Stony Brook University

2024-02-29

Public Key Cryptography

Michalis Polychronakis

Stony Brook University

Public Key Cryptography

Many algorithms with different purposes

One common property: pair of keys, one public and one secret

Session key establishment

Exchange messages to create a shared secret key

Encryption

Anyone can encrypt a message using a recipient's public key

Only the recipient can decrypt a message using their private key

No shared secret! Private key (secret) is stored only at one side

Digital signatures

Sign a message with a private key

Diffie–Hellman Key Exchange

Allows two parties to jointly *establish a shared secret key* over an insecure communication channel

- The established key can then be used to encrypt subsequent communication using a symmetric key cipher
- "New Directions in Cryptography" by Whitfield Diffie and Martin Hellman, 1976

Based on the discrete logarithm problem

$$3^{29} \bmod 17 \xrightarrow{\quad \textit{easy} \quad} ??$$

$$3^{??} \bmod 17 \xleftarrow{\quad \textit{hard} \quad} 12$$

Diffie–Hellman Key Exchange

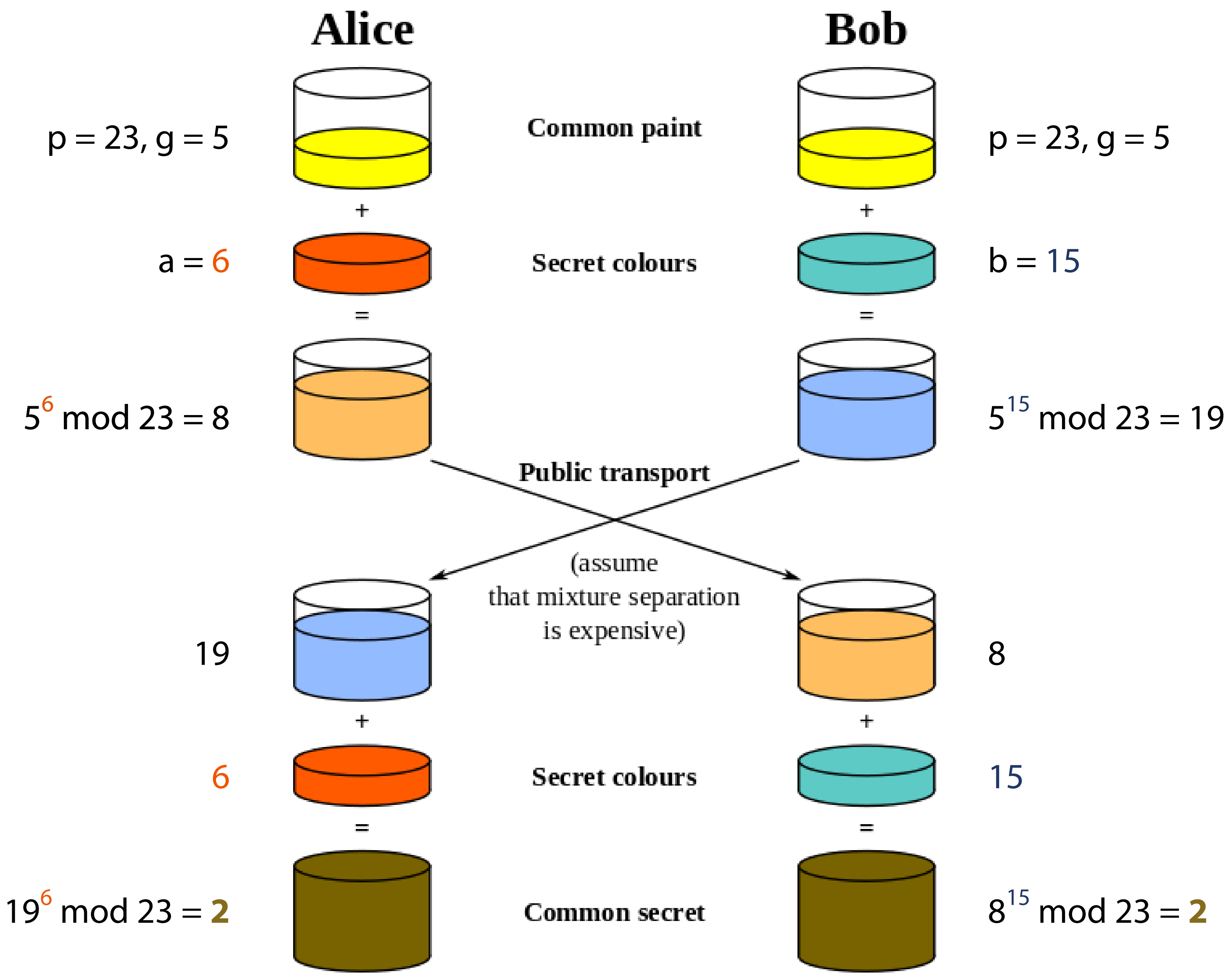
Alice and Bob agree on a large (at least 1024 bit) prime number p and a base g – *both public*

- p is usually of the form $2q+1$ where q is also prime
- g is a *generator* of the multiplicative group of integers modulo p (for every x coprime to p there is a k such that $g^k \equiv x \bmod p$)

Alice picks a secret large random number a and sends to Bob $g^a \bmod p$

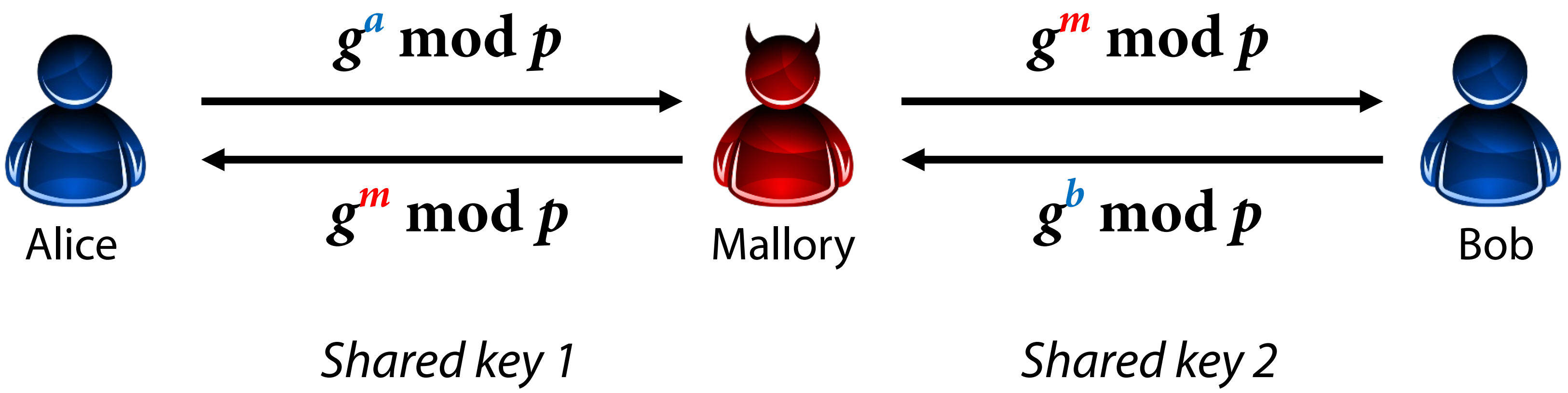
Bob picks a secret large random number b and sends to Alice $g^b \bmod p$

$$\left.\begin{array}{l}\text{Alice calculates } s = (g^b \bmod p)^a = g^{ba} \bmod p \\ \text{Bob calculates } s = (g^a \bmod p)^b = g^{ab} \bmod p\end{array}\right\}\ \textit{shared key}$$


Alice
Bob
p = 23, g = 5
Common paint
p = 23, g = 5
+
+
a = 6
Secret colours
b = 15
=
=
5^6 mod 23 = 8
5^{15} mod 23 = 19
Public transport
(assume
that mixture separation
is expensive)
19
8
+
+
6
Secret colours
15
=
=
19^6 mod 23 = 2
Common secret
8^{15} mod 23 = 2

Man-in-the-Middle Attack

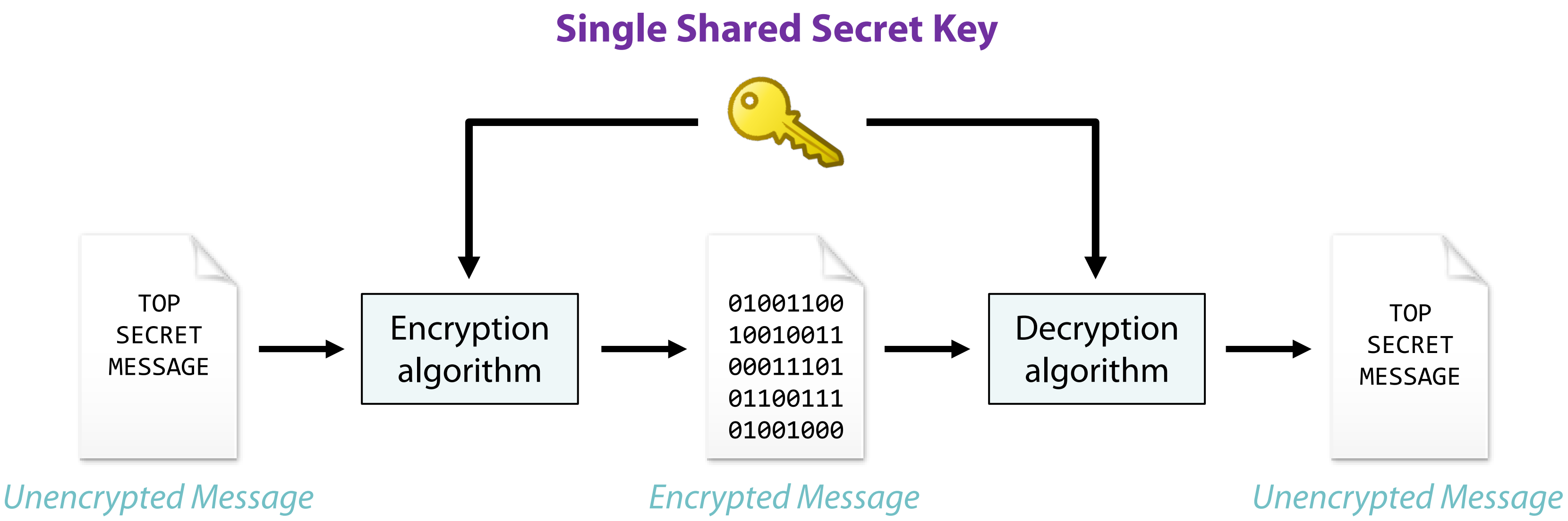
Alice and Bob share no secrets



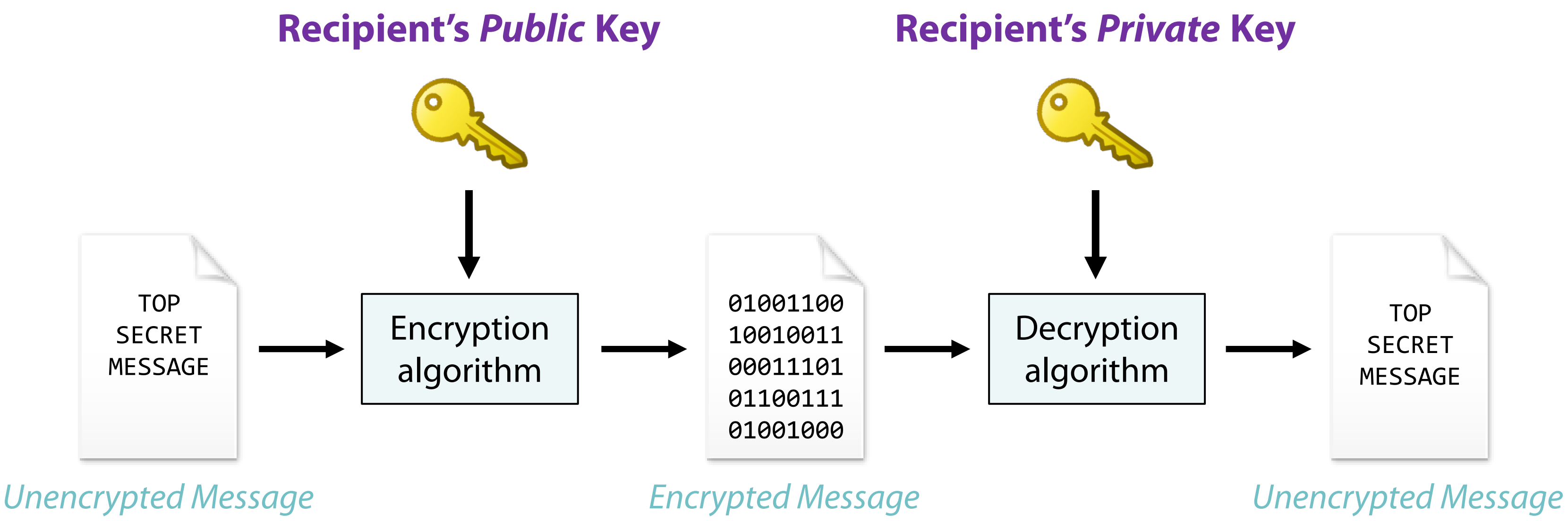
Mallory actively decrypts and re-encrypts all messages

- **No authentication:** Alice and Bob assume that they communicate directly
- General problem: *need for a root of trust* *(future lecture)*

Symmetric Key Cryptography



Public Key Cryptography



Advantages

No shared secrets

Only private keys need to be kept secret, but they are *never* shared

Easier key management

No need to transmit any secret key beforehand

For $\boldsymbol{n}$ parties, $\boldsymbol{n}$ key pairs are needed (instead of $n(n-1)/2$ shared symmetric keys)

Provides both secrecy *and authenticity*

Disadvantages

More computationally intensive

Encryption/decryption is 2–3 orders of magnitude slower than symmetric key primitives

About one order of magnitude larger keys

Key generation is more difficult

RSA Asymmetric Encryption

Named after its inventors: Rivest, Shamir, Adleman

Based on the assumption that factoring large numbers is hard

- Relatively easy to find two large prime numbers p and q
- No efficient methods are known to factor their product N

Variable key length

- Largest (publicly known) factored RSA modulus is ~~768~~ ~~795~~ 829 bits long
- February 2020: took roughly 2700 core-years
- It is believed that 1024-bit keys may already (or in the near future) be breakable by a sufficiently powerful attacker

2048-bit keys should be the absolute minimum

RSA-150	150	496		April 16, 2004	Kazumaro Aoki *et al.*
RSA-155	155	512	US$9,383[8]	August 22, 1999	Herman te Riele *et al.*
RSA-160	160	530		April 1, 2003	Jens Franke *et al.*, University of Bonn
RSA-170 [b]	170	563		December 29, 2009	D. Bonenberger and M. Krone [c]
RSA-576	174	576	US$10,000	December 3, 2003	Jens Franke *et al.*, University of Bonn
RSA-180 [b]	180	596		May 8, 2010	S. A. Danilov and I. A. Popovyan, Moscow State University[11]
RSA-190 [b]	190	629		November 8, 2010	A. Timofeev and I. A. Popovyan
RSA-640	193	640	US$20,000	November 2, 2005	Jens Franke *et al.*, University of Bonn
RSA-200 [b] ?	200	663		May 9, 2005	Jens Franke *et al.*, University of Bonn
RSA-210 [b]	210	696		September 26, 2013[12]	Ryan Propper
RSA-704 [b]	212	704	US$30,000	July 2, 2012	Shi Bai, Emmanuel Thomé and Paul Zimmermann
RSA-220 [b]	220	729		May 13, 2016	S. Bai, P. Gaudry, A. Kruppa, E. Thomé and P. Zimmermann
RSA-230 [b]	230	762		August 15, 2018	Samuel S. Gross, Noblis, Inc.
RSA-232 [b]	232	768		February 17, 2020[13]	N. L. Zamarashkin, D. A. Zheltkov and S. A. Matveev.
RSA-768 [b]	232	768	US$50,000	December 12, 2009	Thorsten Kleinjung *et al.*[14]
RSA-240 [b]	240	795		Dec 2, 2019[15]	F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thomé and P. Zimmermann
RSA-250 [b]	250	829		Feb 28, 2020[16]	F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thomé and P. Zimmermann
RSA-260	260	862			
RSA-270	270	895			

RSA

Choose two distinct large prime numbers p and q

Let $n = pq$ (modulus)

Select e as a relative prime to $(p - 1)(q - 1)$

Calculate d such that $de \equiv 1 \bmod (p - 1)(q - 1)$

Public key = (e, n)

Private key = d

To encrypt m, calculate $c \equiv m^e \bmod n$

- Plaintext block must be smaller than the key length

To decrypt c, calculate $m \equiv c^d \bmod n$

- Ciphertext block will be as long as the key

RSA in Practice

RSA calculations are computationally expensive

- Two to three orders of magnitude slower than symmetric key primitives ➔ RSA is used in combination with symmetric key encryption

Sending an encrypted message:

- Encrypt message with a random symmetric key (e.g., AES)
- Encrypt the symmetric key with the recipient's public key
- Transmit both the encrypted (with AES) message and the encrypted (with RSA) key

Setting up an encrypted communication channel:

- Negotiate a symmetric key using RSA
- Use the symmetric key for subsequent communication

PKCS: Public-Key Cryptography Standards (#1–#15)

- Make different implementations interoperable
- Avoid various known pitfalls in commonly used schemes

Forward Secrecy

Threat: capture encrypted traffic now, decrypt it in the future

- Private keys may be compromised later on (e.g., infiltrate system)
- A cryptanalytic breakthrough may be achieved

FS: Even if current keys are leaked, past encrypted traffic cannot be decrypted

- Generate random *ephemeral* secret keys without using a deterministic algorithm
- Cannot read old messages
- Cannot forge a message and claim that it was sent in the past

Support

- IPsec, SSH, Off-the-Record messaging (OTR), TLS (Diffie–Hellman instead of RSA key exchange)

Not a panacea

- Ephemeral keys may be kept in memory for hours
- Server could be forced to record all session keys
- TLS session resumption needs careful treatment

Elliptic Curve Cryptography

Proposed in 1985, but not used until 15 years later

Relies on the intractability of a different mathematical problem:
elliptic curve discrete logarithm

Main benefit over RSA: shorter key length

- Example: a 256-bit elliptic curve public key is believed to provide comparable security to a 3072-bit RSA public key

Endorsed by NIST

- Key exchange: elliptic curve Diffie–Hellman (ECDH)
- Digital signing: elliptic curve digital signature algorithm (ECDSA)

Commercial National Security Algorithm Suite and Quantum Computing FAQ

Q: What is the Commercial National Security Algorithm Suite?
A: The Commercial National Security Algorithm Suite is the suite of algorithms identified in CNSS Advisory Memorandum 02-15 for protecting NSS up to and including TOP SECRET classification. This suite of algorithms will be incorporated in a new version of the National Information Assurance Policy on the Use of Public Standards for the Secure Sharing of Information Among National Security Systems (CNSSP-15 dated October 2012). The Advisory

Algorithm	Usage
~~RSA 3072-bit or larger~~	~~Key Establishment, Digital Signature~~
~~Diffie-Hellman (DH) 3072-bit or larger~~	~~Key Establishment~~
~~ECDH with NIST P-384~~	~~Key Establishment~~
~~ECDSA with NIST P-384~~	~~Digital Signature~~
SHA-384	Integrity
AES-256	Confidentiality

September 2022: CNSA v2.0

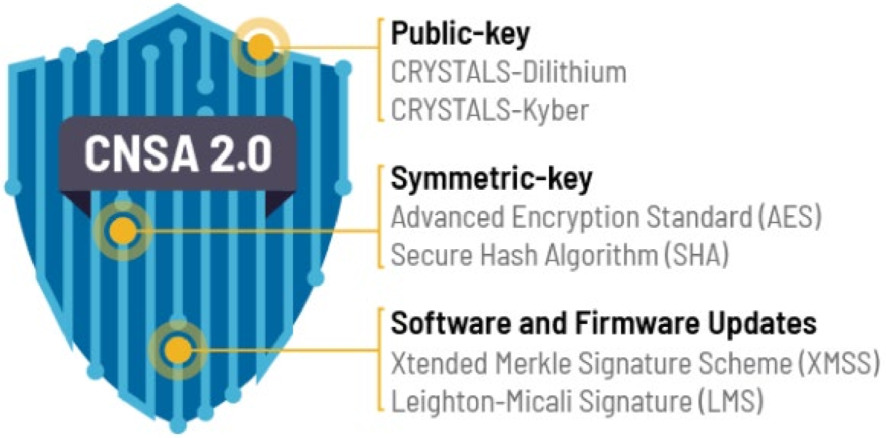


Table I: CNSA 2.0 algorithms for software and firmware updates

Algorithm	Function	Specification	Parameters
Leighton-Micali Signature (LMS)	Asymmetric algorithm for digitally signing firmware and software	NIST SP 800-208	All parameters approved for all classification levels. SHA-256/192 recommended.
Xtended Merkle Signature Scheme (XMSS)	Asymmetric algorithm for digitally signing firmware and software	NIST SP 800-208	All parameters approved for all classification levels.

Table II: CNSA 2.0 symmetric-key algorithms

Algorithm	Function	Specification	Parameters
Advanced Encryption Standard (AES)	Symmetric block cipher for information protection	FIPS PUB 197	Use 256-bit keys for all classification levels.
Secure Hash Algorithm (SHA)	Algorithm for computing a condensed representation of information	FIPS PUB 180-4	Use SHA-384 or SHA-512 for all classification levels.

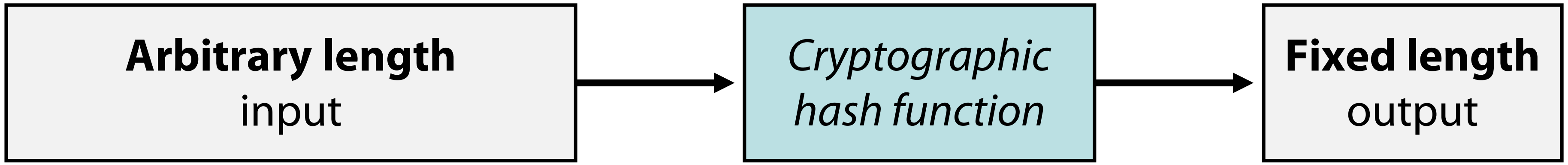
Table III: CNSA 2.0 quantum-resistant public-key algorithms

Algorithm	Function	Specification	Parameters
CRYSTALS-Kyber	Asymmetric algorithm for key establishment	TBD	Use Level V parameters for all classification levels.
CRYSTALS-Dilithium	Asymmetric algorithm for digital signatures	TBD	Use Level V parameters for all classification levels.

https://media.defense.gov/2022/Sep/07/2003071836/-1/-1/0/CSI_CNSA_2.0_FAQ_.PDF

Cryptographic Hash Functions

Hash functions that are considered practically impossible to invert



Properties of an ideal cryptographic hash function

- Easy to compute the hash value for any given message
- Infeasible to generate a message that has a given hash
- Infeasible to modify a message without changing the hash
- Infeasible to find two different messages with the same hash

Many-to-one function: *collisions can happen*

Cryptographic Hash Function Properties

Pre-image resistance

Given a hash value h, it should be computationally infeasible to find any input m such that $h = \mathbf{hash}(m)$

Example: break a hashed password

Second pre-image resistance

Given an input m_1, it should be computationally infeasible to find another input m_2 such that $m_1 \neq m_2$ and $\mathbf{hash}(m_1) = \mathbf{hash}(m_2)$

Example: forge an existing certificate

Collision Resistance

It should be computationally infeasible to find two different inputs m_1 and m_2 such that $\mathbf{hash}(m_1) = \mathbf{hash}(m_2)$ (collision)

Example: prepare two contradicting versions of a contract

Birthday Paradox

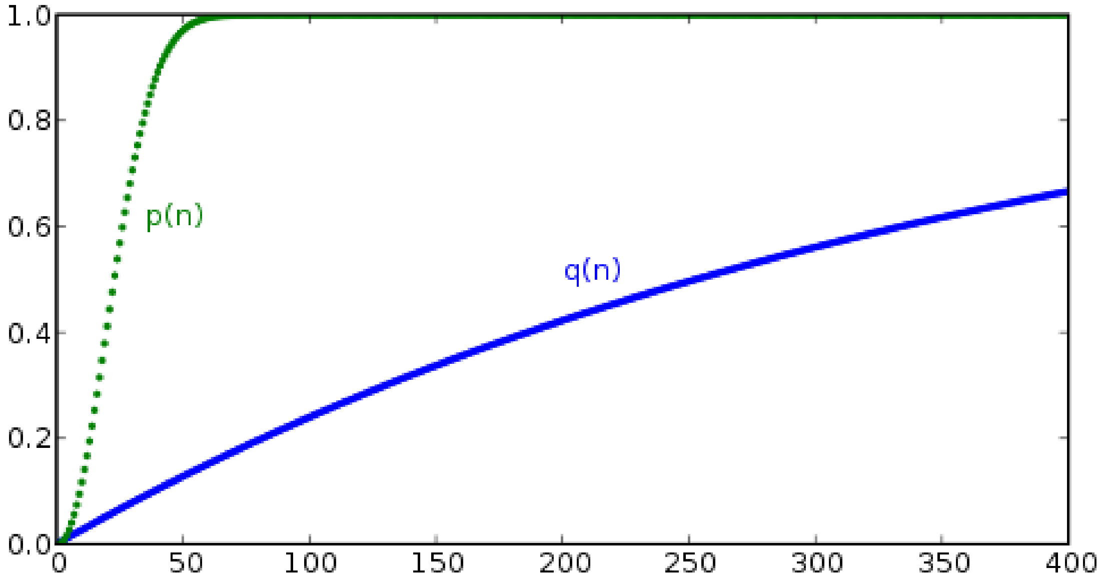
How many people does it take before the odds are 50% or better of having

… another person with the same birthday as *you*? **253**

Second pre-image resistance

… two people with the same birthday? **23**

Collision resistance



Uses of Cryptographic Hash Functions

Data integrity

Digital signatures

Message authentication

User authentication

Timestamping

Certificate revocation management

Common Hash Functions

MD5: 128-bit output

1993: Boer and Bosselaers, "pseudo-collision" in which 2 different IVs produce an identical digest

1996: Dobbertin, collision of the MD5 compression function

2004: Wang, Feng, Lai, and Yu, collisions for the full MD5

2005: Lenstra, Wang, and de Weger, construction of X.509 certs with different public keys but same hash

2008: Sotirov, Stevens, Appelbaum, Lenstra, Molnar, Osvik, de Wege, creation of rogue CA certificates

Use it? NO, it's unsafe

SHA-1: 160-bit output

2005: Rijmen and Oswald, attack on a reduced version of SHA1 (53 out of 80 rounds)

2005: Wang, Yao, and Yao, lowered the complexity for finding a collision to 2^{63}

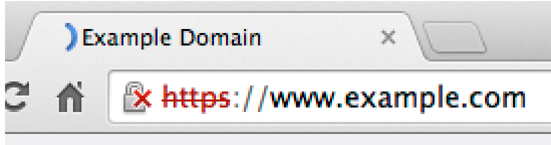
2006: Rechberger, attack with 2^{35} compression function evaluations

2015: Stevens, Karpman, and Thomas, freestart collision attack

2017: *SHAttered* attack, generated two different PDF files with the same SHA-1 hash

2020: Leurent and Peyrin, chosen-prefix collision attack with a complexity of $2^{63.4}$ (~45K USD per collision)

Use it? NO, use SHA-256 or better instead



TN Flame malware collision at

blogs.technet.com/b/srd/archive/2012/06/06/more-information-about-the-digital-certificates-used-to-sign-the-flame-malv

Microsoft

Sign in

Security Research and Defense Blog

Search TechNet with Bing

Search this blog   Search all blogs

Home | About | View More Blogs

TechNet Blogs » Security Research & Defense » Flame malware collision attack explained

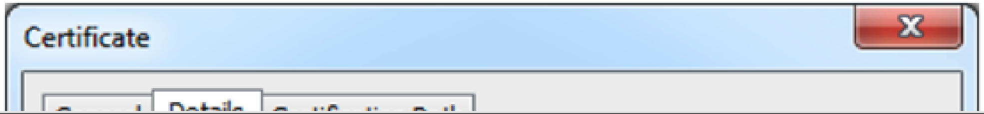
Flame malware collision attack explained

swiat | 6 Jun 2012 9:57 AM | 0

Since our last MSRC blog post, we've received questions on the nature of the cryptographic attack we saw in the complex, targeted malware known as Flame. This blog summarizes what our research revealed and why we made the decision to release Security Advisory 2718704 on Sunday night PDT. In short, by default the attacker's certificate would not work on Windows Vista or more recent versions of Windows. They had to perform a collision attack to forge a certificate that would be valid for code signing on Windows Vista or more recent versions of Windows. On systems that pre-date Windows Vista, an attack is possible without an MD5 hash collision. This certificate and all certificates from the involved certificate authorities were invalidated in Security Advisory 2718704. We continue to encourage all customers who are not installing updates automatically to do so immediately.

Mysterious Missing Extensions

When we first examined the Flame malware, we saw a file that had a valid digital signature that chained up to a Microsoft Root authority. As we reviewed this certificate, we noticed several irregularities. First, it had no X.509 extension fields, which was not consistent with the certificates we issued from the Terminal Server licensing infrastructure. We expected to find a Certificate Revocation List (CRL) Distribution Point (CDP) extension, an Authority Information Access (AIA) extension, and a "Microsoft Hydra" critical extension. All of these were absent.



Share Article

Follow Us

RSS for Posts

@msftsecresponse

Security Newsletter

Report a Vulnerability

SHA-1 Deprecation Updat

https://blogs.windows.com/msedgedev/2015/11/04/sha-1-deprecation-update/

Windows

RSS

Search ...

MICROSOFT EDGE DEV BLOG DEVELOPER RESOURCES PLATFORM CHANGELOG DEMOS

Translate with bing

NOVEMBER 4, 2015 10:00 AM

SHA-1 Deprecation Update

By Kyle Pflug / Program Manager, Microsoft Edge

In a previous update on TechNet, we announced that Windows will block SHA-1 signed TLS certificates starting on January 1, 2017. In light of recent advances in attacks on the SHA-1 algorithm, we are now considering an accelerated timeline to deprecate SHA-1 signed TLS certificates as early as June 2016.

Mozilla recently announced a similar intent on the Mozilla Security Blog. We will continue to coordinate with other browser vendors to evaluate the impact of this timeline based on telemetry and current projections for feasibility of SHA-1 collisions.

For more details on our schedule, please see Windows Enforcement of Authenticode Code Signing and Timestamping on Technet, or

RELATED POSTS

HTTP Strict Transport Security comes to Internet Explorer 11 on Windows 8.1 and Windows 7
Read more

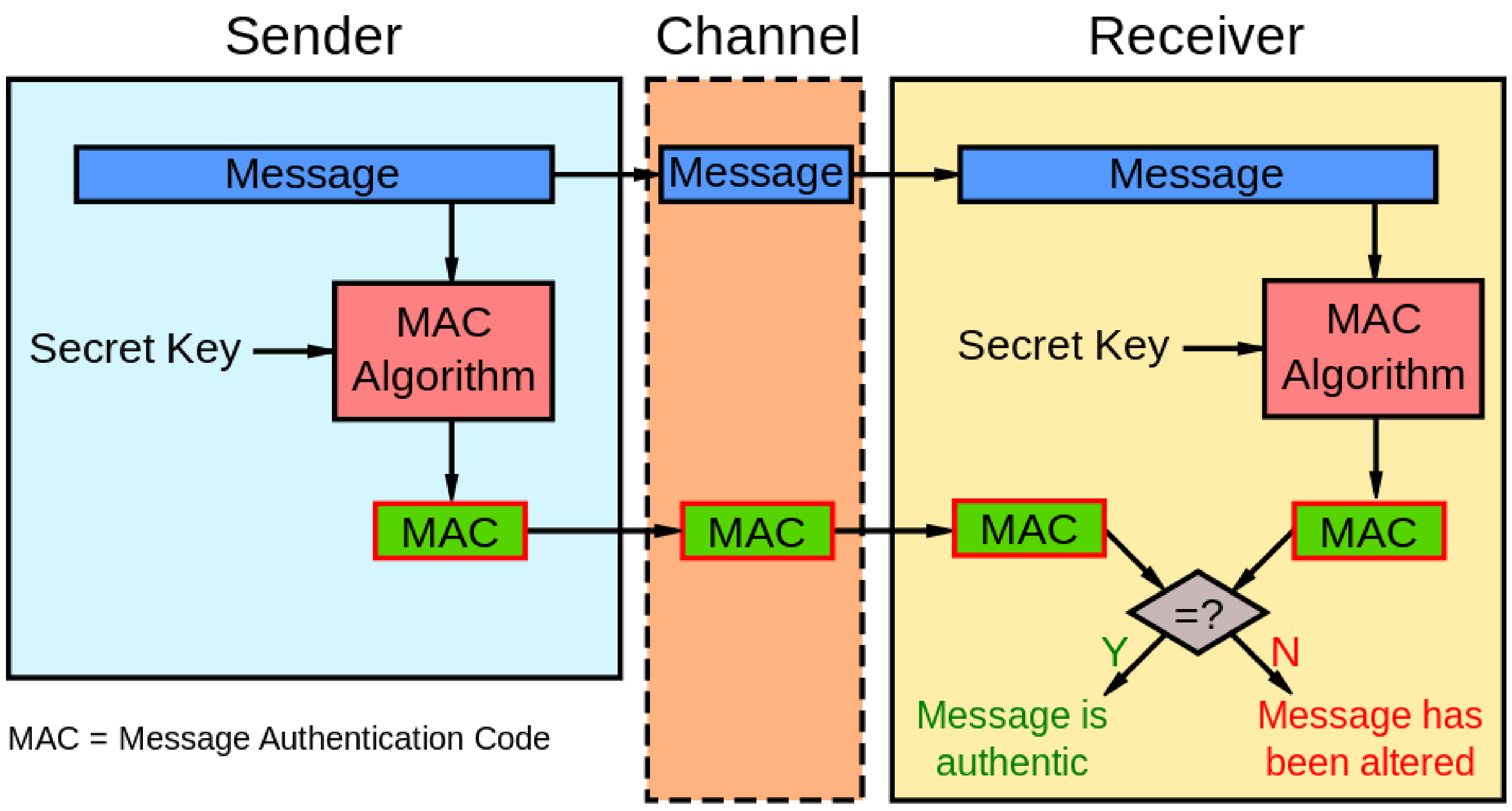
Ending support for the RC4 cipher in Microsoft Edge and Internet Explorer 11
Read more

How Microsoft Edge and Internet Explorer 11 on Windows 10 work better together in the Enterprise
Read more

Protecting Microsoft Edge against binary injection
Read more

Message Authentication Codes (MACs) (AKA authentication “tag”)

Verify both message *integrity* and *authenticity*



https://commons.wikimedia.org/wiki/File:Message_Authentication_Code.svg

Why Are MACs Needed?

Tampering

Any modification to the message will result in a different MAC value

Recipient will detect the tampering because the authentication check will fail

Forgery/Impersonation

MACs are generated using a secret key: only authorized parties in possession of the key can generate a valid MAC

Replay

An attacker could capture a message from Alice to Bob and replay it later pretending to be Alice

Protocols that use MACs include a timestamp or sequence number in each message, which makes it impossible for an attacker to replay an old message

MAC = H(*key* ‖ *message*)

|| denotes concatenation

Problem: easy to append data to the message without knowing the key and obtain another valid MAC

Length-extension attack: calculate H(m_1 ‖ m_2) for an attacker-controlled m_2 given only H(m_1) and the length of m_1

Keyed-hash message authentication code (HMAC)

$$\text{HMAC} = H(\ (K \oplus opad)\ \|\ H(K \oplus ipad \,\|\, m)\ )$$

opad/ipad: outer/inner padding

Impossible to generate the HMAC of a message without knowing the secret key

Double nesting prevents various forms of length-extension attacks

Order of Encryption and MACing

Encrypted data usually must be protected with a MAC

- Encryption alone protects only against passive adversaries

Different options:

MAC-and-Encrypt $E(P) \,||\, M(P)$

- No integrity of the ciphertext

MAC-then-Encrypt $E(P \,||\, M(P))$

- No integrity of the ciphertext (have to decrypt it first)

Encrypt-then-MAC $E(P) \,||\, M(E(P))$

- Provides integrity of the ciphertext
- **Preferable option – *always MAC the ciphertext***

Digital Signatures

Use RSA backwards:

- *Sign* (encrypt) with the private key
- *Verify* (decrypt) with the public key

Ownership of a private key turns it into a *digital signature*

- Anyone can verify that a message was signed by its owner ➔ *Non-repudiation*

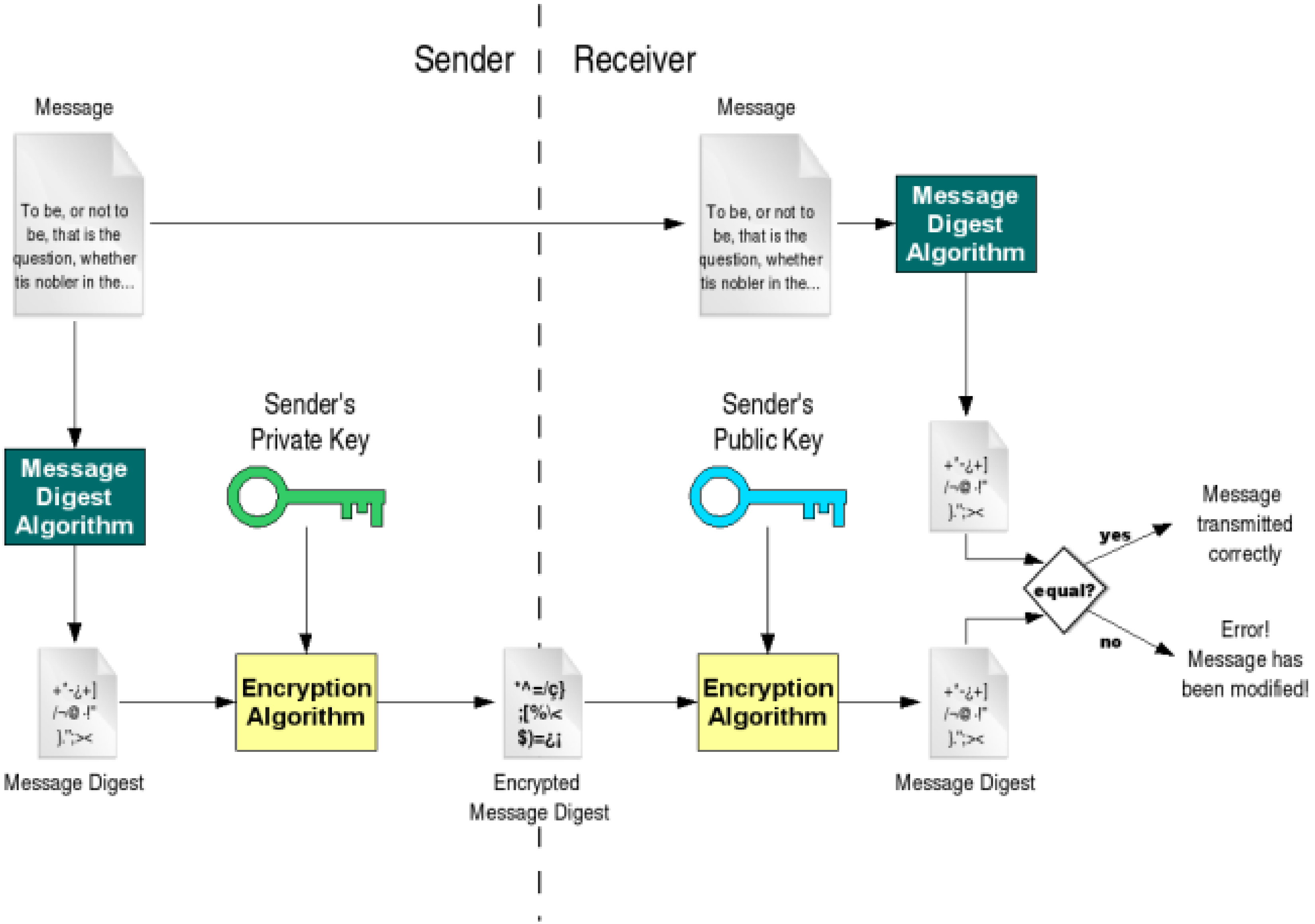
Again, too expensive to sign the whole message

- Calculate a cryptographic hash of the message and then sign the hash

What if a private key was stolen or deliberately leaked?

- All signatures (past and future) of that signer become suspect
- The signer might know which signatures were issued legitimately, but there is no way for the verifier to distinguish between them

Digital Signatures



Hashes vs. **MACs** vs. **Digital Signatures**

	Hash	**MAC**	**Signature**
Integrity	✓	✓	✓
Authentication		✓	✓
Non-repudiation			✓
Keys	*None*	*Symmetric*	*Asymmetric*

Public Key Authenticity

Authentication without confidence in the keys used is pointless

Need to obtain evidence that a given public key is authentic

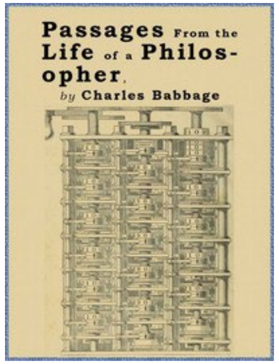
- It is correct and belongs to the person or entity claimed
- Has not been tampered with or replaced by an attacker

Different ways to establish trust (future lecture)

- **TOFU:** trust on first use (e.g., SSH)
- **Web of trust:** decentralized trust model (e.g., PGP)
- **PKI:** public key infrastructure (e.g., TLS)

Never Roll Your Own Crypto

"One of the most singular characteristics of the art of deciphering is the strong conviction possessed by every person, even moderately acquainted with it, that he is able to construct a cipher which nobody else can decipher. I have also observed that the cleverer the person, the more intimate is his conviction."



— Charles Babbage

Anyone can create an algorithm that they themselves can't break

What is hard is creating an algorithm that no one else can break

Even after years of analysis by experts

https://www.schneier.com/blog/archives/2011/04/schneiers_law.html

Adi Shamir: *Crypto is typically bypassed, not penetrated*