OSY.SSI[2018][10]

In the news...

- TODO

In the previous episode...

$$n \leftarrow pq,\ h \leftarrow g^u \cdot 2^{128} \cdot \mathsf{AKE} \cdot \mathsf{Sig}$$

$\bullet_{0}$ $\bullet_{1}$ $\bullet_{2}$ $\bullet_{3}$ $\bullet_{4}$ $\bullet_{5}$ $\bullet_{6}$ $\bullet_{7}$ $\bullet_{8}$ $\bullet_{9}$ $\bullet_{10}$ $\bullet_{11}$ $\bullet_{12}$ $\bullet_{13}$ $\bullet_{14}$ $\bullet_{15}$ EXAM

buT yOu saID “no cRypTo”!

buT yOu saID “no cRypTo”!

- I changed my mind. It happens sometimes. Doctors say it’s ok.

buT yOu saID “no cRypTo”!

- I changed my mind. It happens sometimes. Doctors say it’s ok.
- That being said it’s going to be practice-oriented.

Reminder

By Kerckhoffs' principle, we design algorithms that rely on a *key* to operate.

We build them so that

- Without that key, the algorithm cannot be operated correctly
- If you don't have the key, it should be hard to find
- The whole system can be made public because it does not contain anything sensitive

To capture mathematically the security properties we want, we introduced games:

IND-1T (Shannon)   EF-CMA

IND-1T and IND-CCA2

We saw Vigenère/OTP is IND-1T-secure (Shannon-secure) under some assumptions.

Adversary		Alice
Picks m_1, m_2		
	$\xrightarrow{m_1,m_2}$	
		Chooses one of them, m_i Encrypts m_i, which yields c_i
	$\xleftarrow{c_i}$	
Bets on $i = 1$ or $i = 2$.		

One issue however is that this only guarantees confidentiality for a single message.

Let's design a better game.

IND-1T and IND-CCA2

Let's design a better game.

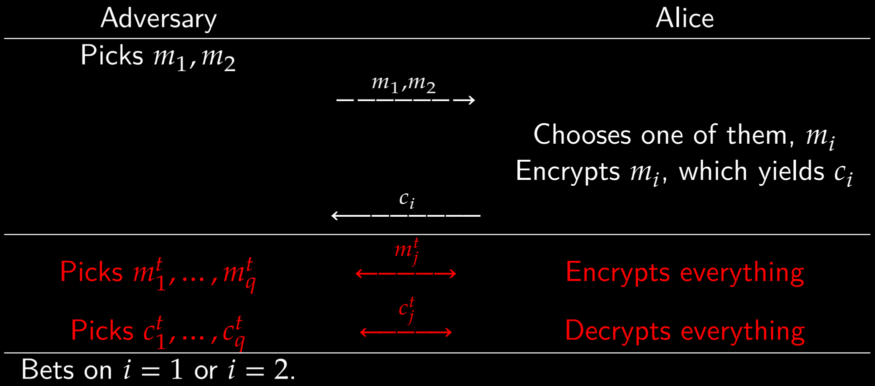
Adversary		Alice
Picks m_1, m_2		
	$\xrightarrow{m_1,m_2}$	
		Chooses one of them, m_i Encrypts m_i, which yields c_i
	$\xleftarrow{c_i}$	
Picks $m_1^t, \ldots, m_q^t$	$\xleftrightarrow{m_j^t}$	Encrypts everything
Picks $c_1^t, \ldots, c_q^t$	$\xleftrightarrow{c_j^t}$	Decrypts everything
Bets on $i = 1$ or $i = 2$.		

The adversary can encrypt or decrypt any message of her choosing except c_i.

We call this game IND-CCA2

IND-1T and IND-CCA2

Let's design a better game.



Adversary		Alice
Picks m_1, m_2		
	$\xrightarrow{m_1, m_2}$	
		Chooses one of them, m_i Encrypts m_i, which yields c_i
	$\xleftarrow{c_i}$	
Picks $m_1^t, \dots, m_q^t$	$\xleftrightarrow{m_j^t}$	Encrypts everything
Picks $c_1^t, \dots, c_q^t$	$\xleftrightarrow{c_j^t}$	Decrypts everything
Bets on $i = 1$ or $i = 2$.		

The adversary can encrypt or decrypt any message of her choosing except c_i.

We call this game IND-CCA2

(indistinguishability of encryption under an adaptively chosen ciphertext attack)

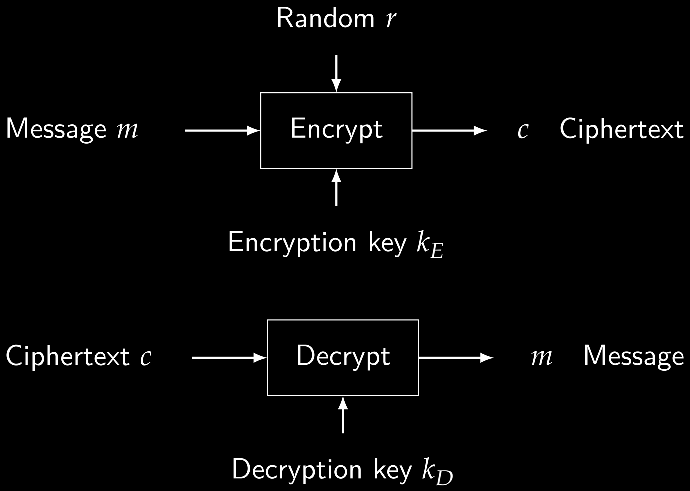
IND-1T and IND-CCA2

IND-CCA2 gives more power to the adversary, so it's easier for her to win.

If the adversary's winning probability is negligible, that means we have a very strong encryption mechanism.

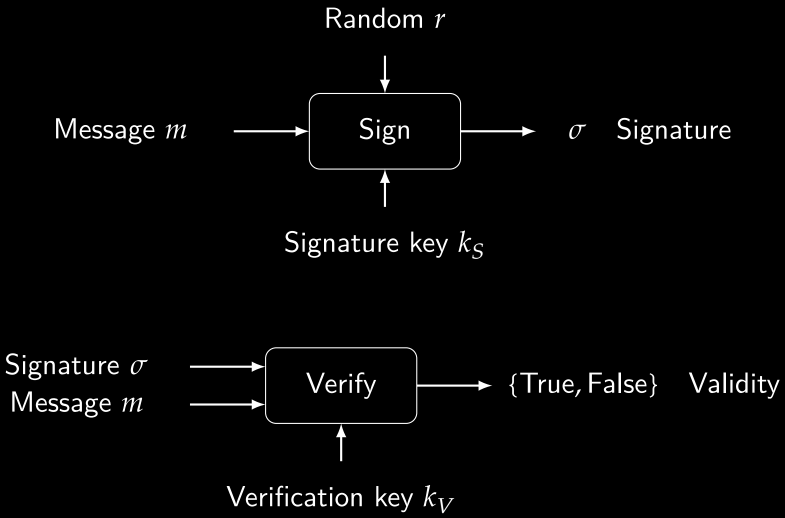
Note that a necessary condition is that encryption is randomised.
And as you know an algorithm cannot create randomness.

Summary: Encryption (ideally)



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

Summary: Signature (ideally)



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

Let’s talk about keys

Let's talk about keys

- We mentionned the notion of key-pair: $k_E \sim k_D$, $k_S \sim k_D$

Let's talk about keys

- We mentionned the notion of key-pair: $k_E \sim k_D$, $k_S \sim k_D$
- Confidentiality is ensured by the secrecy of k_D
- Integrity is ensured by the secrecy of k_S

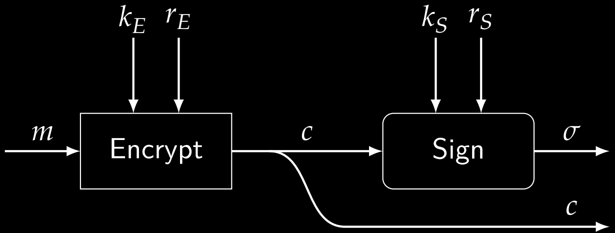
Let's talk about keys

- We mentionned the notion of key-pair: $k_E \sim k_D$, $k_S \sim k_D$
- Confidentiality is ensured by the secrecy of k_D
- Integrity is ensured by the secrecy of k_S
- In principle the other key k_E (resp. k_V) can be made public.

Both are called *public keys* (which may be confusing).
In contrast the decryption (resp. signature) keys are both called *private keys*.

Confidentiality + Integrity

You can have both



Remarks:

- Should k_S be the same as k_E?
- Should k_D be the same as k_E?
- Should k_V be the same as k_S?

Essential to have very clear ideas about which key does what!

Keys lifespan

A cryptographic key is

- Generated by some device at some point
- Used by some algorithm
- Destroyed once it is no longer needed

We distinguish *short-term keys* from *long-term keys*.

Example: HTTPS/TLS uses

- A short term confidentiality key pair $k_E = k_D$ for each session
- A short term integrity key pair $k_S = k_V$ for each session
- A long term authentication key pair $K_S \sim K_V$

More about this long term key pair

The issue is: we don't know who we're talking to over the Internet.

So we ask them to sign their messages, and we can verify with their public verification key.

The issue is: we don't know who we're talking to over the Internet. What proves that the verification key is really theirs?

A basic certificate

A certificate is a document that binds a public verification key to an entity (e.g. a website).

The document itself is signed (by a so-called certification authority, CA).

We can verify this signature using the CA's public verification key.

A basic certificate

A certificate is a document that binds a public verification key to an entity (e.g. a website).

The document itself is signed (by a so-called certification authority, CA).

We can verify this signature using the CA's public verification key.

But how do I know that the CA's verification key is really theirs?

PKI, or turtles all the way down

The CA itself has a certificate, which is signed by another CA, which has a certificate, etc.

This is called a certificate chain.

At some point, you end up with either of two situations:

PKI, or turtles all the way down

The CA itself has a certificate, which is signed by another CA, which has a certificate, etc.

This is called a certificate chain.

At some point, you end up with either of two situations:

- A CA that signed its own public verification key (trust me! signed: me)

PKI, or turtles all the way down

The CA itself has a certificate, which is signed by another CA, which has a certificate, etc.

This is called a certificate chain.

At some point, you end up with either of two situations:

- A CA that signed its own public verification key (trust me! signed: me)
- A CA that has no certficate (trust me.)

PKI, or turtles all the way down

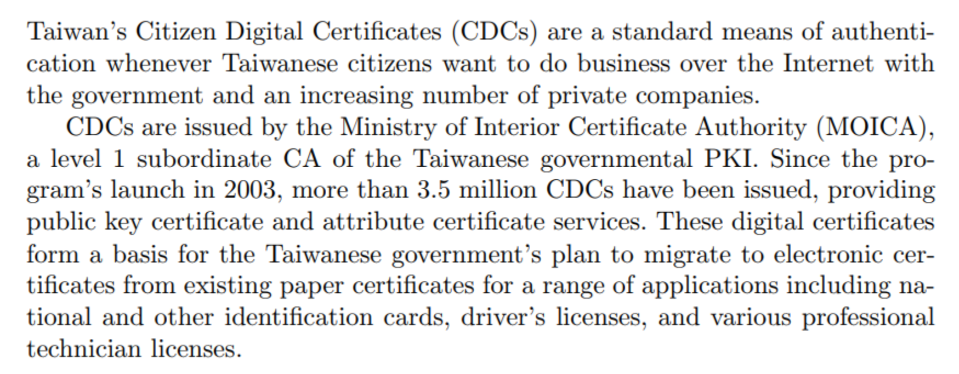
The CA itself has a certificate, which is signed by another CA, which has a certificate, etc.

This is called a certificate chain.

At some point, you end up with either of two situations:

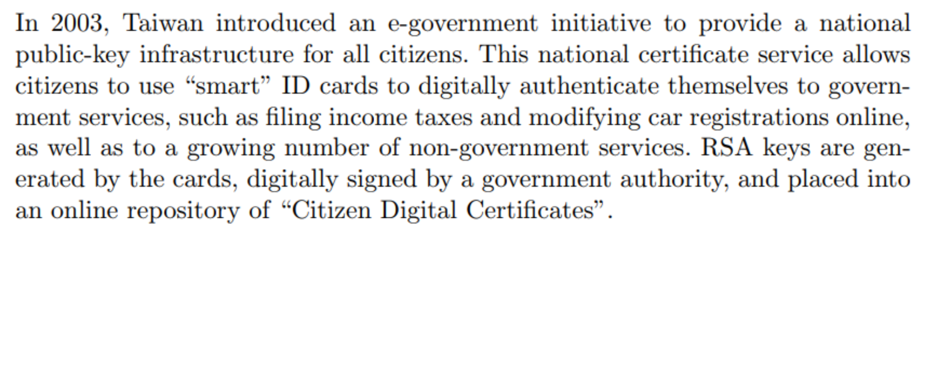
- A CA that signed its own public verification key (trust me! signed: me)
- A CA that has no certficate (trust me.)

Root certificates are trusted blindly.

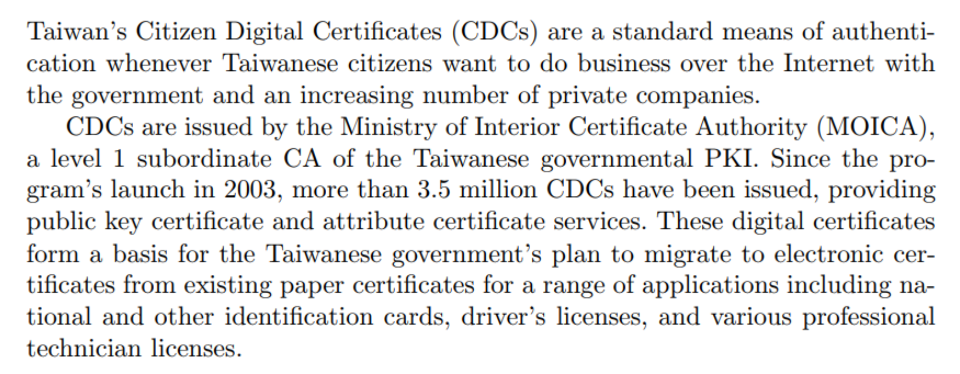


Taiwan's Citizen Digital Certificates (CDCs) are a standard means of authentication whenever Taiwanese citizens want to do business over the Internet with the government and an increasing number of private companies.

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.

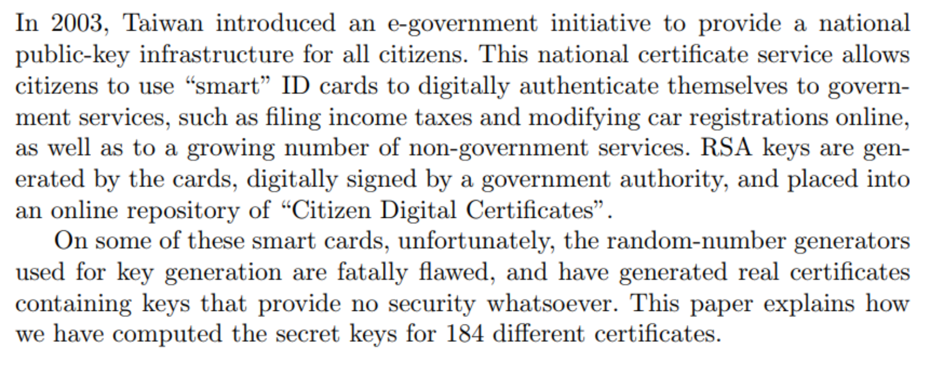


In 2003, Taiwan introduced an e-government initiative to provide a national public-key infrastructure for all citizens. This national certificate service allows citizens to use "smart" ID cards to digitally authenticate themselves to government services, such as filing income taxes and modifying car 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".



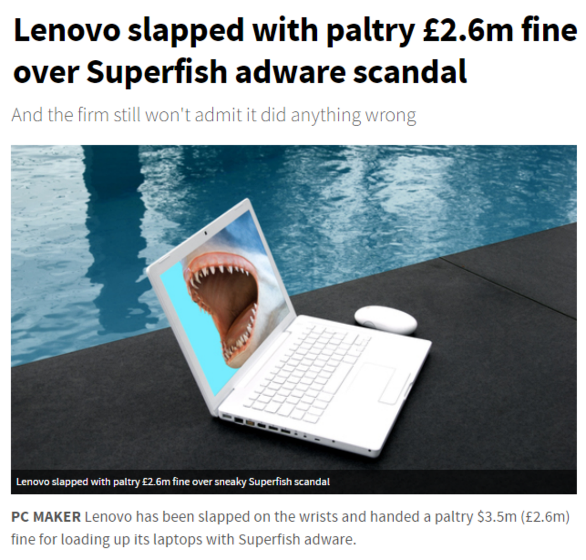
Taiwan's Citizen Digital Certificates (CDCs) are a standard means of authentication whenever Taiwanese citizens want to do business over the Internet with the government and an increasing number of private companies.

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.

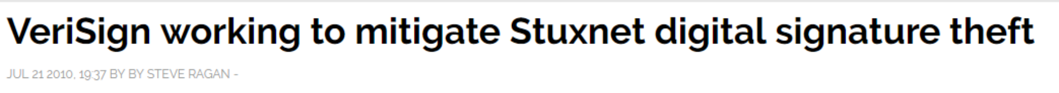


Lenovo slapped with paltry £2.6m fine over Superfish adware scandal

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

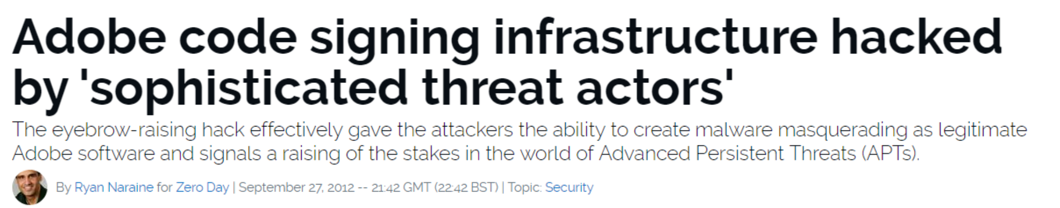
Lenovo slapped with paltry £2.6m fine over sneaky Superfish scandal

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

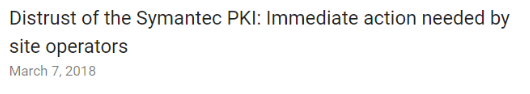
JUL 21 2010, 19:37 BY BY STEVE RAGAN -



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 ~~privilege~~ 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	2048 bits
Key exchange (EC)	ECDHE, Curve25519	256 bits
Signature (RSA, DSA)	RSA-OAEP, DSA	2048 bits
Signature (EC)	ECDSA, EdDSA	256 bits

These are based on the best known classical generic algorithms.

How is it managed, in practice?

- Choose public key parameters (e.g.: Curve25519 + EdDSA)

How is it managed, in practice?

- Choose public key parameters (e.g.: Curve25519 + EdDSA)
- Choose a cipher suite (e.g. Chacha20 + Poly1305)

How is it managed, in practice?

- Choose public key parameters (e.g.: Curve25519 + EdDSA)
- Choose a cipher suite (e.g. Chacha20 + Poly1305)
- Generate your private signing key and your public verification key
- Get a certificate for your verification key

Do this for each domain you need a TLS certificate for.

How is it managed, in practice?

- Choose public key parameters (e.g.: Curve25519 + EdDSA)
- Choose a cipher suite (e.g. Chacha20 + Poly1305)
- Generate your private signing key and your public verification key
- Get a certificate for your verification key

Do this for each domain you need a TLS certificate for.

The rest is figuring out how you deal with keys. Where you store them. How you destroy them. What you do when they get lost.

Session vs. Storage

Over the Internet, a session is short. Keys don't need to be stored.

But sometimes you need to store encrypted information for longer.

That means you need to store a decryption key for longer. (And that's annoying.)

Session vs. Storage

Over the Internet, a session is short. Keys don't need to be stored.

But sometimes you need to store encrypted information for longer.

That means you need to store a decryption key for longer. (And that's annoying.)
Oh, and you need to rotate your keys if you keep data for very long.

Session vs. Storage

Over the Internet, a session is short. Keys don't need to be stored.

But sometimes you need to store encrypted information for longer.

That means you need to store a decryption key for longer. (And that's annoying.)
Oh, and you need to rotate your keys if you keep data for very long.

Try to avoid that if you can. If you can't, do everything in your power to ensure C+I+A of your keys.

Typical example: Credential storage

You want to authenticate a user.

We know passwords are a bad idea, and we know some crypto, so what should we do?

Typical example: Credential storage

You want to authenticate a user.

We know passwords are a bad idea, and we know some crypto, so what should we do?

Ask users to sign a message. We only store their public verification key.

(why don't we do this?)

Typical example: Credential storage

You want to authenticate a user.

We know passwords are a bad idea, and we know some crypto, so what should we do?

Ask users to sign a message. We only store their public verification key.
(why don't we do this?) we do, just not everywhere.

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...)

Illustration: Bellcore (Boneh–DeMillo–Lipton) attack

Reminder: to RSA-sign a message m, compute $\sigma = m^{1/e} \bmod n$.
The quantity $1/e = d$ is the private signing key.

Illustration: Bellcore (Boneh–DeMillo–Lipton) attack

Reminder: to RSA-sign a message m, compute $\sigma = m^{1/e} \bmod n$.
The quantity $1/e = d$ is the private signing key.

It is possible to speed things up by using the Chinese remainder theorem (CRT)

$$x = 4 \bmod 7$$
$$x = 3 \bmod 11$$

$\Rightarrow x = 4 + 7k + 3 + 11\ell \Rightarrow x = 25 \bmod 77$

Illustration: Bellcore (Boneh–DeMillo–Lipton) attack

In our case, we define $m_p = m \bmod p$ and $m_q = m \bmod q$, as well as $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$. Then

$$\begin{aligned}\sigma_p &= m_p^{d_p} \bmod p\\ \sigma_q &= m_q^{d_q} \bmod q\\ \sigma &= \left[\left(\sigma_q - \sigma_p\right) p^{-1} \bmod q\right] p + \sigma_p\end{aligned}$$

(check that at least experimentally)

Illustration: Bellcore (Boneh–DeMillo–Lipton) attack

In our case, we define $m_p = m \bmod p$ and $m_q = m \bmod q$, as well as $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$. Then

$$\begin{aligned} \sigma_p &= m_p^{d_p} \bmod p \\ \sigma_q &= m_q^{d_q} \bmod q \\ \sigma &= \left[\left(\sigma_q - \sigma_p\right) p^{-1} \bmod q\right] p + \sigma_p \end{aligned}$$

(check that at least experimentally)

Now, assume that during the computation of (say) σ_q there was a glitch.

$$\sigma' = \left[\left(\sigma_q' - \sigma_p\right) p^{-1} \bmod q\right] p + \sigma_p$$

Then $\gcd(\sigma - \sigma', n)$ gives you the factorisation of n.

Illustration: Bellcore (Boneh–DeMillo–Lipton) attack

In our case, we define $m_p = m \bmod p$ and $m_q = m \bmod q$, as well as $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$. Then

$$\begin{aligned} \sigma_p &= m_p^{d_p} \bmod p \\ \sigma_q &= m_q^{d_q} \bmod q \\ \sigma &= \left[\left(\sigma_q - \sigma_p\right) p^{-1} \bmod q\right] p + \sigma_p \end{aligned}$$

(check that at least experimentally)

Now, assume that during the computation of (say) σ_q there was a glitch.

$$\sigma' = \left[\left(\sigma_q' - \sigma_p\right) p^{-1} \bmod q\right] p + \sigma_p$$

Then $\gcd(\sigma - \sigma', n)$ gives you the factorisation of n. Same thing happens with σ_p.

Illustration: Bellcore (Boneh–DeMillo–Lipton) attack

In our case, we define $m_p = m \bmod p$ and $m_q = m \bmod q$, as well as $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$. Then

$$\begin{aligned}
\sigma_p &= m_p^{d_p} \bmod p \\
\sigma_q &= m_q^{d_q} \bmod q \\
\sigma &= \left[\left(\sigma_q - \sigma_p\right) p^{-1} \bmod q\right] p + \sigma_p
\end{aligned}$$

(check that at least experimentally)

Now, assume that during the computation of (say) σ_q there was a glitch.

$$\sigma' = \left[\left(\sigma_q' - \sigma_p\right) p^{-1} \bmod q\right] p + \sigma_p$$

Then $\gcd(\sigma - \sigma', n)$ gives you the factorisation of n. Same thing happens with σ_p.

In fact, you don't need to know σ, you can just compute $\gcd\left(m - (\sigma'^e \bmod n), n\right)$ and BAM.

Trends

- Simplifying design and operation 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)

Pause

Please come and ask questions - Please look into it

- David Kahn, The codebreakers
- Lindell and Katz, Introduction to crypto
- Boneh and Shoup, Graduate course in crypto

Pause and then injections