OSY.SSI[2019][13]

In the previous episode

- Killchain
- Basic cryptography (OTP, provable sec, hybrid crypto)
- Basic issues (need for randomness, padding)

Quick summary

Functionality	*Provable*	*Symmetric*	*Examples*
Key exchange	AKE / KEM	-	ECDHE
Confidentiality	PKE	Block ciphers / Stream ciphers	RSA-OAEP, AES-128-CTR
Integrity	Signature	MAC	ECDSA, Poly1305-AES

Hybrid crypto: negotiate symmetric session keys, then use ciphers + MACs.

Reminder: What you'll need to understand

Even if you're not a cryptographer

1. How to *correctly use it* (what to encrypt, how, what mode, what blocksize?, where to sign, when to MAC? etc.)

Reminder: What you'll need to understand

Even if you're not a cryptographer

1. How to *correctly use it* (what to encrypt, how, what mode, what blocksize?, where to sign, when to MAC? etc.)
2. How to manage keys? How to manage randomness?
3. How to *correctly implement cryptographic algorithms*

A quick review of AKE

Idea: Diffie–Hellman key exchange protocol is vulnerable to a Man-in-the-Middle attack. We avoid this by using a signature to authenticate the parties.

This means Alice (resp. Bob) owns a secret signing key. Write $[m]_A$ a message signed by Alice (similarly for Bob).

To check the validity of this signed message, Bob must know the corresponding public verification key. How does he know it?

A quick review of AKE

Idea: Diffie–Hellman key exchange protocol is vulnerable to a Man-in-the-Middle attack. We avoid this by using a signature to authenticate the parties.

This means Alice (resp. Bob) owns a secret signing key. Write $[m]_A$ a message signed by Alice (similarly for Bob).

To check the validity of this signed message, Bob must know the corresponding public verification key. How does he know it?

A quick review of AKE (cont'd)

Idea: a third party commits to the claim that a certain public key is Alice's. Let call this third party Charlie.

Charlie publishes a certificate of the form $[\mathrm{pk}_A, \ldots]_C$.

A quick review of AKE (cont'd)

Idea: a third party commits to the claim that a certain public key is Alice's. Let call this third party Charlie.

Charlie publishes a certificate of the form $[\mathrm{pk}_A, \ldots]_C$.

But how do we know that it's really Charlie's signature?

A quick review of AKE (cont'd)

Idea: a third party commits to the claim that a certain public key is Alice's. Let call this third party Charlie.

Charlie publishes a certificate of the form $[\mathrm{pk}_A, \ldots]_C$.

But how do we know that it's really Charlie's signature?Well Denise publishes a certificate attesting to this. And how do we know that Denise...

Public-key infrastructure

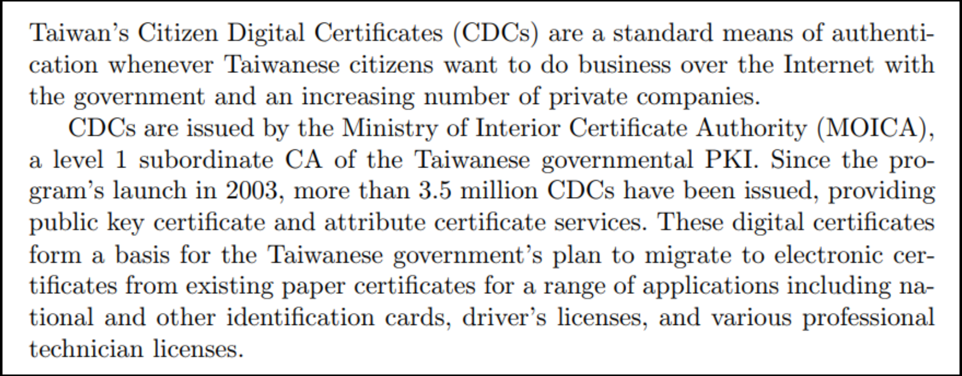
At some point we reach one of these:

- We forget to check the validity of a certificate (oops!)
- The certificate is a tautology (self-signed certificate, trust me! signed: me)
- We trust the certificate (root certificate, trust me! ok.)

Questions:

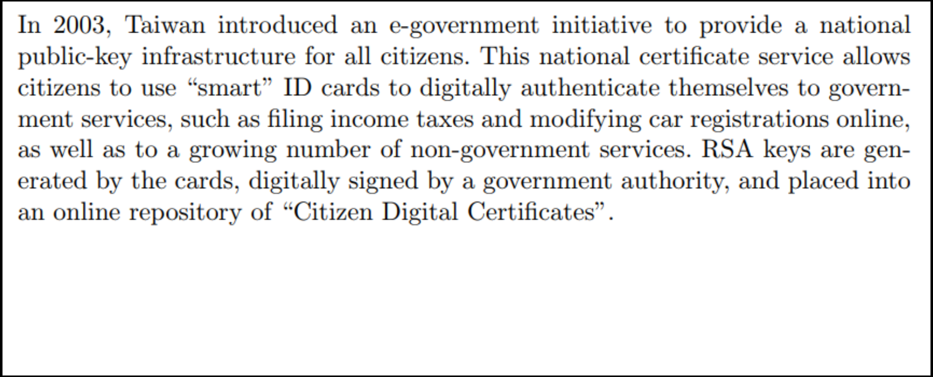
- how are root certificates published?
- what happens if a certificate is revoked?

Root certificates are trusted blindly.

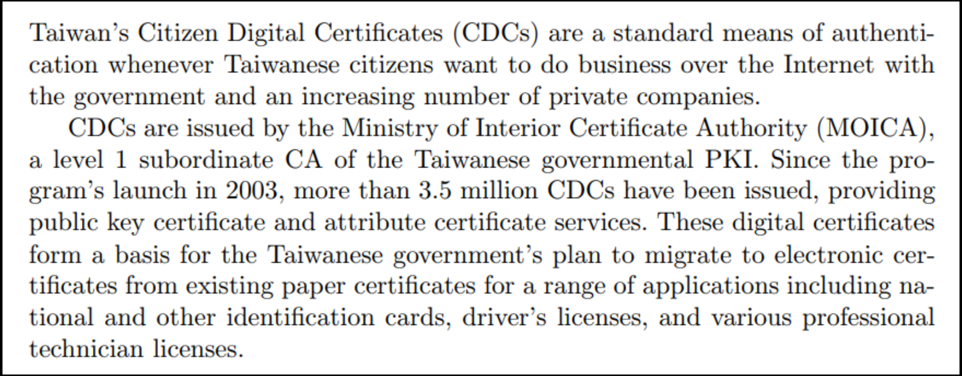


Taiwan's Citizen Digital Certificates (CDCs) are a standard means of authentication whenever Taiwanese citizens want to do business over the Internet with the government and an increasing number of private companies.

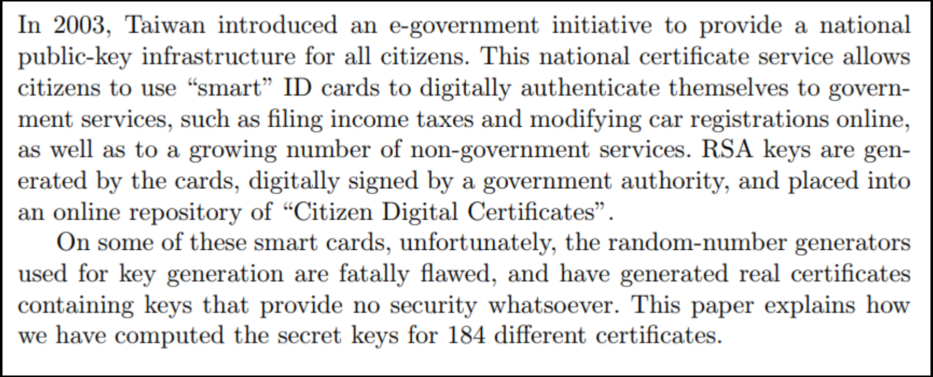
CDCs are issued by the Ministry of Interior Certificate Authority (MOICA), a level 1 subordinate CA of the Taiwanese governmental PKI. Since the program's launch in 2003, more than 3.5 million CDCs have been issued, providing public key certificate and attribute certificate services. These digital certificates form a basis for the Taiwanese government's plan to migrate to electronic certificates from existing paper certificates for a range of applications including national and other identification cards, driver's licenses, and various professional technician licenses.



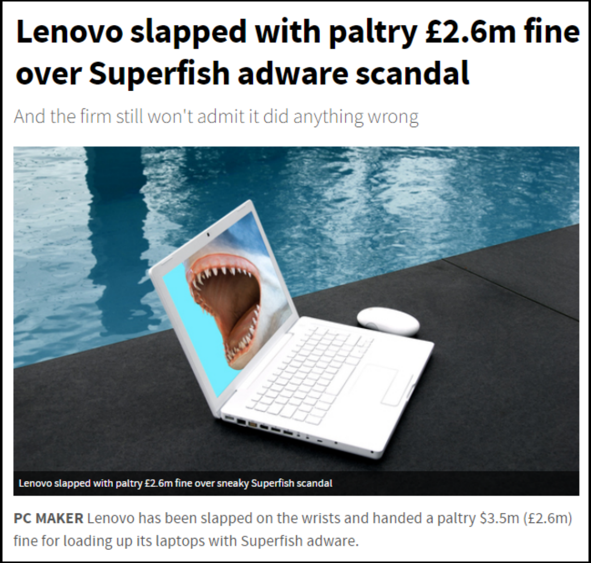
In 2003, Taiwan introduced an e-government initiative to provide a national public-key infrastructure for all citizens. This national certificate service allows citizens to use "smart" ID cards to digitally authenticate themselves to government services, such as filing income taxes and modifying car registrations online, as well as to a growing number of non-government services. RSA keys are generated by the cards, digitally signed by a government authority, and placed into an online repository of "Citizen Digital Certificates".


Taiwan's Citizen Digital Certificates (CDCs) are a standard means of authentication whenever Taiwanese citizens want to do business over the Internet with the government and an increasing number of private companies.

CDCs are issued by the Ministry of Interior Certificate Authority (MOICA), a level 1 subordinate CA of the Taiwanese governmental PKI. Since the program's launch in 2003, more than 3.5 million CDCs have been issued, providing public key certificate and attribute certificate services. These digital certificates form a basis for the Taiwanese government's plan to migrate to electronic certificates from existing paper certificates for a range of applications including national and other identification cards, driver's licenses, and various professional technician licenses.


In 2003, Taiwan introduced an e-government initiative to provide a national public-key infrastructure for all citizens. This national certificate service allows citizens to use "smart" ID cards to digitally authenticate themselves to government services, such as filing income taxes and modifying car registrations online, as well as to a growing number of non-government services. RSA keys are generated by the cards, digitally signed by a government authority, and placed into an online repository of "Citizen Digital Certificates".

On some of these smart cards, unfortunately, the random-number generators used for key generation are fatally flawed, and have generated real certificates containing keys that provide no security whatsoever. This paper explains how we have computed the secret keys for 184 different certificates.

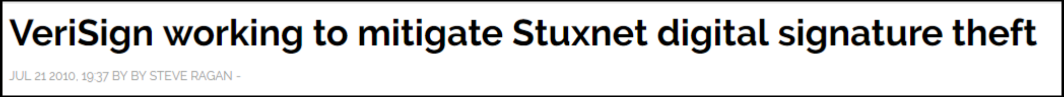


Lenovo slapped with paltry £2.6m fine over Superfish adware scandal

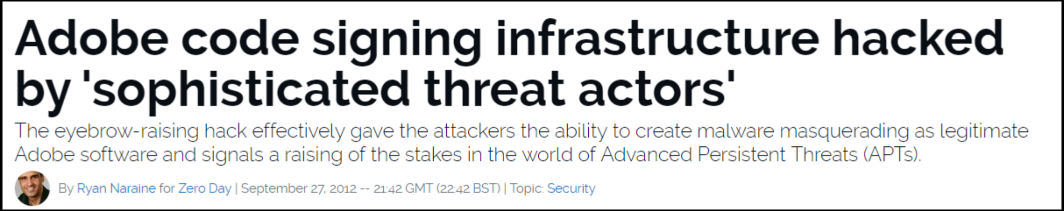
And the firm still won't admit it did anything wrong

Lenovo slapped with paltry £2.6m fine over sneaky Superfish scandal

PC MAKER Lenovo has been slapped on the wrists and handed a paltry $3.5m (£2.6m) fine for loading up its laptops with Superfish adware.

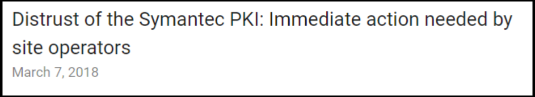

VeriSign working to mitigate Stuxnet digital signature theft

JUL 21 2010, 19:37 BY BY STEVE RAGAN -


Adobe code signing infrastructure hacked by 'sophisticated threat actors'

The eyebrow-raising hack effectively gave the attackers the ability to create malware masquerading as legitimate Adobe software and signals a raising of the stakes in the world of Advanced Persistent Threats (APTs).

By Ryan Naraine for Zero Day | September 27, 2012 -- 21:42 GMT (22:42 BST) | Topic: Security


Distrust of the Symantec PKI: Immediate action needed by site operators

March 7, 2018

Check your certificates! `https://www.ssllabs.com/ssltest/`

Assume a trusted PKI

You can ask for a certificate. Some ask questions, some don't.

You can even generate a certificate yourself.

If you don't use a certificate, or if you use a self-signed certificate, an attacker can fool you on the verification key. You have no protections. Don't do that.

Getting a valid and trusted certificate is not hard: `https://letsencrypt.org/`
Note: Your public (verification) key is only certified for a certain duration (1 yr).

Assume a trusted PKI

Ok so we have

- A long-term private signing key
- A long-term verification key
- A certificate attesting that we are indeed the owner of the above key pair
- A short-term encryption key
- A short-term integrity key

That's a lot of keys! What happens if I lose one? If it's stolen?

Add to that the parameters (group definition, generators, etc.)

Key size

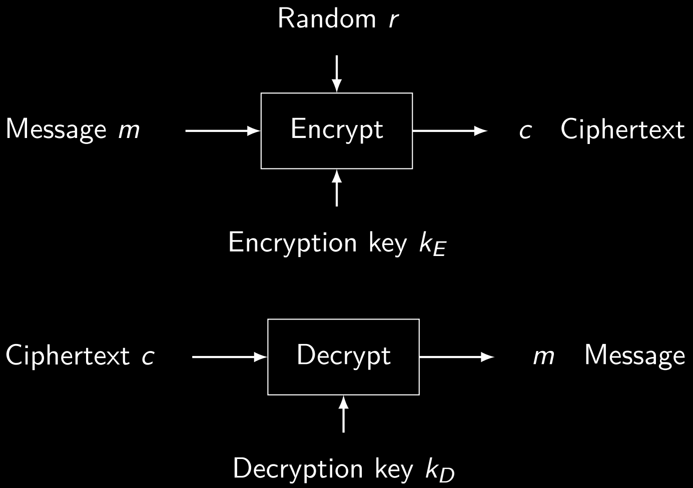
We saw how key size is determined: find the best attack, and make sure it takes around 2^{128} operations to run.
This gives the following estimates:

Algo	Example	Key size
Encryption (sym.)	AES, Chacha20, 3DES	256 bits
MAC (sym.)	HMAC-SHA256, Poly1305	256 bits
Key exchange (FF)	DHE	3072 bits
Key exchange (EC)	ECDHE, Curve25519	256 bits
Signature (RSA, DSA)	RSA-OAEP, DSA	3072 bits
Signature (EC)	ECDSA, EdDSA	256 bits

These are based on the best known classical generic algorithms.

Question: why don't I just choose the biggest option available?

Say you want to encrypt



For a correct key pair $k_E \sim k_D$, we can encrypt and decrypt, recovering m.
For an incorrect key pair $k_E \not\sim k_D$ we get a random value.
In any case, we can't learn r nor k_E nor k_D from m and c.

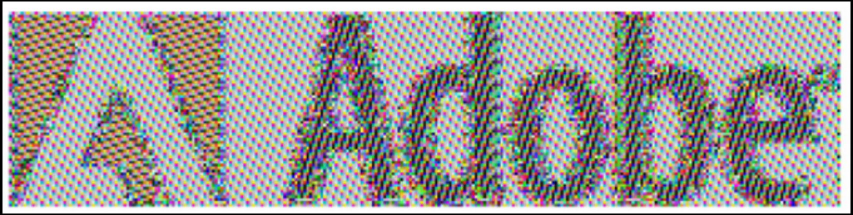
The chaining problems

An encryption algorithm usually has a fixed input and output size, e.g. 128 bits for AES.

We usually want to encrypt more (or less) than this amount.

How we achieve this precisely is called a mode of operation.
For instance we may cut the input into 128 bits block, and encrypt them.

The ECB mode of operation



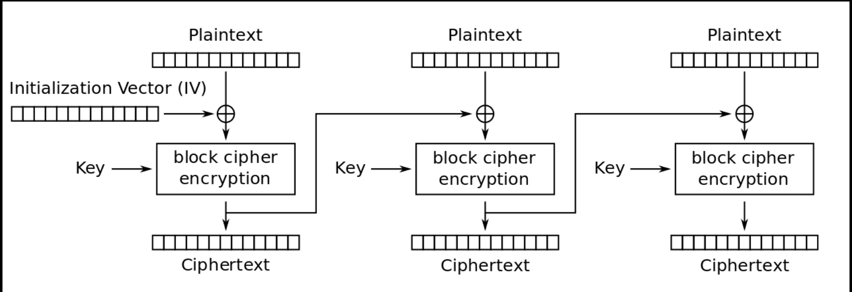
Good modes of operation

A “good” mode of operation ideally

- Has a security reduction (i.e. doesn’t degrade security)
- Has an input for the randomness (IV or nonce)
- Is easy to implement and efficient

Question: pros/cons with ECB and CTR?

Is this a good mode?



Cipher Block Chaining (CBC) mode encryption

Good modes of operation for encryption

Some decent choices:

- OCB (patented)
- GCM (hard to implement, nonce-misuse issues)
- GCM-SIV (not standardised)
- (CCM, EAX, DFF, FFX, RAC)

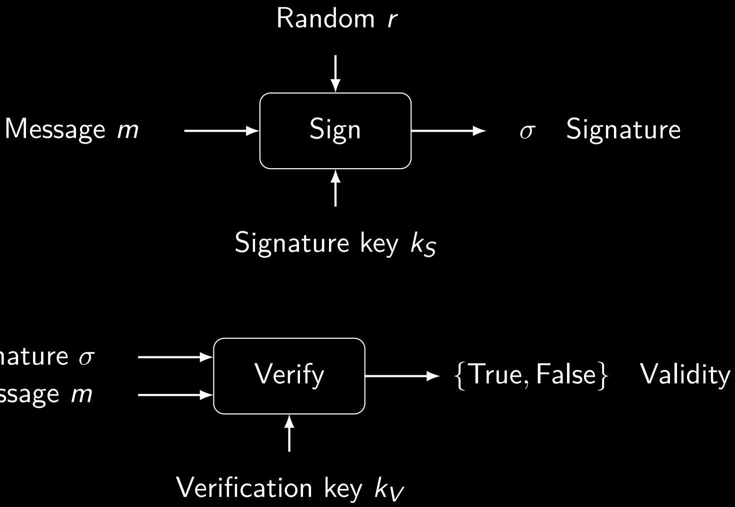
Some less decent choices:

- ECB (deterministic)
- CBC (no CCA security)
- CFB, OFB (no CCA security)

XTS is in the middle somewhere

Note: What constitutes a ‘good’ mode depends on the application.

So you want to sign?



Property: EF-CMA, it is difficult to obtain a valid couple $\{m, \sigma\}$ without knowing k_S.

What do we sign exactly?

As for encryption, a signature algorithm typically has a fixed input (and output).

So we don't really sign a message, but rather a fixed-length function of it. This is achieved by using a collision-resistant hash function.

Otherwise, an adversary may generate a fake message $m' \neq m$ for which σ is a valid signature.

- Reasonable choices of CRHF: SHA-2, SHA-3
- Bad choices of CHRF: MD4, MD5, RIPE-MD, SHA-1
- Worst choice: non-cryptographic hash functions

What remains

- How to manage your keys (generate, share, destroy, use, etc.)
- How to produce randomness

What is random?

Let’s start with a fun exercise: how to generate a fair coin toss using a biased coin?

What is random?

Let’s start with a fun exercise: how to generate a fair coin toss using a biased coin?

von Neumann: toss twice, if same discard. If head-tail output 0 else 1.

Exercise: prove that this works.

What is random?

Let $f\colon \{0,1\}^n \to \{0,1\}^m$ be a function; let X be a distribution over $\{0,1\}^n$. We say that f is an ϵ-extractor over X if the statistical distance between $f(X)$ and the uniform distribution is $< \epsilon$.

Remark: if f is 0-ext, $\Pr[X = x] \leq 2^{-m}$. This leads to the following definition

The min-entropy of X is $H_\infty(X) = -\min_x \log_2(\Pr[X = x])$.

Intuition: The min-entropy measures the number of random bits that can (hopefully) be extracted from a source.

Unfortunately, not all sources X can be extracted. One common strategy is to accumulate different sources into “pools”.

Using true random to seed pseudo random

Because randomness is *hard* to produce, it is sometimes ok to use a PRNG seeded with a random.

In cryptography we usually want cryptographically secure (fast xor provable) PRNGs, which provide some guarantees that the results “looks random” and does not inform the adversary too much.

Example: Linux `/dev/random`, `/dev/urandom`, `/dev/arandom`.

What sources?

What sources?

- **Quantum physics:** shot noise, nuclear decay, photon noise, avalanche noise, spontaneous parametric down-conversion scattering
- **Thermal noise**
- **Oscillator drift**

These are considered *true* sources of randomness, but they are still biased and manipulable.

When no hardware RNG is available, some entropy may be gathered from less reliable sources (network events, user inputs).

The amount of randomness that can be produced per second depends on the source quality as a TRNG.

Key management issues

- How do I securely and correctly generate a key?
- How do I securely store a key? How long?
- How do I securely use a key? How many times?
- Can I use the same key for encryption and signature?
- How do I securely dispose of a key?
- Who's in charge?

Key management issues

- How do I securely and correctly generate a key?
- How do I securely store a key? How long?
- How do I securely use a key? How many times?
- Can I use the same key for encryption and signature?
- How do I securely dispose of a key?
- Who's in charge?

The best way is not to do it yourself: use security module interfaces (HSM, TPM) and hire professionals for operations, checks and balances.

1 slide summary

PKI certification	$\rightarrow$ long term, personal signature key
Authenticated key exchange	$\rightarrow$ ephemeral, shared session key
Symmetric encryption+MAC (or AEAD)	$\rightarrow$ confid + integr. on channel
Key sizes chosen to ensure heuristic effort of about 2^{128}	$\rightarrow$ security level
Key management is essential and critical	Use HSMs
Randomness is essential and critical	Use TRNGs

E.g.: Use Curve25519 + EdDSA + SHA3 + AES-GCM-SIV

How trustworthy is cryptography

As long as politicians stay out of it, quite safe.

How trustworthy is cryptography

As long as politicians stay out of it, quite safe.

We described algorithms as ideal machines, but we know they are not.

How trustworthy is cryptography

As long as politicians stay out of it, quite safe.

We described algorithms as ideal machines, but we know they are not.

- Bad randomness (BULLRUN, Debian...)

How trustworthy is cryptography

As long as politicians stay out of it, quite safe.

We described algorithms as ideal machines, but we know they are not.

- Bad randomness (BULLRUN, Debian...)
- Bad implementation (Apple, Shellshock...)

How trustworthy is cryptography

As long as politicians stay out of it, quite safe.

We described algorithms as ideal machines, but we know they are not.

- Bad randomness (BULLRUN, Debian...)
- Bad implementation (Apple, Shellshock...)
- Bad usage (Venona, ...)

How trustworthy is cryptography

As long as politicians stay out of it, quite safe.

We described algorithms as ideal machines, but we know they are not.

- Bad randomness (BULLRUN, Debian…)
- Bad implementation (Apple, Shellshock…)
- Bad usage (Venona, …)
- Side channels (ROBOT, FREAK, Spectre/Meltdown…)

How trustworthy is cryptography

As long as politicians stay out of it, quite safe.

We described algorithms as ideal machines, but we know they are not.

- Bad randomness (BULLRUN, Debian...)
- Bad implementation (Apple, Shellshock...)
- Bad usage (Venona, ...)
- Side channels (ROBOT, FREAK, Spectre/Meltdown...)
- Legacy compatibility (Logjam, DROWN...)

How trustworthy is cryptography

As long as politicians stay out of it, quite safe.

We described algorithms as ideal machines, but we know they are not.

▶ Bad randomness	(BULLRUN, Debian...)
▶ Bad implementation	(Apple, Shellshock...)
▶ Bad usage	(Venona, ...)
▶ Side channels	(ROBOT, FREAK, Spectre/Meltdown...)
▶ Legacy compatibility	(Logjam, DROWN...)
▶ Bad key handling	(WPA2, SGX, Sony PS3, Yes cards...)

How trustworthy is cryptography

As long as politicians stay out of it, quite safe.

We described algorithms as ideal machines, but we know they are not.

▶ Bad randomness	(BULLRUN, Debian...)
▶ Bad implementation	(Apple, Shellshock...)
▶ Bad usage	(Venona, ...)
▶ Side channels	(ROBOT, FREAK, Spectre/Meltdown...)
▶ Legacy compatibility	(Logjam, DROWN...)
▶ Bad key handling	(WPA2, SGX, Sony PS3, Yes cards...)
▶ Software or hardware faults	(Bellcore...)

How trustworthy is cryptography

As long as politicians stay out of it, quite safe.

We described algorithms as ideal machines, but we know they are not.

▶ Bad randomness	(BULLRUN, Debian...)
▶ Bad implementation	(Apple, Shellshock...)
▶ Bad usage	(Venona, ...)
▶ Side channels	(ROBOT, FREAK, Spectre/Meltdown...)
▶ Legacy compatibility	(Logjam, DROWN...)
▶ Bad key handling	(WPA2, SGX, Sony PS3, Yes cards...)
▶ Software or hardware faults	(Bellcore...)
▶ Incorrect use of algorithms	(Adobe, Twitter...)

Trends

- Simplifying design and operation (e.g. AEAD e.g. AES-GCM-SIV)
- Prevent future attacks: ratcheting, post-quantum (e.g. LWE)
- Lightweight crypto (e.g. Simon)
- Homomorphic crypto (e.g. FE, FHE)
- Doing without PKI (e.g. IBE, ABE, MPC)
- Avoid backdoors and manipulations (e.g. Curve25519, Chacha20, Poly1305)

A word on quantum computers

- Leverages physics to compute some unitary operations “for free”
- First QC 1995, large-scale/hybrids QC may appear in our lifetime
- Generic speedups (Grover) theoretically require key size doubling
- Specific speedups (Shor) theoretically break some encryption families ($\rightarrow$ PQC)
- Ongoing NIST standardisation for PQC