matic summits; yet mobile phones are often lost or stolen, or sent in for repair  
 when the screens break. So key material in phones is frequently exposed to com-  
 promise, and it’s not enough to just have a single long-lived private key in an  
 app. The Signal protocol therefore provides the properties of *forward secrecy*,  
 that a key compromise today won’t expose any future tra�c, and *backward se-*  
 *crecy*, which means that it won’t expose previous tra�c either. These are now

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 609 | Ross Anderson |

*20.3. SIGNAL*

formalised as *post-compromise security* [451].

The protocol has three main components: the *Extended Triple Di�e-Hellman*

(X3DH) protocol to set up keys between Alice, Bob and the server; a ratchet  
 protocol to derive message keys once a secret key is established; and mechanisms  
 for ﬁnding the Signal keys of other people in your address book.

We can’t use vanilla Di�e-Hellman to establish a fresh key between Alice

and Bob, as they might not be online at the same time. So in the X3DH

protocol [1227], each user *U* publishes an identity key *IKU* and a prekey *SKU*  
 to a server, together with a signature on the latter that can be veriﬁed using  
 the former. The algorithms are elliptic-curve Di�e-Hellman and elliptic-curve  
 DSA. When Alice wants to send a message to Bob, she fetches Bob’s keys *IKB*  
 and *SKB* from the server, generates an ephemeral Di�e-Hellman key *EKA*,  
 and combines them with Bob’s keys in all the feasible ways: *DH*(*IKA, SPKB*),  
 *DH*(*EKA, IKB*), and *DH*(*EKA, SPKB*). These are hashed together to give  
 a fresh key *KAB*. Alice then sends Bob an initial message containing her keys  
 *IKA* and *EKA*, a note of which of Bob’s prekeys she used, and a ciphertext  
 encrypted using *KAB* so that he can check he’s got it too. Optionally, Bob can  
 upload a one-time ephemeral key that Alice will combine with *EKA* and hash  
 into the mix.

Given an initial Di�e-Hellman key *KAB*, Alice and Bob then use the *dou-*

*ble ratchet algorithm* to derive message keys for individual texts and calls. Its  
 purpose is to recover security if one of their phones is compromised. It uses  
 two mechanisms: a *key derivation function* (KDF) or one-way hash function to  
 update stored secret keys, and further Di�e-Hellman key exchanges. Alice and  
 Bob each maintain separate *KDF chains* for sending and for receiving, each with  
 a shared-secret key and a Di�e-Hellman key. Each message carries a new Di�e  
 Hellman key part which is combined with the key for the relevant chain, while  
 the shared-secret key is passed through the KDF. The actual details are slightly  
 more ﬁddly, because of the need to deal with out-of-order messages [1512]. The  
 goal is that an opponent must compromise either Alice’s phone or Bob’s con-  
 tinuously in order to get access to the tra�c between them.

The really tricky part is the initial authentication step. If Charlie could take

over the server and send Alice his own *IK* instead of Bob’s, all bets are o↵. This  
 is the attack being mounted on messaging apps by some intelligence agencies.  
 Systems such as Apple’s iMessage don’t just send a single identity key *KI* to  
 your counterparty but a whole keyring of device keys – one for each of your  
 MacBooks, iPhones and other Apple devices. Ian Levy and Crispin Robinson of  
 GCHQ propose that laws such as the UK’s Investigatory Powers Bill be used to  
 compel providers to add an extra law-enforcement key to the keyring of any user  
 against whom they get a warrant [1153]. This has led to policy tussles in the  
 USA, the UK and elsewhere, to which I return in section 26.2.8. Keeping such  
 surveillance covert will depend on the phone app software remaining opaque  
 to users; otherwise the double ratchet algorithm will prevent Alice and Bob’s  
 private conversation being joined by Charlie as a silent conference call partner,  
 or ‘ghost user’. Signal attempts to forestall this by being open source.

The upshot is that if Charlie wants to exchange Signal messages with Alice

while pretending to be Bob, he has to either compromise Bob’s phone or steal

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 610 | Ross Anderson |

*20.3. SIGNAL*

Bob’s phone number. The options are much the same as if he wanted to steal  
 money from Bob’s bank account. They include hacking and stealing the phone;  
 using SS7 exploits to steal Bob’s SMS messages; and a SIM swap attack to take  
 over Bob’s phone number. The easiest attack for an individual to mount is

probably SIM swapping, which we discussed in section 12.7.4. Signal now o↵ers  
 an additional PIN that you need to enter when recovering service on a phone  
 number on which a di↵erent handset was previously active. Nation states have  
 sophisticated hacking tools, and have SS7 access – so if the FSB’s in your threat  
 model, it’s best to use a phone whose number they don’t know, and don’t carry  
 it around switched on at the same time as a phone they do know is yours, or  
 they might correlate the traces – as I described in section 2.2.1.10.

As we will discuss in section 26.2.2, much of the beneﬁt of signals intelligence

comes from metadata, from knowing who called whom and when (or who trav-  
 eled with whom and when). So for a whistleblower, the game depends on how  
 many other people will become suspects as well as you – the *anonymity set*. If  
 you’re a senior civil servant thinking of leaking an illegal policy to a newspaper,  
 and you’re one of ten people who knows the story, then you might be the only  
 one of the ten who has ever used Signal.

However, if you’re one of hundreds of low-level suspects (say you’re a union

organiser or NGO sta↵er) and might be on a long list of targets for thematic  
 collection, then you may want to block the local police from systematically  
 recording your patterns of contacts, and here Signal can indeed help. It o↵ers  
 the interesting innovation of *private contact discovery*.

Previous attempts to help ordinary people use end-to-end encryption, such

as the email encryption program PGP, never got much traction outside specialist  
 niches because key management was too much bother. Messaging apps solved  
 the usability problem by demanding access to your address book, looking up  
 all your contacts on their servers to see who else was a user and then ﬂagging  
 them so you know you can message them. However, giving service ﬁrms a

copy of your address book is already a privacy compromise, and if you also let  
 them keep a plaintext record of your social graph, proﬁle name, location, group  
 memberships and who is messaging whom, then investigators can get all this by  
 subpoena. The original version of Signal compared hashes of the phone numbers  
 in people’s address books to discover who was using it; however, Christof Hagen  
 and colleagues used 100 accounts over 25 days to scan all 505m phone numbers  
 in the USA, discovering 2.5m Signal users [848]. Signal has now implemented  
 private contact discovery; I will discuss it later in section 20.6 which discusses  
 SGX, the mechanism it uses. However, when you set up a Signal account on  
 your phone, even private contact discovery makes this fact immediately apparent  
 to everyone in your address book who’s also on Signal (and they’ll say – ‘Hey,  
 Fred’s about to leak something’ – so a careful leaker would buy a burner phone  
 for cash.)

A critical but less visible part of the system is the message server. This has

to store encrypted messages that have not yet been delivered but how much  
 else is kept and for how long2? Signal keeps records of group memberships,  
 but there’s now a proposal for anonymous group messaging, which would make

2There was a debate about how to handle undelivered messages when keys change, and

the WhatsApp implementation was criticised for prioritising delivery over failing closed.

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 611 | Ross Anderson |

*20.4. TOR*

group members known to each other but not to Signal’s servers [409]. Again,  
 technology can only do so much; if one member of your group is disloyal, they  
 can betray others. However Signal has got real traction as the leading commu-  
 nications security tool available to the public. There was a signiﬁcant uptick in  
 usage in the USA after the 2016 election, and in 2020 the European Commission  
 (Europe’s civil service) ordered its sta↵ to switch to Signal after the compromise  
 of a server containing thousands of diplomatic cables [399].

There was an upset in July 2020, when a Signal update forced users to select

a PIN, with a view to keeping each user’s contact data encrypted in an enclave,  
 so it could be recovered if the user got a new phone, and so that there could be  
 some other way to make a Signal contact other than by sharing a phone number.  
 This created a storm of protest as users assumed that Signal would also keep  
 message content; other users didn’t think a PIN gave enough protection, or  
 didn’t want to give Signal a PIN they used for banking, or just didn’t like the  
 idea of any centralised data at all. People started questioning the wisdom of  
 relying on a secure communications app whose chief maintainer is someone who  
 uses a pseudonym, who can hold millions of users hostage on a whim, and whose  
 backing was partly from the government and partly from a billionaire3. What  
 should the governance of public-interest critical infrastructure look like?

Signal claims to keep no records of tra�c, but what if a FISA warrant from

the NSA had forced them to do so and lie about it? This brings us to the harder  
 question of how communications can be made anonymous.

**20.4** **Tor**

*The Onion Router* (Tor) is the main system people use to get serious anonymity  
 online, with about 2 million concurrent users in 2020. It began its life in 1998  
 at the US Naval Research Laboratory, and was called Onion Routing because  
 messages in it are nested like the layers of an onion [1590]. If Alice wants to  
 visit Eve’s website without Eve or anyone else being able to identify her, she  
 sets up a TLS connection to a Tor relay operated by Bob, which sets up a TLS  
 connection to a Tor relay operated by Carol, which in turn a TLS connection to  
 a Tor relay operated by David – from whose ‘exit node’ Alice can now establish a  
 connection to Eve’s website [1360]. The idea is to separate routing from identity  
 – anyone wanting to link Alice to Eve has to subvert Bob, Carol and Dave, or  
 monitor the tra�c in and out of Bob’s and David’s systems.

The inspiration had been a 1981 idea of David Chaum’s, the *mix* or *anony-*

*mous remailer* [410]. This accepts encrypted messages, strips o↵ the encryption,  
 and then remails them to the address that it ﬁnds inside. Various people ex-  
 perimented with these in the 1990s and found that you need three more things  
 to make it work properly. First, you need more than one mix; an opponent  
 could compromise a single mix by coercing the operator, or simply correlating  
 the tra�c in and out. Second, you need to engineer it for the tra�c you want  
 to protect, be that email, web or messaging. Third, and hardest of all, you need  
 scale.

3Brian Acton, one of the founders of WhatsApp.

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 612 | Ross Anderson |

*20.4. TOR*

The Navy opened Tor up to the world in 2003 because you can only be

anonymous in a crowd. If Tor had been restricted to US intelligence agents, then  
 anyone using it would be a target. It is now maintained by the Tor Project, a  
 US nonproﬁt that maintains the Tor Browser, which has become the default Tor  
 client. This not only handles circuit setup and encryption but manages cookies,  
 javascript and other browser features that are hazardous to privacy. Similar  
 functionality is also built into some other browsers, such as Brave. There’s

also software for Tor relays, which are run by volunteers with high-bandwidth  
 connections; in 2020, about 6,000 active relays serve about 2 million users. When  
 you turn on a Tor-enabled browser, it opens a circuit by ﬁnding three Tor relays  
 through which it connects to the outside world.

Tor’s cryptographic and software design has evolved over 20 years in the face

of a variety of threats and abuse, and it is now used as a component in many  
 applications. It’s used to defeat censorship in countries like Iran and Pakistan so  
 you can connect to Facebook and read American and European newspapers. The  
 US State Department supports it, and Facebook is the biggest Tor destination.  
 It can also be used to connect to underground dark markets where you can buy  
 drugs and malware. It can be used to leak classiﬁed documents. It can be used  
 to visit child sex abuse websites. The police also use it to visit such sites, so the  
 operators don’t know they’re police.

The principal vulnerabilities were known from day one and documented in

the 1998 paper that introduced onion routing to the world, six years before  
 Tor itself appeared [1590]. But they have frequently been overlooked by care-  
 less users. First, a *malicious exit node* can monitor the tra�c if Eve’s website  
 doesn’t use encryption, or if she uses it in such a way that the exit node can do  
 a man-in-the-middle attack. In September 2007, someone set up ﬁve Tor exit  
 nodes, monitored the tra�c that went through them, and published the inter-  
 esting stu↵ [1359]. This included logons and passwords for a number of webmail  
 accounts used by embassies, including missions from Iran, India, Japan and  
 Russia4. Yet the Tor documentation made clear that exit tra�c can be read,  
 so more careful diplomats would have used a mail service that supported TLS  
 encryption, as Gmail already did by then.

The second problem is the many tricks that web pages employ to track users.

This was the main reason for the introduction in 2008 of the Tor Browser, which  
 limits the tracking ability of cookies and other ﬁngerprinting mechanisms. But  
 many applications get users to identify themselves explicitly, or leak information  
 without realising it. In section 11.2.3 I discussed how supposedly anonymous  
 search histories from AOL identiﬁed users: a few local searches (that tell where  
 you live) and a few special-interest searches (that reveal your hobbies) can be  
 enough.

Third, low-latency, high-bandwidth systems such as Tor have some intrinsic

exposure to tra�c analysis [1363]. A global adversary such as the NSA, that  
 taps tra�c at many points in the Internet, need only tap a small number of  
 exchange points to get a good enough sample to reconstruct circuits [1365]. In  
 practice this is harder than it looks5. Tor has made clear since the start that

4This gave an insight into password choice: Uzbekistan came top with passwords like

‘s1e7u0l7c’ while Tunisia just used ‘Tunisia’ and an Indian embassy ‘1234’.

5The intelligence community paid a compliment to Tor, on a GCHQ slide deck leaked by

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 613 | Ross Anderson |

*20.4. TOR*

it does not protect against tra�c conﬁrmation attacks, where the opponent  
 controls both the entry and exit relays and correlates the timing, volume or  
 other characteristics of the tra�c to identify a particular circuit. Indeed, in 2014  
 it was discovered that someone (presumably an intelligence agency) had been  
 doing just this, volunteering relays into the system that tinkered with protocol  
 headers in order to make it easier [561]. Tor relays now have countermeasures  
 against such tweaks, but tra�c conﬁrmation is still a threat.

Fourth, as Tor connects through a pool of some 6,000 relays, a ﬁrewall can

simply block their IP addresses. This is done by some companies and also by  
 some countries, most notably China. To circumvent such blocking, volunteers  
 make available *Tor bridges* – Tor entry nodes not listed in the public directory.  
 Various games are played as Chinese and other censors try to ﬁnd and block  
 these too, and to characterise Tor tra�c. China appears to prefer that people  
 circumventing its national ﬁrewall use VPNs instead; these are not only more  
 scalable but easier to shut down completely at times of crisis (such as in the  
 early stages of the 2020 coronavirus outbreak).

Law-enforcement agencies have on a number of occasions managed to ﬁnd

and close down *Tor onion services*, websites that are available only through the  
 Tor network; rather than a normal URL, they have a ‘.onion’ address that is  
 essentially a cryptographic key. The most famous such service was Silk Road,  
 an underground marketplace where people bought and sold drugs; its operator  
 was arrested because of poor operational security (the email address he used to  
 announce his new service could be traced back to him). Other onion services  
 have had their servers hacked, or supply chains traced. Many of them use

cryptocurrencies, which we’ll describe later and which can also be traced in  
 various ways. There have also been attacks on the browsers of Tor users with  
 techniques such as zero-days and sandbox escapes. And even in the absence of  
 technical failures, anonymity is intrinsically hard; real-world transactions (and  
 indeed real-world web tra�c) can be very dirty, so unexpected inferences can  
 often be drawn.

As with FDE, Tor has a signiﬁcant entanglement with compliance, helping

a variety of actors to evade surveillance and circumvent laws both good and  
 bad. The engineering has become a lot more complex under the hood than it  
 looks. It deﬁnitely imposes a performance penalty – websites can take a second  
 to load rather than a few hundred milliseconds. And despite the robustness of  
 the Tor system itself, it has intrinsic limitations that are not intuitively obvious  
 and make anonymity systems built on it hazardous to use. Anonymity systems  
 require careful operational security as well as just the right software.

The governance aspects are of interest. Tor is maintained by the Tor Project,

a US nonproﬁt set up in 2006 to formalise a volunteer project that had started  
 in 2002. Although it has many volunteers, a growing core of permanent sta↵  
 have been funded from various sources over the years, from the EFF to the US  
 State Department. It remains at heart an international community of people  
 motivated by human rights. An ethnographic study by Ben Collier describes  
 it as made up of three overlapping groups: a group of engineers who see Tor  
 as a structure, and believe that political problems can be solved by doing engi-

Ed Snowden, saying “Tor stinks!”

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 614 | Ross Anderson |

*20.5. HSMS*

neering; a group of activists see it as a struggle, and are committed to speciﬁc  
 political values such as anti-racism; while a third group of people largely main-  
 tain the Tor relays, are generally politically agnostic, and see what they do as  
 providing infrastructure – “privacy as a service” [458]. Security at scale requires  
 infrastructure, and to provide this largely by volunteer e↵ort requires leaders  
 who can translate between the di↵erent stakeholders’ agendas and negotiate  
 values rather than just contracts.

**20.5** **HSMs**

In the chapter on Banking and Bookkeeping, we described how banks use HSMs  
 to enforce a separation-of-duty policy: no single person at the bank should be  
 able to get their hands on a customer’s card details and PIN. HSMs are also  
 used to protect the SSL/TLS keys for many websites; you don’t want important  
 live keys to be sitting on a developerˆa˘A´Zs laptop, or to be easily extractable  
 by a cloud provider through a memory dump. In the cryptocurrency industry,  
 HSMs are used to protect keys that could sign away substantial assets. In the  
 chapter on Tamper Resistance, we described the mechanisms used to make the  
 HSM tamper-proof. But this isn’t enough. You also have to ensure that when  
 you split a computation between a more trusted component such as an HSM  
 and a less trusted component, an attacker can’t exploit the split.

Whenever a trusted computer talks to a less trusted one, you have to expect

that the less trusted device will lie and cheat, and probe the boundaries by using  
 unexpected combinations of commands, to trick the more trusted one. How can  
 we analyse this systematically?

Banking HSMs have a lot to teach. In 1988, Longley and Rigby identiﬁed the

importance of separating key types while doing work for security module vendor  
 Eracom [1184]. In 1993, we reported a security ﬂaw that arose from a custom  
 transaction added to a security module [107]. However we hit paydirt in 2000  
 when Mike Bond, Jolyon Clulow and I observed that HSM APIs had become  
 immensely complex, with hundreds of di↵erent transactions involving complex  
 combinations of cryptographic operations to support dozens of payment protocol  
 variants, and started to think systematically about whether there might be a  
 series of HSM transactions that would break it [71]. We asked: “How can you  
 be sure that there isn’t some chain of 17 transactions which will leak a clear  
 key?’ After we spent some time staring at the manuals, we started to discover  
 lots of vulnerabilities of this kind.

**20.5.1** **The xor-to-null-key attack**

HSMs are driven by transactions sent to them by servers at a bank or ATMs in  
 the ﬁeld. The HSM contains a number of master keys that are kept in tamper-  
 responding memory. Most keys are stored outside the device, encrypted under  
 one or more master keys. It’s convenient to manage keys for ATMs and other  
 terminals in the databases used to manage them; and nowadays many HSMs  
 are located in the Azure and Amazon clouds where they serve multiple tenants.

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 615 | Ross Anderson |

*20.5. HSMS*

The encrypted working keys have a type system which classiﬁes them by

function. For example, in the PCI standard for security modules, a PIN deriva-  
 tion key – the master key used to derive a PIN from an account number as  
 described in section 12.4.1 – is stored encrypted under a particular pair of mas-  
 ter DES keys to mark it as a non-exportable working key. The *Terminal Master*  
 *Key* for an ATM is of the same type, and you’ll recall from section 12.4.1 that  
 ATM security policy is dual control, so the bank generates separate keys for  
 two ATM custodians, say the branch manager and the branch accountant, who  
 enter them at a keypad when the device is commissioned, or following a service  
 visit. The HSM thus has a transaction to generate a key component and print  
 it out on an attached security printer. It also returns its encrypted value to the  
 calling program. There was another transaction that combines two components  
 to produce the terminal master key: given two encrypted keys, it would decrypt  
 them, exclusive-or them together, and return the result – encrypted in such a  
 way as to mark it as a non-exportable working key.

The attack was to combine a key with itself, yielding a known key – the

key of all zeros – marked as a non-exportable working key. As there was a

further transaction, which would encrypt any non-exportable working key with  
 any other, you were now home and dry. You could extract the crown jewels –  
 the PIN derivation key – by encrypting it with your all-zero key. You can now  
 decrypt the PIN derivation key and work out the PIN for any customer account.  
 The HSM has been defeated.

The above attack went undiscovered for years. The documentation did not

spell out what the various types of key in the device were supposed to do; non-  
 exportable working keys were just described as *‘keys supplied encrypted under*  
 *master keys 14 and 15’*, and the implications of a transaction to encrypt one such  
 key under another were not immediately obvious. In fact, the HSMs had simply  
 evolved from earlier, simpler designs as ATM networking was introduced in the  
 1980s and banks asked for lots more features so they could make heterogeneous  
 networks talk to each other.

So Mike Bond built a formal model of the key types used in the device

and immediately discovered another ﬂaw. You could supply the HSM with an  
 account number, pretend it’s a MAC key, and get it encrypted with the PIN  
 veriﬁcation key – which also gives you the customer PIN directly. Confused?  
 Initially everyone was – modern APIs are way too complicated for bugs to be  
 evident on casual inspection. Anyway, the full details are at [100]. The latest  
 HSMs have strong typing to make it easier to reason formally about keys.

**20.5.2** **Attacks using backwards compatibility and time-**  
 **memory tradeo↵s**

We worked with an HSM vendor, nCipher, who supplied us with samples of their  
 competitors’ products, so we could break them – not just to help their marketing,  
 but to enable them to migrate customer key material to their own products. The  
 top target at the time was the IBM product, the 4758 [951]. This was the only  
 device certiﬁed to FIPS 140-1 level 4; in e↵ect the US government had said it was  
 unbreakable. It turned out to be vulnerable to an attack exploiting backwards

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 616 | Ross Anderson |

*20.5. HSMS*

compatibility [279].

As DES became vulnerable to keysearch during the 1980s, banks started

migrating to two-key triple-DES: each block was encrypted with the left key,  
 decrypted with the right key and then encrypted with the left key once more.  
 This bright idea gave backward compatibility: if you set the left key equal to  
 the right key, the encryption reverts to single-DES. The 4758 stored left keys  
 and right keys separately, and encrypted them di↵erently, giving them di↵erent  
 types – but failed to bind together the two halves of a triple-DES key. You could  
 take the ‘left half’ of a single-DES key plus the ‘right half’ of another, put them  
 together into a true triple-DES key, and then use this to export other keys.

So all you had to do to break the 4758 was a single-DES keysearch. That’s

|  |
| --- |
| not too hard now, but was still a fair bit of work back in 2002. Fortunately there  was another vulnerability – a time-memory tradeo↵ attack. That generation of  HSMs had ‘check values’ for keys – one-way hashes of each key, calculated by  encrypting a string of zeroes. Suppose you want a single DES key of a speciﬁc  type. You precompute a table of (say) 240 keys and their hashes. You get the  HSM to generate keys of the desired type and output the hashes until you see  a hash that’s already in the table. This takes about 216 hashes, which takes  an hour or so [447]. The backwards-compatibility and time-memory tradeo↵  attacks are examples of an API attack on the HSM platform itself rather than  on the PCI PIN management app. |

**20.5.3** **Di↵erential protocol attacks**

The 4758 bugs got ﬁxed, and recent models of ATM o↵er public-key mechanisms  
 for automatic enrolment. But legacy key-management and PIN-management  
 mechanisms persist at the app layer, as it’s hard to change the architecture of a  
 distributed system with hundreds of vendors and thousands of banks. And there  
 was much more to come. The next wave of attacks on HSM APIs was initiated  
 by Jolyon Clulow in 2003; they perform active manipulation of the application  
 logic to leak information. Many HSMs support transactions tailored for speciﬁc  
 applications; the largest market segment is to support card payments, though  
 there are also HSMs for prepayment utility meters, for certiﬁcation authorities  
 and even for nuclear command and control.

Clulow’s ﬁrst attack exploited error messages [449]. I described in sec-

tion 12.4.2 how banks who just wrote a customer’s encrypted PIN to their bank  
 card got attacked, as a customer could change the account number to another  
 one and use their PIN to loot that account. In order to stop such attacks, Visa  
 introduced an optional PIN block format that exclusive-ors the PIN with the  
 account number before encrypting them. But if the wrong account number was  
 sent along with the PIN block, the HSM would decrypt it, xor in the account  
 number, and when the result was not a decimal number, it would return an error  
 message. So by sending a few dozen transactions to the HSM with a variety of  
 wrong account numbers, you could work out the PIN6. There are now special

6There are now four di↵erent PIN block formats for PIN transmission, three of which

include the PAN as well; and there’s a further format, the *PIN Veriﬁcation Value* (PVV),  
 which is a one-way encryption of the PIN and PAN that’s sent by banks to switches such as  
 VISA and Mastercard if they want the switch to do stand-in PIN veriﬁcation when their own

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 617 | Ross Anderson |

*20.5. HSMS*

PCI rules for HSMs on PIN translation [977]. Complexity opens up new attacks,  
 which need yet more complexity to patch them.

A further class of attacks was then found by Mike Bond and Piotr Zielinski.

Recall the method used by IBM (and most of the industry) to generate PINs,  
 as shown in Chapter 12, ﬁgure 12.3. The primary account number is encrypted  
 using the PIN veriﬁcation key, giving 16 hex digits. The ﬁrst four are converted  
 to decimal, and while most banks do this by taking the hex digits modulo 10,  
 not all do. HSM vendors parametrised the operation by having a *decimalisation*  
 *table*, of which the default is 0123456789012345, which just reduces the hex  
 output modulo 10. This was a big mistake.

If we set the decimalisation table to all zeros (i.e., 0000000000000000) then

the HSM will return a PIN of ’0000’, albeit in encrypted form. We then repeat  
 the call using the table 1000000000000000. If the encrypted result changes, we  
 know that the DES output contained a 0 in its ﬁrst four digits. Given a few dozen  
 queries, the PIN can be deduced. Attacks that compare repeated, but slightly  
 modiﬁed, runs of the same protocol, we call *di↵erential protocol analysis*. The  
 only real solution was to pay your HSM vendor extra for a machine with your  
 own bank’s decimalisation table hard-coded. That may cause more problems  
 when you want to move your bank to the cloud, and share HSMs maintained  
 by Amazon or Azure7.

At a philosophical level, this illustrates the di�culty of designing a robust

*secure multiparty computation* – a computation that uses secret information  
 from one party, but also some inputs that can be manipulated by a hostile  
 party [99]. Even in this extremely simple case, it’s so hard that you end up  
 having to abandon the IBM method of PIN generation, or at least nail down  
 its parameters so hard that you might as well not have made them tweakable in  
 the ﬁrst place.

At a practical level, it illustrates one of the main reasons APIs fail over time.

They get made more and more complex, to accommodate the needs of more and  
 more customers, until suddenly there’s an attack.

**20.5.4** **The EMV attack**

You’d have thought that after the initial wave of API attacks were published  
 in the early 2000s, HSM designers would have been more careful about adding  
 new transactions. However, just as security researchers and HSM vendors found  
 and ﬁxed bugs, the banking industry mandated new ones.

For example, an HSM feature ordered by EMVCo to support secure mes-

saging between a smartcard and a bank HSM introduced an exploitable vul-  
 nerability in all EMV compliant HSMs [22]. The goal was to enable a bank to  
 order any EMV card it had issued to change some parameter, such as a key,  
 the next time it did an online transaction. So EMVCo deﬁned a transaction  
 *Secure Messaging For Keys* whereby a server can command an HSM to encrypt

system is down.

7One vendor decreed that a table must have at least eight di↵erent values, with no

value occurring more than four times. But this doesn’t work: 0123456789012345, then

1123456789012345, and so on.

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 618 | Ross Anderson |

*20.5. HSMS*

a text message, followed by a key of a type for sharing with bank smartcards.  
 The encryption can be in CBC or ECB mode, and the text message can be of  
 variable length. The attack is to choose the message length so that just one byte  
 of the target key crosses the boundary of an encryption block. That byte can  
 then be determined by sending a series of messages that are one byte longer,  
 and where the extra byte cycles through all 256 possible values until the key  
 byte is found.

**20.5.5** **Hacking the HSMs in CAs and clouds**

The most recent HSM break, in 2019, was by Jean-Baptiste B´edrune and Gabriel  
 Campana, on a Gemalto HSM whose application supported the PKCS#11 stan-  
 dard for public-key cryptography so it could be used in certiﬁcation authorities  
 and as a TLS accelerator. (This standard is notoriously obscure and di�cult to  
 implement.) They got a software development kit for the HSM, which contained  
 an emulator for the device, and fuzzed it until they found several vulnerabili-  
 ties. They managed to patch the authentication function so they could login as  
 admin into the HSM and install tools that read out the keys [203]. This is just  
 one example of many where sophisticated cryptography was fatally undermined  
 by careless software engineering.

**20.5.6** **Managing HSM risks**

At one time or another, someone had found an attack on at least one version  
 of every security module on the market. The root cause, as so often in security  
 engineering, is featuritis. People make APIs more complex until they break.

Banks still have to use HSMs for compliance with PCI rules, but the crypto

keys in them are not protected by the tamper responding enclosures alone. The  
 conﬁguration management has to be tight and vendor software patches have to  
 be applied promptly, just like in other systems. But while most banks of any  
 size have people who understand software security and the patching lifecycle,  
 they are less likely to have serious HSM expertise.

Specialist ﬁrms o↵er HSM management systems, and we’ll have to see if

these get subsumed eventually by the big cloud service providers. Management  
 of cloud HSMs is still a work in progress, and products such as Microsoft Cloud  
 Key Vault allow keys to be moved back and forth between HSMs and enclaves  
 that o↵er similar functionality. Of course, if a PIN management app has in-  
 trinsic API vulnerabilities, these will be independent of whether it’s running on  
 a traditional on-premises HSM, an HSM in a cloud data centre, or an enclave.  
 Indeed, one selling point of the Microsoft o↵ering is ‘Removing the need for  
 in-house knowledge of Hardware Security Modules’ [1309].

With that warning, it’s time to look at enclaves.

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 619 | Ross Anderson |

*20.6. ENCLAVES*

**20.6** **Enclaves**

Enclaves are like HSMs in that they aim to provide a platform on which you  
 can do some computation securely on a machine operated by someone you don’t  
 entirely trust. Early attempts involved mechanisms for *digital rights manage-*  
 *ment* (DRM) which obfuscated code to make it hard to interfere with8, and were  
 followed by the ‘trusted computing’ initiative of the early 2000s. This proposed  
 an architecture in which CPUs would execute encrypted code, with the keys  
 stored in a separate Trusted Platform Module (TPM) chip. Arm duly produced  
 TrustZone in 2004, as I described in section 6.3.2.

TrustZone is typically implemented in the System-on-Chip (SoC) at the heart

of a modern Android phone, although its trust boundary is typically the whole  
 motherboard; enclave data may be available in clear on the bus and in DRAM  
 chips. The main application has been mobile phones, whose vendors wanted  
 mechanisms to protect the baseband against user tampering (for regulatory  
 reasons) and to enable the phone itself to be locked (so that mobile network  
 operators who subsidise phones could tie them to a contract). In neither case  
 are hardware attacks a real concern.

Could an enclave mechanism such as TrustZone be used to harden a phone-

banking system against the kind of attacks we discussed in section 12.7.4? At-  
 tempts were made to market it for this purpose, but even ﬁrms that write  
 banking apps were reluctant to adopt it. Up until 2015, it was a closed system,  
 and you could only run code in TrustZone if you had it signed by the OEM.  
 So a developer of a banking app who wanted a ‘more secure’ authentication  
 component would have to get that signed by Samsung for Samsung phones, by  
 Huawei for their products, and so on. What’s more, the code would be di↵erent  
 depending on which SoC the product used. Now it’s hard enough to make an  
 app run robustly on enough versions of Android without also having to cope  
 with multiple customised versions of TrustZone running on di↵erent SoC o↵er-  
 ings. It’s also hard to assess security claims that vendors make about closed  
 platforms. For the gory details, see Sandro Pinto and Nuno Santos [1529].

In 2015, Intel launched SGX, whose access-control aspects I discussed in

section 6.3.1. SGX enclaves have aimed at a more ambitious use case, namely  
 cloud computing. It’s become cheaper to run systems on services such as AWS,  
 Azure and Google: virtualisation lets resources be shared e�ciently, so the  
 costs of data centres, sysadmins and so on can be amortised over thousands of  
 customers. But this raises many questions. How can you be sure that sensitive  
 data isn’t leaked to other tenants of the cloud service, for example via technical  
 exploits of the hypervisor software? Such products have dozens of bugs patched  
 every year [479]. And what protection do you have against a nation state using  
 a warrant to get access to your data – in e↵ect a legal exploit of the hypervisor?  
 The cloud service providers themselves long for a technical mechanism that  
 would save them the trouble of dealing with such warrants. Because of these  
 concerns, the security perimeter of SGX is the boundary of the chip itself. Code  
 and data are encrypted as they leave the chip, and decrypted as they’re imported  
 into the cache. The CPU’s hardware protects both conﬁdentiality and integrity.

8For an introduction, see the chapter on ‘Copyright and DRM’ in the second edition of

this book, available free online.

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 620 | Ross Anderson |

*20.6. ENCLAVES*

The key cryptographic mechanism is *software attestation* which enables the

CPU to certify to the owner of the software that it is running without modi-  
 ﬁcation on top of trustworthy hardware. SGX enclaves run as applications, at  
 ring 3, and the CPU machinery isolates their code and data from everything  
 underneath, including both operating system and hypervisor9. The full details  
 of enclave initialisation, address translation, page eviction, exception handling  
 and so on are extremely complicated; for an explanation and analysis, see Vic-  
 tor Costan and Srini Devadas [479]. One concern they raise is that with the  
 exception of memory encryption, SGX is implemented in microcode, which can  
 be updated; the whole system is therefore changeable. There are also multi-  
 ple side-channel attacks, particularly since Meltdown and Spectre introduced  
 the transient execution family of side-channel attacks, which I discussed in sec-  
 tion 19.4.5. Some have been patched, but the real scandal may be that Intel  
 has said it won’t ﬁx the Membuster attack as a matter of policy10.

Here my concern is the cryptography used to support the enclave and attest

to the software running on it, and its suitability as a platform for other crypto  
 or crypto support for applications.

As the silicon processes used in high-end CPUs don’t support nonvolatile

memory, the ﬁrst problem is to provide unique and persistent chip keys. Each  
 chip has fuses into which the fab burns a seal secret and a provisioning secret,  
 of which the former is not known to Intel but the latter is. This is used to  
 generate the master derivation key (MDK) which in turn generates key material  
 dependably across power cycles. Provisioning seal keys are persistent, so when  
 a computer changes owners, Intel doesn’t need to know. These keys enable the  
 CPU to prove its authenticity to Intel which supplies it with an attestation key –  
 a member private key in Intel’s *Enhanced Privacy ID* (EPID), a group signature  
 scheme intended to preserve signer anonymity.

These operations are done in a privileged *launch enclave* (LE). Originally all

SGX code had to be signed by Intel, but recent versions allow code signed by  
 third parties. Each enclave author is now a CA and certiﬁes each enclave, which  
 has a public key, a product ID and a version number (migration of secrets is  
 allowed only to higher version numbers to support patching but not rollback).  
 The same ratchet applies to updates of the CPU microcode.

One issue is that the compromise of one chip’s MDK – in any CPU, anywhere

– breaks the attestation security of every CPU in the same group. This happened  
 in 2019 for AMD’s equivalent of SGX, when a bug in the microcode enabled  
 such a key to be extracted [337]. Intel is vulnerable in the same way: given a  
 clear value of MDK you can create an SGX enclave outside of SGX’s protection  
 mechanisms. If such a break were discovered, Intel would have to blacklist all  
 the CPUs in the same EPID group. We have no idea how large these groups  
 are, as all attestations are done opaquely by Intel and users must simply trust  
 the results.

9The earlier proposals of the Trusted Computing Group required that the whole software

stack underneath the enclave be attested and trustworthy, which is incompatible with an  
 untrusted hypervisor.

10SGX doesn’t defend against cache timing attacks, so when writing enclave code, you can’t

use data-dependent jumps. More generally, it does not protect against software side-channel  
 attacks that rely on performance counters, but doesn’t give enough information for developers  
 to model the possible leakage.

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 621 | Ross Anderson |

*20.6. ENCLAVES*

There are now some SGX systems doing real work. An example I mentioned

earlier in this chapter is the messaging app Signal, which uses an enclave for  
 private contact discovery. Its developers published the source code along with an  
 extensive discussion of the di�culties of developing it on the Signal blog [1226].  
 The goal is to enable Signal clients to determine whether the contacts in their  
 address book are also Signal users without revealing their address book to the  
 Signal service. How can you build a large social graph without having any insight  
 into it? The idea is that clients can contact the enclave, verify it’s running the  
 right software, and send their contacts in to see who’s also a user. However,  
 doing this within the memory limit of an SGX enclave (128Mb) needs careful  
 organisation of hash tables of an inverted ﬁle of users’ phone numbers.

There are many more things you have to do to prevent information leakage

through memory access patterns: as branches might be observed through such  
 patterns, critical sections of code must not contain branches. In short, blocking  
 side channels is much like organising crypto code to run in constant time: ﬁddly,  
 ad hoc, manual and prone to error. SGX is also slow: while the memory encryp-  
 tion itself adds little overhead, context switching is a killer. Checking contacts  
 against others is really slow, so the process has to be batched for multiple joiners  
 to make it acceptable.

Another example of an SGX app is Microsoft’s Cloud Key Vault, which

enables Azure tenants to store secrets such as keys, passwords and tokens sep-  
 arately from their code [1309]. There’s an app to help you create and manage  
 certiﬁcates for TLS; secrets and keys can also be stored in cloud HSMs at the top  
 end, while routine applications can be both more secure and more manageable  
 if you don’t have to store database passwords inline in your code.

In short, writing good SGX code is hard. The toolchain is restricted, and

things like antivirus are excluded. If you’re smart, you can write trusted mal-  
 ware. You can even write malware that will run in one SGX enclave and do  
 timing attacks on code in other enclaves in the same machine, using the SGX  
 mechanisms to hide itself from detection [1689].

And even if you trust Intel completely; even if you believe that the NSA

won’t use a FISA warrant to force Intel to attest to an enclave in debug mode;  
 even if you’re not worried about an MDK compromise or the exploitation of side  
 channels – then there’s still the risk of app-layer exposure, just as with HSMs.  
 If you write your enclave code in such a way that it can be used as an oracle by  
 less trusted code, you’re in trouble.

Intel (and Arm) are talking about successor versions of their enclave tech-

nology. Meantime Intel points crypto developers at their management engine  
 (ME), a separate microcontroller shipped in the CPU chipset that starts the  
 CPU and contains a ﬁrmware TPM to do secure boot. It can brick a CPU by  
 erasing keys if the machine is reported stolen. Its code is proprietary, based  
 on Minix, and is signed by Intel. It supports yet another enclave with a Java  
 trusted execution environment, in which developers can do crypto; for example,  
 in payment terminals you can engineer a hardware trusted path from the ME  
 to a PIN pad [1698]. This enables crypto code to be shielded from malware on  
 the CPU but brings issues of its own, such as attacks involving physical access.  
 The ME has also had a whole series of vulnerabilities and exploits. It is consid-

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 622 | Ross Anderson |

*20.7. BLOCKCHAINS*

ered by the EFF to be a backdoor, and at least one vendor has made machines  
 available to governments where it is switched o↵ after boot.

**20.7** **Blockchains**

The previous sections on the uses and limits of cryptography, on how cryptog-  
 raphy can be used to support anonymity, and how crypto apps can su↵er ﬂaws  
 at various levels in the stack, set us up to discuss cryptocurrencies and smart  
 contracts. During 2016–7, cryptocurrencies were ‘the’ thing, taking their place  
 in the hype cycle after Big Data and the Internet of Things, alongside AI and  
 quantum. To many people, the word ‘crypto’ now refers to bitcoins rather than  
 to ciphers.

In 2008, Bitcoin was released quietly by someone using the pseudonym of

Satoshi Nakamoto, with a white paper and an implementation [1375]. This

system of anonymous digital cash circulated initially among hobbyists and ac-  
 tivists on the cypherpunks mailing list, but within two years it had gone viral.  
 In February 2011, a young libertarian called Ross Ulbricht set up Silk Road,  
 an online marketplace outside government control. Buyers and sellers met on  
 a Tor onion service and could pay for goods and services using Bitcoin. They  
 could rate each other, as on eBay, and there was an escrow service so that a  
 buyer could deposit bitcoins for release when goods were delivered. Silk Road  
 rapidly became the market for the mail-order supply of controlled drugs, and  
 over $1bn worth of trades went through it before the FBI arrested Ulbricht in  
 October 2013 [421]. Other underground markets adopted Bitcoin too. While  
 Silk Road was trading, the price had risen from about a dollar to over a hundred  
 dollars, and the rising price attracted investors11. Further transaction demand  
 came from people wanting to get their money out of countries with exchange  
 controls, leading to investment demand from people seeing Bitcoin as an asset  
 to be bought in times of crisis, like gold. By 2017 we had a bubble – with the  
 price of a bitcoin rising steeply through the thousand-dollar mark to a peak in  
 December 2017 of almost $20k.

Bitcoin has spawned multiple imitators – most of them scams, but some real

innovations too. Boosters claimed that cryptocurrency would enable a new wave  
 of innovation and automation as machines could negotiate smart contracts with  
 each other without humans or banks getting in the way. At the time of writing  
 (2020), the peak of enthusiasm has passed, but cryptocurrencies have become  
 a new asset class for investors, as well as posing multiple problems for ﬁnancial  
 regulators and law enforcement.

All that said, Bitcoin is a fascinating construct of cryptography and eco-

nomics which has led to the emergence of a payment system that is also a  
 trusted computer, out of the distributed e↵ort of millions of machines that at-  
 tempt to mine bitcoins. There are no trusted parties other than the people  
 who write the software, and no pre-assumed identities of participants. The

mechanisms provide a new way of achieving consensus in distributed systems,  
 quite distinct from the Byzantine fault-tolerance mechanisms we discussed in

11When Ulbricht was busted, the Bitcoin price fell from $145.70 to $109.76, but as other

drug markets got going, it quickly recovered.

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 623 | Ross Anderson |

*20.7. BLOCKCHAINS*

section 7.3.1. That is one reason to include cryptocurrencies as an example of  
 advanced cryptographic engineering; another is the smart contracts and other  
 second-layer protocols built on top of them, which are of technical interest al-  
 though they have had little impact so far on business (the total capital of digital  
 exchanges may be only about $1bn).

Here is a brief summary of the basic mechanisms.

1. The Bitcoin *blockchain* is an append-only ﬁle containing a series of *trans-*

*actions*.

2. Users appear on the blockchain as addresses – pseudonyms which are

hashes of public keys.

3. Most transactions transfer currency from one address to another by taking

an *unspent transaction output* (UTXO) from a previous transaction and  
 transferring it to one or more addresses. Such a transaction must be signed  
 by the private key corresponding to the UTXO address.

4. To make a payment, you sign a transaction and broadcast it via a peer-

to-peer network to other users. Other users are free to select a set of

requested transactions, check that they’re valid, and mine them into a  
 new block for the blockchain.

5. Each block of transactions is authenticated by a miner by means of a

SHA256 hash of the block contents and a random salt. Miners try di↵erent  
 salts until the hash output has enough leading zeros to make it a hard  
 enough puzzle. Such a hash constitutes a *proof of work*, and ﬁnding them  
 is a random process, so it’s hard to predict which miner will ﬁnd the next  
 one. The blockchain consists of a chain of hashes and the blocks they  
 authenticate. The di�culty of the puzzle is adjusted automatically so

that a new block is *mined* about every ten minutes.

6. Miners are paid a *block reward* for each block they mine; at the time of

writing, this is 12.5 bitcoins, or over $100,00012.

7. Miners also get *transaction fees*, which are the amount by which the inputs

of each transaction exceed the outputs. Users bid transaction fees to get  
 priority for their transactions; they are usually in the tens of cents but  
 can rise into the tens of dollars at times of congestion.

8. If two competing next blocks are mined then the conﬂict is resolved by the

rule that miners mine the longest chain. As a result, transactions aren’t  
 really considered ﬁnal until about half a dozen further blocks have been  
 mined – about an hour for classic Bitcoin. Even so, a majority of miners  
 could rewrite history by constructing a chain that reached even further  
 back – a so-called *chain reorganisation*.

12In early 2020 a miner who could buy electricity for 5c per kWh could expect to mine

Bitcoin worth about half what the coins would fetch on the market, if you disregard the costs  
 of the equipment. However the reward halves from time to time to limit the total supply of  
 bitcoin, and the reward is due to drop to 6.25 bitcoin in mid-2020. People investing in mining  
 rigs are therefore gambling that the Bitcoin price will rise and that regulators will not be  
 e↵ective in suppressing demand.

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 624 | Ross Anderson |

*20.7. BLOCKCHAINS*

9. If the conﬂict isn’t resolved then you can end up with a *fork* – the system

spawns two incompatible successors. Bitcoin split in 2017 into Bitcoin  
 and Bitcoin Cash over a policy dispute about block length, and users who  
 owned bitcoins before the fork ended up owning bitcoins in both. But  
 some forks have been deliberate, and on top of that entrepreneurs have  
 started several thousand Bitcoin clones – most of which were scams.

10. Transactions can also contain scripts, which make payments programmable.

For a detailed description, there are three standard references. The ﬁrst

two are technical expositions by a group of Princeton computer scientists: an  
 18-page systematisation-of-knowledge paper in 2015 by Joe Bonneau, Andrew  
 Miller, Jeremy Clark, Arvind Narayanan, Joshua Kroll and Ed Felten [293] while  
 at 308 pages there’s a 2016 book by Arvind Narayanan, Joe Bonneau, Ed Felten,  
 Andrew Miller and Steven Goldfeder [1383]. The third is a 2015 paper in the  
 Journal of Economic Perspectives by Rainer B¨ohme, Nicolas Christin, Benjamin  
 Edelman, and Tyler Moore [274]. At the time of writing, these are getting out  
 of date, so in what follows I will concentrate on developments since then. I’ll  
 assume you know the detail, or can look it up, or are not too bothered.

To understand what can go wrong with cryptocurrencies, we have to look at

a lot more than just the cryptomathematics. A common pattern has been that  
 elegant cryptographic ideas are let down by shoddy software engineering, a lack  
 of systems thinking and a near-total lack of concern for users.

**20.7.1** **Wallets**

In the beginning, all Bitcoin users were peers: the full client software would  
 mine Bitcoin and let you spend the coins you mined. But things soon started to  
 specialise with custom rigs for miners, and light clients for ordinary users which  
 don’t do mining or store the whole blockchain, but make the process of buying  
 and selling more manageable. There is no intrinsic concept of an account, as  
 you own Bitcoin by knowing a private key that will unlock one or more UTXOs.  
 *Wallets* initially stored one or more private keys and provided an interface so the  
 user could see the UTXOs that these keys could spend (‘my bitcoins’). Wallet  
 security rapidly became a big deal. So-called ‘brain wallets’ which generated  
 private keys from a user-selected passphrase were broken by attackers doing  
 exhaustive search over the public keys visible on the blockchain; brain wallets  
 with guessable passwords were typically emptied within 24 hours [1947].

Software wallets that keep your signing keys on your hard disk, protected by

a passphrase, are an improvement, but vulnerable to malware and other attacks.  
 Serious operators use hardware wallets, which are essentially small HSMs and  
 which may be kept o✏ine (so-called *cold wallets*). Even so it is not unknown  
 for people who are known to own millions of dollars worth of Bitcoin to be held  
 up by armed robbers in their homes and forced to transfer it. If you have sole  
 physical custody of a Bitcoin wallet then you’re just as vulnerable as when,  
 centuries ago, people kept their savings in gold coins. By 2013 we’d seen the  
 emergence of *hosted wallets* where an exchange or other online service provider  
 does everything for you. That doesn’t really solve the robbery problem, as the

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 625 | Ross Anderson |

*20.7. BLOCKCHAINS*

robber will just force you to log on and pay him. But hosted wallets have led  
 to widespread other fraud and abuse as I’ll describe below.

**20.7.2** **Miners**

As bitcoins grew in popularity and value, more people joined in to mine them.  
 Mining rigs appeared using FPGAs and then ASICs that run so much faster  
 than software on general-purpose machines that within a few years they had  
 taken over. Miners operate where electricity is naturally cheap, such as Iceland  
 and Quebec, but are mostly in places like Russia or China where they can  
 do deals with local o�cials. The total energy consumption of cryptocurrency  
 mining during 2019 was about 75TWh, and the CO2 emissions were over 35Mt  
 – comparable to the carbon footprint of New Zealand. As of 2020, each bitcoin  
 transaction consumes over half a MWh and emits over a quarter ton of CO2.

Miners have organised themselves into a small number of mining pools that

average their earnings. The control of these pools is opaque. Capacity can be  
 rented and is sometimes used to attack cryptocurrencies in so-called 51% attacks.  
 The whole point of the blockchain is to prevent double spending by creating  
 a tamper-proof, public, append-only log of transactions; but if a majority of  
 miners collude then they can rewrite history and spend coins multiple times. In  
 the early days, people thought that such an attack would be instantly fatal to a  
 currency’s credibility, but reality turned out to be more complex. For example,  
 in January 2019, attackers used this technique to steal over $1m from Ethereum  
 Classic, a cryptocurrency with a market capitalisation of over $500m, with chain  
 reorganisations dozens of blocks in length [1428]. Yet its market value was not  
 signiﬁcantly a↵ected. Had they stolen most of it, the price would have collapsed  
 and their loot would have been worthless. There were two furthers attack in  
 August 2020, in one of which the attackers spent $192,000 to buy the hash power  
 required to steal $5.6m [1519]. So we need to think carefully about the game  
 theory as well as the cryptography when reasoning blockchains; the simplistic  
 arguments don’t always align with reality.

**20.7.3** **Smart contracts**

The scripting language in Bitcoin is simple, but a later cryptocurrency system,  
 Ethereum, has a Turing-complete VM whose bytecode is usually compiled from  
 a language called *Solidity*. Ethereum has become the second cryptocurrency by  
 market cap as it holds out the prospect of *smart contracts* that can perform  
 complex transactions automatically. During the bubble, many startups talked  
 of using smart contracts to animate the Internet of Things, and to create new  
 services such as distributed storage, where people might pay others for the use  
 of their spare hard disk space for backup. The idea of such a *distributed au-*  
 *tonomous organisation* was heavily promoted during the bubble. This is linked  
 to the ‘redecentralize’ movement which seeks to move the online world away  
 from the large service ﬁrms that came to dominate it during the 2000s; and  
 while we have good tools to decentralize the distribution of static, read-only  
 content, we lacked a good way to decentralize transactions [509]. As of 2020,  
 the main applications seem to be around trading, where distributed exchanges

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 626 | Ross Anderson |

*20.7. BLOCKCHAINS*

(DEXs) enable people to trade one cryptocurrency for another without human  
 intervention. (They still account for only a tiny fraction of the total trading  
 volume.)

This has led to interesting new failure modes. Although the consensus mech-

anisms of the original Bitcoin blockchain are believed to be incentive compati-  
 ble, this is not the case when the transactions on a blockchain represent extra  
 value that a miner can extract by manipulating the consensus. There have

now appeared *arbitrage bots* that exploit ine�ciencies in DEXs by frontrunning  
 (anticipating and exploiting) trades. The bots bid up transaction fees, called  
 *gas* in Ethereum; there have been hundreds of millions of these *priority gas*  
 *auctions* where traders hustle to get priority for their trades [508]. Bots might  
 in theory take over the governance of a market and loot it if they could raise  
 enough money [869]; they already make large proﬁts by exploiting bugs in smart  
 contracts [1507].

Fixing bugs can be expensive. In 2016, an investment fund called DAO

was set up as a smart contract on the Ethereum blockchain, and attracted over  
 $150m from over 10,000 investors. Attackers exploited a ﬂaw in the contract to  
 steal the money13, and after some discussion the Ethereum software was changed  
 to move the stolen money to a recovery account. This resulted in a hard fork  
 of the blockchain, with holders of the original cryptocurrency acquiring units in  
 both the modiﬁed currency and in ‘Ethereum Classic’, as the unmodiﬁed version  
 became known.

A Danish study illustrates the further problems of using smart contracts in

a real-world application context. There had been a proposal to use them to  
 pay parents who have to take time o↵ work to care for sick children, which has  
 complex legal rules that clerks often miss, leading to appeals. The idea was  
 to put hashes of the case documents on the Ethereum blockchain so that both  
 parents and the appeals board can track them, in the hope that automating the  
 execution of decisions would cut bureaucratic foot-dragging. But what about  
 insiders, hackers and mistakes? Local governments tend to get hacked a lot  
 and end up paying ransomware. And who updates the contract when the law  
 changes, or a bug is discovered? Blockchains are by design immutable, so can’t  
 be patched. But the real deal-breaker was local government fear of losing control  
 of the process. Two further issues include the fact that people often have to bend  
 the rules to get stu↵ done, and that programmers are more likely to write bugs  
 in an unfamiliar language such as Solidity rather than a familiar one such as  
 Python or even Cobol – a known problem with new languages, which I discussed  
 in section 7.3.1.2.

**20.7.4** **O↵-chain payment mechanisms**

A standard Bitcoin transaction can take six blocks, or one hour, to become  
 ﬁnal, and even longer at times of congestion. This may be fast enough for

paying ransoms or buying drugs online, but it’s unimpressive compared with  
 EMV. What’s more, Bitcoin’s throughput of about 5 transactions per second is

13An alternative view is that if the contract was to accept the output of the code, then the

ﬂaw was in the users’ grasp of what the code did, and in that case nobody stole anything!

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 627 | Ross Anderson |

*20.7. BLOCKCHAINS*

no match for Visa’s 50,000.

People are trying to ﬁx this using side chains, an example of a *layer 2 protocol*;

such protocols do transactions outside, but tethered to, a layer 1 protocol such  
 as Bitcoin or Ethereum. Alice and Bob open a channel by locking coins on  
 a layer 1 blockchain, and can now do rapid transactions between themselves.  
 The key idea is that they commit some cryptocurrency to each other using a  
 *hashed time-lock contract* (HTLC) made of two conditional transfers. In such a  
 transfer, Bob sends Alice *h*(*R*), where *R* is a random number, and Alice makes  
 a commitment in the blockchain’s scripting language to the e↵ect that “if you  
 show me *R* by time *t* I’ll give you this coin.” Bob makes a similar commitment.  
 This opens a channel for them to trade signed transactions at speed, until they  
 decide to settle up and close the channel.

Quite a bit more engineering is needed to turn this into a working payment

system. You need a dispute resolution mechanism in case Alice and Bob dis-  
 agree how much each of them should take from the proceeds. Then you build  
 mechanisms for Alice to pay Charlie via Bob, and routing algorithms so you can  
 get money to anybody. In theory this can be peer-to-peer but in practice such  
 systems appear to organise themselves into hubs, with channels that are always  
 open, like a banking network. Protocol security involves ensuring that honest  
 users must not lose money even if others collude. Costs include the need for  
 intermediate nodes to have enough liquidity to forward transactions, and the  
 need for all active players to be online – whose implications range from the theft  
 risks of hot wallets, to the risk of miners front-running Bob when he broadcasts  
 *R*, to the risk of mass collapse following a network failure [831]. The leading  
 such system in 2020 is the Lightning network, which makes payments ﬁnal in  
 seconds, enables people with the right phone app to pay to a QR code as with  
 WeChat Pay, and is now handling 1000 transactions per day. The limit here ap-  
 pears to be liquidity: although Lightning chains themselves are trust-free, they  
 tie up capacity at the nodes, and the recipient has to decide whether or not to  
 accept them. So a malicious user can set up hundreds of payments, leave them  
 for hours and then cancel them at no cost. As Lightning’s total capitalisation  
 appears to be only a few million dollars, this may leave it somewhat fragile. It  
 also appears very possible that regulators will crack down on forwarding nodes.

**20.7.5** **Exchanges, cryptocrime and regulation**

Mining all your own coins is inconvenient, and by 2010 entrepreneurs had set up  
 exchanges that would trade Bitcoin for conventional money. Most went bust,  
 often because they were hacked, or because insiders stole the money and claimed  
 to have been hacked. The leader by 2011 was Mt Gox in Japan which survived  
 one hack in 2011 but went bust in 2014 claiming that it had been hacked for  
 $460m. The court case continues; news coverage at the time reported that

internal controls and software development processes were chaotic [1280].

That was not all. One of Mt Gox’s innovations was to become a *custodial ex-*

*change* over the course of 2013. Instead of keeping customer bitcoins in separate  
 wallets, for which the exchange might or might not have temporary access to  
 the private key after the customer entered the correct password, Mt Gox started  
 to keep all the Bitcoin in its own wallets, showing customers a notional account

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 628 | Ross Anderson |

*20.7. BLOCKCHAINS*

balance when they visited its website. It had made the transition we saw in  
 eighteenth-century ﬁnance from being a gold merchant to being a bank: rather  
 than owning a speciﬁc bag of gold coins in the vault, the customer now just  
 had a claim on the bank’s whole assets. Victims related how after their wallets  
 were hosted, they started to see outgoing transactions they had not authorised.  
 Analysis after the collapse of Mt Gox revealed that many of these transactions  
 did not even appear on the blockchain. From mid-2013, when you bought a  
 bitcoin from them, all they did was to show you a web page saying that you had  
 a balance of one bitcoin. (And that’s how many exchanges work to this day.)

The Bitcoin world has been full of scams, and it looks like the majority

of victims of cryptocrime were ripped o↵ by exchanges that went bust, or got  
 hacked, or that claimed to have been hacked. Even in the ﬁrst three years that  
 exchanges existed, 2010–13, 18 of the 40 exchanges collapsed [1339].

A report by Chainalysis, a Bitcoin analytics ﬁrm, concluded that exchanges

lost about $1bn to hackers in 2018, with most of the thefts perpetrated by two  
 crime gangs; one of them has since been linked to North Korea. In addition  
 to this, turnover on underground markets where drugs and other illicit goods  
 are bought and sold was $600m, approximately double the value for 2017 [400].  
 There’s also market manipulation. John Gri�n and Amin Shams present evi-  
 dence that Bitcoin’s price was supported by insider trading involving Tether, a  
 digital currency pegged to the U.S. dollar, during the 2017 boom [822], raising  
 the prospect that the market price of many cryptocurrencies may often have  
 been a result of unlawful manipulation. This has been borne out by subsequent  
 studies showing that much of the spot trading is generated by unregulated ex-  
 changes [1615].

Market manipulation aside, the largest single cryptocurrency scam to date

appears to have been a Ponzi scheme called PlusToken, which netted some $3bn  
 from Chinese nationals before the organisers were arrested in 2019 [864]. But  
 Bitcoin has a↵ected many other crime types too. Ransomware went up from  
 about $2–3m a year to maybe $8m a year between 2001 and 2015, as Bitcoin  
 suddenly made ransoms easy to collect [91]; this crime type is growing steadily,  
 although ransoms are also collected via gift cards [1190]. By 2018, bulletproof  
 hosting sites, which provide services to cybercriminals, were moving to cryp-  
 tocurrency as other payment mechanisms became more di�cult [1452]. In that  
 year, the world’s largest darknet child pornography website, Welcome to Video,  
 was closed down after its operators were traced via ﬂows of Bitcoin on the  
 blockchain, so the pseudonymous nature of cryptocurrency has its limits [551].  
 In total, scams and other abuse add up to something like 3% of cryptocurrency  
 transaction volume directly; and in addition to the visible cryptocurrency ex-  
 changes, there are a number of over-the-counter brokers, some 100 of which have  
 been identiﬁed as involved in money laundering [401]. The regular exchanges  
 also make life di�cult for law enforcement. Crime gangs may turn proceeds into  
 Bitcoin through one channel, switch it into a di↵erent coin in a second country,  
 and then send it to a third country where they get it out via bank transfer.

However, although Bitcoin uses pseudonyms, the blockchain contains a per-

manent record of all transactions. As we’ve discussed in a number of contexts  
 – from our chapter on inference control to the section on Tor in this chapter –  
 anonymity is hard. Real-world transactions and data have context and allow

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 629 | Ross Anderson |

*20.7. BLOCKCHAINS*

inferences to be made. Bitcoin users have tried all sorts of tricks to make trans-  
 actions more anonymous, for example by splitting payments into many smaller  
 ones, mixing them up, and then recombining them – a so-called ‘tumbler’ or  
 ‘mixer’. However, if you do that, you taint your bitcoins with attempted money  
 laundering; and in total, perhaps 10% of Bitcoin have been stolen, or passed  
 though a money-laundering service, at least once. (For an analysis, see [116].)  
 As an example, an Ohio man was indicted in 2020 for operating just such a  
 mixer that laundered $300m [553]. There are also cryptocurrencies that o↵er  
 more privacy using further cryptographic techniques, notably Zcash and Mon-  
 ero. At present, Monero o↵ers the strongest privacy and is designed so that  
 coins can be mined using software; over 4% of its coins have been mined by  
 malware running on other people’s machines [1529].

Governments have been trying to push back using ﬁnancial regulation. The

US Treasury’s Financial Crimes Enforcement Network (FinCEN) drives anti-  
 money-laundering (AML) and know-your-customer (KYC) regulations world-  
 wide, which get incorporated into local law, for example via the EU’s 5th Anti-  
 money-laundering Directive. Some governments go further. For example, Ger-  
 many’s regulator BaFin has used existing ﬁnancial regulations to insist that all  
 exchanges get licenses; as localbitcoins.com, a peer-to-peer exchange that  
 enables individuals to buy and sell cryptocurrency from each other for cash,  
 didn’t apply for one, it is blocked there. But at the time of writing, the biggest  
 push comes from a FinCEN advisory in 2019 that required cryptocurrency ex-  
 changes to implement the ‘travel rule’ whereby anyone handling a transaction  
 over $10,000 has to identify both sender and recipient and ﬁle a suspicious ac-  
 tivity report if relevant. The exchanges were given until June 2020 to come up  
 with a solution; at least one individual exchanging sums over $10,000 has been  
 ﬁned [688].

Further regulation is on the agenda in Europe too. Mt Gox largely had

Japanese clients while most Chinese appear to use Binance and many people  
 in the UK and the USA use Coinbase. When one British or American user  
 sends Bitcoin to another, there’s a fair chance that the transaction never goes  
 near the blockchain: if they’re both Coinbase customers, then Coinbase can  
 simply adjust the balances displayed in their Bitcoin wallet webpages. This

immediately raises the question of why the exchanges are not regulated like  
 any other money service business. In the EU, the E-money Directive might

seem to apply, yet regulators in the UK and Germany only enforce it in respect  
 of the traditional currency balances that customers have with the exchanges;  
 the exchanges argued that as transaction demand is much less than investment  
 demand, virtual currencies should be treated as assets rather than as payment  
 mechanisms. But in that case, why does the regulator not require the exchanges  
 to operate under the same rules as stockbrokers, so that a customer’s bitcoins  
 can’t be used for transactions, but merely sold back to market with the proceeds  
 being sent to the bank account used to purchase them?

In an analysis that colleagues and I produced of exchange operations and

of the mechanics of tracing stolen Bitcoin, we also recommended applying the  
 Payment Services Directive, which would give exchange customers consumer  
 protection comparable to that with banks [116]. It is notable, for example, that  
 while banks have shown a lot of interest in how to block SIM swap attacks on

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 630 | Ross Anderson |

*20.8. CRYPTO DREAMS THAT FAILED*

their customers’ phones, most cryptocurrency exchanges have shown no interest  
 at all – despite the fact that exchange credentials are one of the main targets of  
 the SIM swap gangs [1449]. Consumer protection in the world of cryptocurrency  
 is unﬁnished business, and regulatory agencies in Europe and elsewhere are  
 working on it.

**20.7.6** **Permissioned blockchains**

The hype around cryptocurrencies and blockchains piqued commercial interest,  
 and from about 2015, CEOs coming back from Davos told their IT departments  
 they needed a blockchain. The CIOs then had to explore whether blockchains  
 could be created that could do useful work, without Bitcoin’s environmental  
 waste, illegal content and illegal actors. This led to initiatives such as Hyper-  
 ledger and the Enterprise Ethereum Alliance, with corporate supporters devel-  
 oping a variety of blockchain tools and standards. Many involve a *permissioned*  
 *blockchain* fabric that is based on Byzantine fault tolerance rather than proof-  
 of-work and can still support smart contracts. A number of them use SGX as  
 part of their consensus mechanism, such as Intel’s own proof of elapsed time  
 (PoET) proposal. There are many other proposed consensus mechanisms; for a  
 survey, see Bano et al [165].

As an application example, JP Morgan worked on a system from 2015 that

would enable participating banks to enter mortgages on a blockchain, so that  
 its scripting language would allow traders to create futures and options of arbi-  
 trary complexity. They explored a number of design tradeo↵s, such as between  
 low latency and security in adversarial settings, and how transaction privacy  
 can be extended to keep business logic private as well as the names of indi-  
 vidual participants [1421]. One conclusion was that for the vast majority of  
 applications, you don’t need a blockchain; a forward-secure sealed log will do.  
 And where a blockchain might help, you can’t use a public one. Above all,  
 blockchain apps must talk to legacy systems and must be no more likely to cre-  
 ate application security mistakes or usability hazards. There have been enough  
 screw-ups: for example, Argentina published its o�cial gazette (Boletin O�cial)  
 on a blockchain, and decreed it to be legally valid, whereupon someone hacked  
 it to publish fake news about the coronavirus [499]. Such real-world experience  
 appears to be taming the initial exuberance of the bubble.

Perhaps the most controversial project is Libra, a Facebook proposal to

create a payment system with its value pegged against a basket of currencies.  
 This was supposed to be run by a consortium of ﬁnancial, tech and other ﬁrms,  
 but has run into signiﬁcant opposition from central banks, resulting in key  
 ﬁnancial players such as Visa, MasterCard and PayPal pulling out.

**20.8** **Crypto dreams that failed**

A number of people have proposed electronic voting systems based on blockchains  
 because they’re supposedly immutable and you can build functionality on them  
 using crypto. These proposals follow over thirty years of research into the pos-  
 sible use of cryptography in electronic elections to provide a system that is

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 631 | Ross Anderson |

*20.9. SUMMARY*

simultaneously anonymous and provably accurate. In fact, during the Bitcoin  
 boom of 2017–8, a common student project proposal was ‘solving world peace  
 by putting elections on the blockchain’.

Election systems claiming to use a blockchain have now been deployed in

both Russia and America, with less than impressive results. In 2018 a system  
 for three wards in the city of Moscow used an Ethereum blockchain for vote  
 tallying, but the link between vote tallying and the blockchain was broken when  
 two crypto vulnerabilities were ﬁxed just before the election – and the blockchain  
 vanished just afterwards [782]. Also in 2018, West Virginia became the ﬁrst US  
 state to allow some voters to cast their ballot using a mobile phone app. Michael  
 Specter, James Koppel and Danny Weitzner from MIT reverse engineered it and  
 found a number of vulnerabilities that would let an attacker expose or alter votes,  
 despite the app’s use of a blockchain, which was irrelevant to the attacks [1810].  
 According to the researchers, an attacker could create a tainted paper trail,  
 making a reliable audit impossible – despite the selling points of blockchains  
 including transparency and accountability.

The idea that blockchains can solve the problems of elections makes the expe-

rienced security engineer despair. You can’t ﬁx elections with this technology,  
 because it doesn’t tackle how they’re stolen. Parties in power are constantly  
 changing the rules and subverting the technology at all levels in the stack, from  
 voter registration through campaign funding and advertising rules through me-  
 dia censorship, voter intimidation and voting schemes that can be manipulated.  
 We’ll discuss this at greater length in section 25.5.

**20.9** **Summary**

Starting in the 1980s, many people have tried to use cryptography as a trusted  
 platform for some aspect of system security. The original killer app for com-  
 mercial cryptography was the protection of PINs in ATMs and then of card  
 payments more generally, as we described in Chapter 12. Many cryptography  
 researchers (including me) then started to hope that we could solve other eco-  
 nomic and social problems with cryptography. Anonymous communications

would stop censorship; anonymous digital cash would protect our privacy; digi-  
 tal voting would make elections harder to rig; threshold signatures would help us  
 build robust internal control systems; and electronic auctions would push back  
 on corruption. The research papers at the Crypto and Eurocrypt conferences of  
 the period are brimming with ideas like these. A generation later, and with a  
 techlash of scepticism about the e↵ects of globalised technology, it may be time  
 to take stock.

Our case studies teach a technical point, an economic one, and a policy one.

The technical point is that cryptographic systems aren’t magic; they have

bugs and have to be patched like anything else. Even the simplest applications,  
 like FDE, get complex as they mature as products, and vary widely in imple-  
 mentation quality. HSMs are another example of cryptosystems that acquired  
 ever more features until the features broke them, and now require other com-  
 ponents to block targeted attacks. SGX runs on processors so complex that

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 632 | Ross Anderson |

*20.9. SUMMARY*

it’s vulnerable to multiple side-channel attacks, and Intel doesn’t even consider  
 some of them to be within its threat model: if a capable motivated opponent  
 can run their code on the same machine as you, you’re basically toast. Much the  
 same holds for blockchains, which have developed the most complex ecosystem  
 of all. Even the basic assumption that rational miners are not motivated to  
 rewrite history starts to fail when applications create the necessary incentives.  
 Again, a cryptocurrency can go on acquiring features until they break it, and  
 smart contracts can help the process along.

The economic point is that the advanced crypto mechanisms we’ve seen

deployed all come with a signiﬁcant cost. HSMs cost more than servers. SGX has  
 memory limits and a real performance overhead on context switching. Bitcoin  
 miners emit as much CO2 as New Zealand. Smart contracts may be able to do  
 some clever things but in practice are very restricted in size and scope compared  
 with other software. There is a ﬁne calculation about whether the cost is worth  
 it; and this calculation may become more adverse over time as the maintenance  
 costs mount and the system gets into technical debt.

The policy point is that advanced cryptographic mechanisms all get tangled

up with liability. If successful they seem to acquire, as part of their core purpose,  
 either the desire to satisfy some regulation or the desire to avoid regulation. So  
 the decision to deploy them, or maintain them, may involve subtle externalities.

Hardware security modules are mandatory in card payment systems because

of card scheme rules based, ultimately, on banks’ desire to not be liable for  
 fraud. SGX is seen as a way to assure customers of cloud computing services  
 that they protect their most valuable assets against rogue sysadmins and against  
 intelligence agencies. Bitcoin and its many clones have become a mechanism  
 for circumventing everything from securities and payment law to anti-money-  
 laundering regulations. Real systems get built for strategic reasons, and that  
 tends to mean creating or entrenching power for their creators – be it market  
 power or political power.

As for cryptocurrencies, they have so far had extreme volatility, limited

capacity, unpredictable transaction costs, no governance, and limited trans-  
 parency. The proof-of-work mechanisms used by most of them cause CO2 emis-  
 sions that reasonable people might consider unacceptable, and their use in prac-  
 tice is entangled with all sorts of criminality. While the law should defend the  
 right of private ﬁrms and individuals to create value tokens such as coupons  
 and air miles, once these start being used as currency and institutions emerge  
 that behave like banks, it is reasonable for the lawgiver to treat them as such.  
 It is also reasonable for the lawgiver to think about carbon taxes, or to require  
 organisations that use blockchains to account for the CO2 they produce.

If we had to sum up the experience of forty years of trying to apply the magic

of mathematics to solve real-world problems, it would probably be TANSTAAFL:  
 there ain’t no such thing as a free lunch.

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 633 | Ross Anderson |

*20.9. SUMMARY*

**Research Problems**

There are deep problems around decentralisation that cross the boundary be-  
 tween cryptography and system security. Decentralised protocols tend to fos-  
 silise; we’re still using email, DNS and BGP mechanisms from the early 1990s  
 because of the di�culty of changing anything. End-to-end crypto could not be  
 layered on top of SMTP email, despite the e↵orts of PGP, but needed to wait  
 for a new platform like Signal that could impose it by ﬁat.

Bitcoin provides another example. The original cypherpunks ideal was a

fully decentralised payment system providing a means of exchange and a store  
 of value without the involvement of governments or other dominant players  
 such as banks. Yet the production of mining rigs has become a monopoly,

controlled by Bitmain, while the ASICs all come from TSMC. The great majority  
 of Bitcoin users rely on custodial exchanges to hold their cryptocurrency, and  
 these exchanges do most of the trading – DEXs are only 0.01% of it. The

custodial exchanges have in e↵ect become unregulated banks.

In systems such as Signal, Tor and Bitcoin, the real consensus is not crypto-

graphic but social; it’s the consensus of the developers. In Tor this is a commu-  
 nity while in the world of cryptocurrency there are competing developer teams  
 working for proﬁt. The security economics may be expected be more important  
 than the cryptography, and we’ve already seen how smart contracts can create  
 application-layer incentives that could break the underlying consensus layer.

What about the dependability of smart contracts in general? The computer

science approach to the API security problem has been to try to adapt formal-  
 methods tools to prove that interfaces are safe. There is a growing literature on  
 this, and even a series of workshops, but the methods can still only tackle fairly  
 simple APIs. Smart contracts are running into similar problems, complicated  
 by the di�culty of changing them to ﬁx bugs or to respond to changing circum-  
 stances. It is unsurprising that many of the smart contracts used to set up DEXs  
 have hard-coded admin keys that enable human intervention if need be. This is  
 just prudent engineering, but calls into question the ideological justiﬁcation of  
 such exchanges as ‘trustless’.

**Further Reading**

To get up to speed on Tor, a good starting point is the Tor Project’s docu-  
 mentation page. For more detail on how Bitcoin works, read the Princeton

book [1383] or the JEP paper [274], while for our more detailed view on tracing  
 stolen Bitcoin and on cryptocurrency regulation, see [116]. For a discussion of  
 the interaction between centralisation and privacy, see Carmela Troncoso and  
 colleagues [1910]. A survey of the state of play in messaging apps in 2015 (the  
 time when Signal came together from previous apps for messaging and VOIP)  
 can be found at [1917].

|  |  |  |
| --- | --- | --- |
| **Security Engineering** | 634 | Ross Anderson |