CSE331 Computer Security Fundamentals

9/28/2017

Public Key Cryptography

Michalis Polychronakis

Stony Brook University

Public Key Cryptography

Many algorithms with different purposes

- One common property: pair of values, 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

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

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

Diffie–Hellman Key Exchange

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

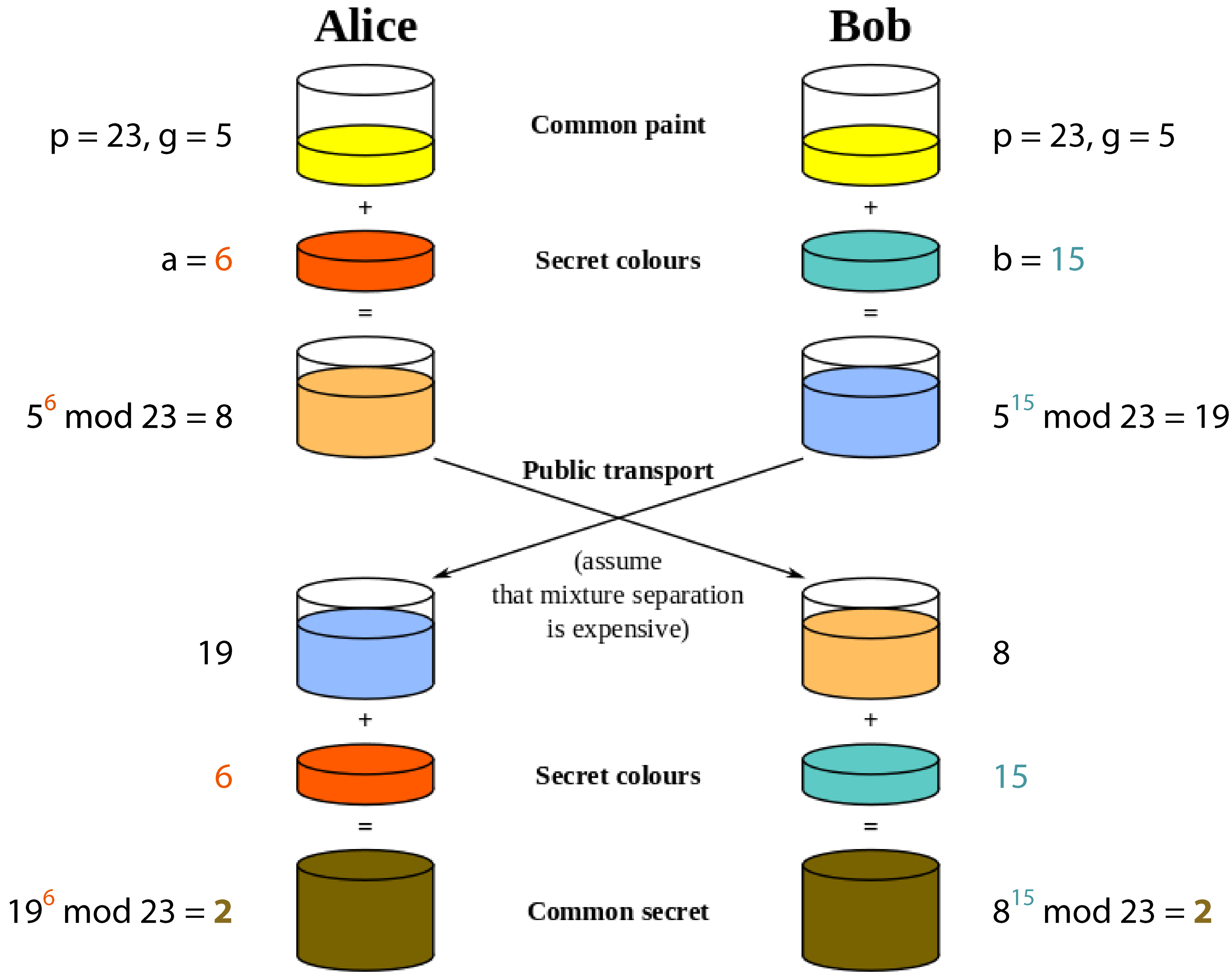
p is usually of the form $2q+1$ where q is also prime

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 (private) large random number a and sends to Bob $g^a \bmod p$

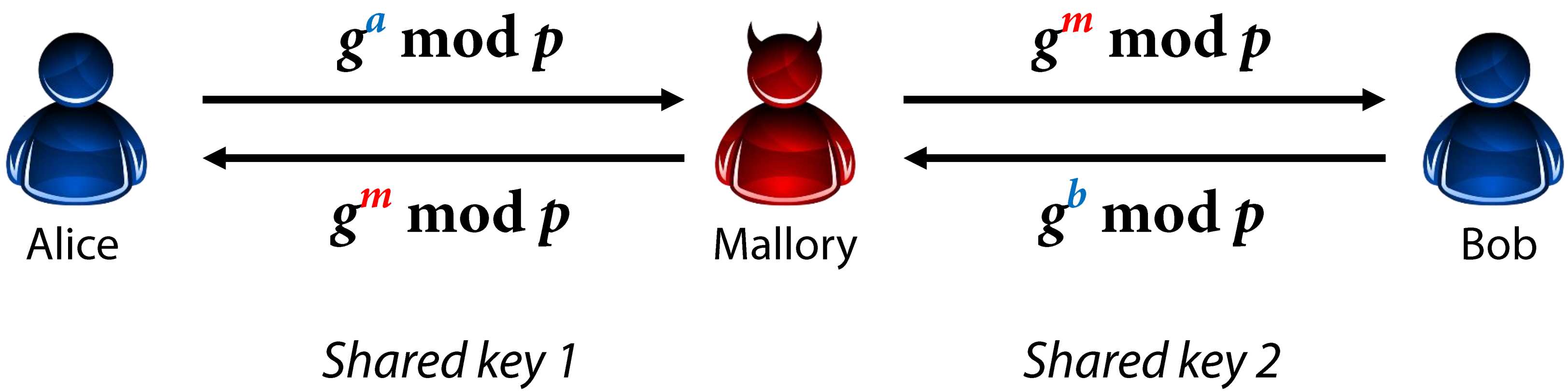
Bob picks a secret large random number b and sends to Alice $g^b \bmod p$

$$\left.\begin{array}{l} \text{Alice calculates } s = (g^b \bmod p)^a = g^{ba} \bmod p \\[1em] \text{Bob calculates } s = (g^a \bmod p)^b = g^{ab} \bmod p \end{array}\right\} \textit{shared key}$$


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

Man-in-the-Middle Attack

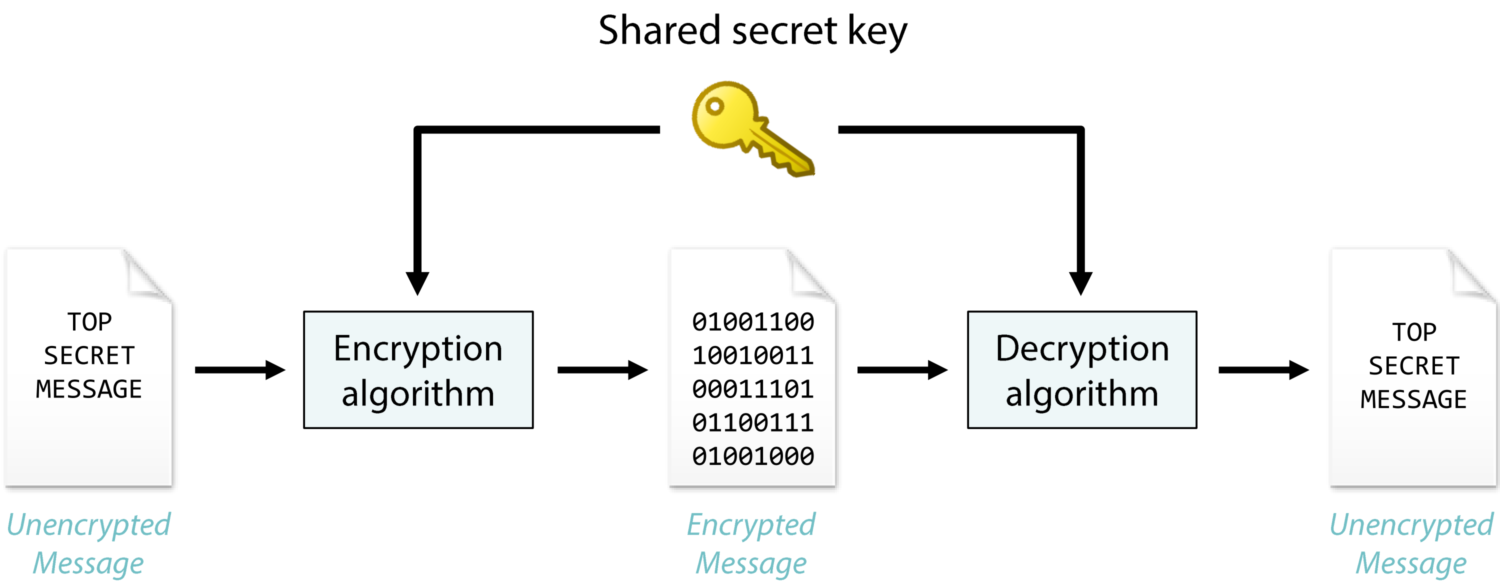
Alice and Bob share no secrets



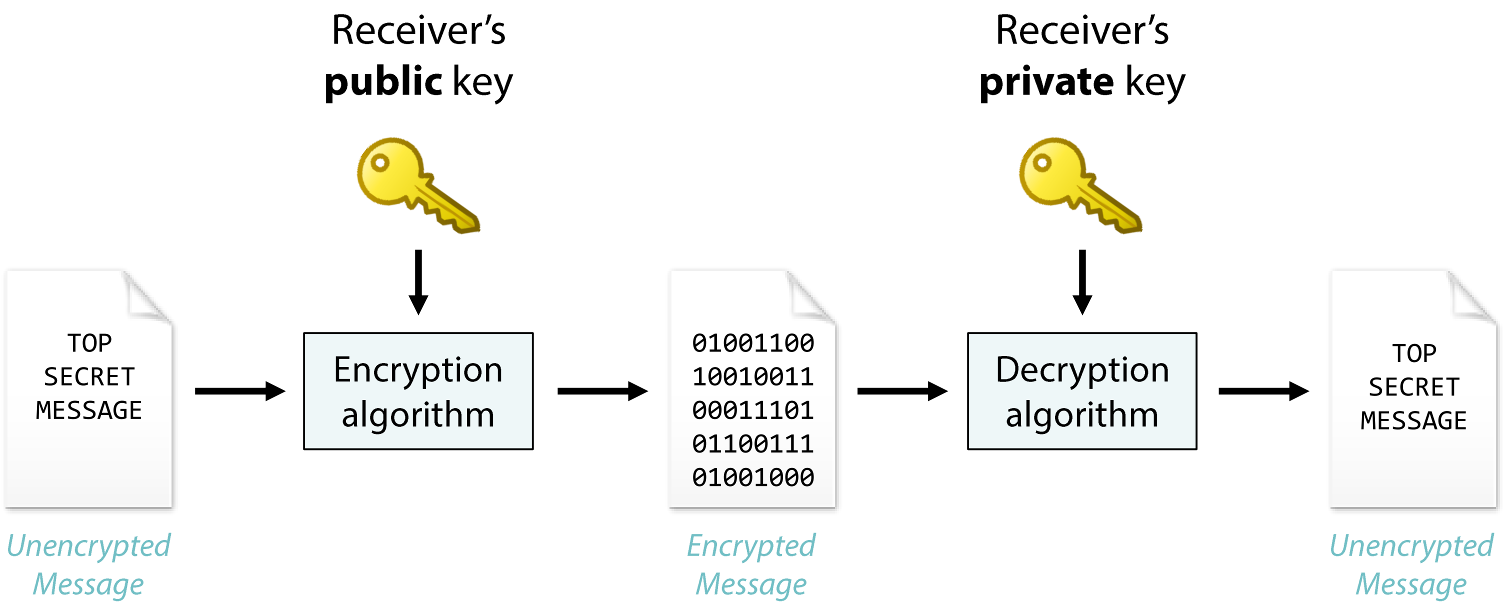
Mallory actively decrypts and re-encrypts all traffic

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

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 n parties, n key pairs are needed (instead of $n(n-1)/2$ shared keys)

Provides both secrecy *and authenticity*

Disadvantages

More computationally intensive

- Encryption/decryption operations are 2–3 orders of magnitude slower than symmetric key primitives

About one order of magnitude larger keys

Key generation is more difficult

RSA

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 $\boldsymbol{p}$ and $\boldsymbol{q}$
- No efficient methods are known to factor their product $\boldsymbol{N}$

Variable key length

- Largest (publicly known) factored RSA modulus is 768-bit long
- That was in 2009: it took 2 years and many hundreds of machines
- 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

- For now (next decade or two)…

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 ➔ use RSA in combination with a symmetric key

Sending an encrypted message:

- Encrypt message with a random symmetric key
- Encrypt the symmetric key with recipient's public key
- Transmit both the encrypted message and the encrypted 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, use in the future

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

FS: Ensure that even if current keys are compromised, past encrypted traffic cannot be compromised

- Generate random 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)

Note 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

- E.g., a 256-bit elliptic curve public 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

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 $h = \text{hash}(m)$

Example: break a hashed password

Second pre-image resistance

Given m_1 it should be computationally infeasible to find m_2 such that $m_1 \neq m_2$ and $\text{hash}(m_1) = \text{hash}(m_2)$

Example: forge an existing certificate

Collision Resistance

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

Example: prepare two contradicting versions of a contract

Birthday Paradox

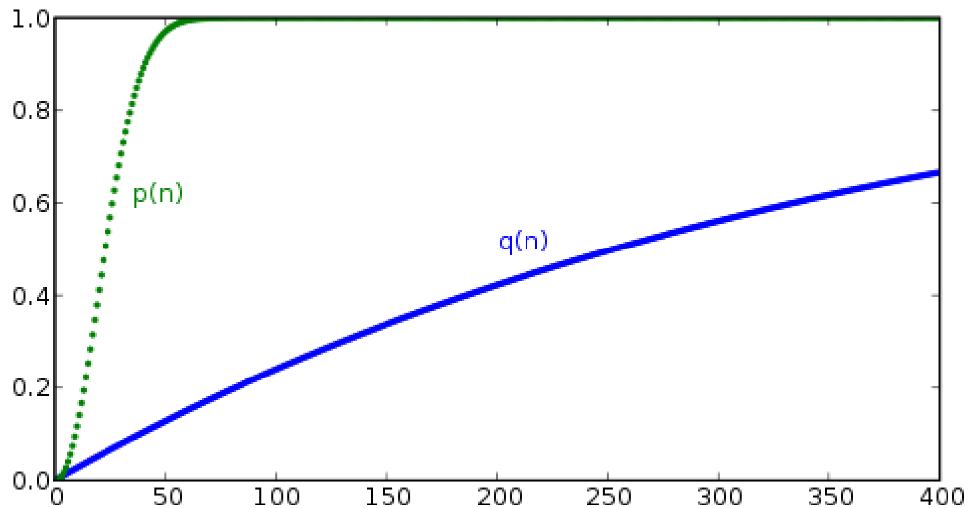
How many people does it take before the odds are 50% or better of having…

…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" of the MD5 compression function: 2 different IVs which 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 two X.509 certificates with different public keys but same hash

2008: Sotirov, Stevens, Appelbaum, Lenstra, Molnar, Osvik, de Wege, creating 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, an improvement, lowering the complexity for finding a collision to 2^{63}

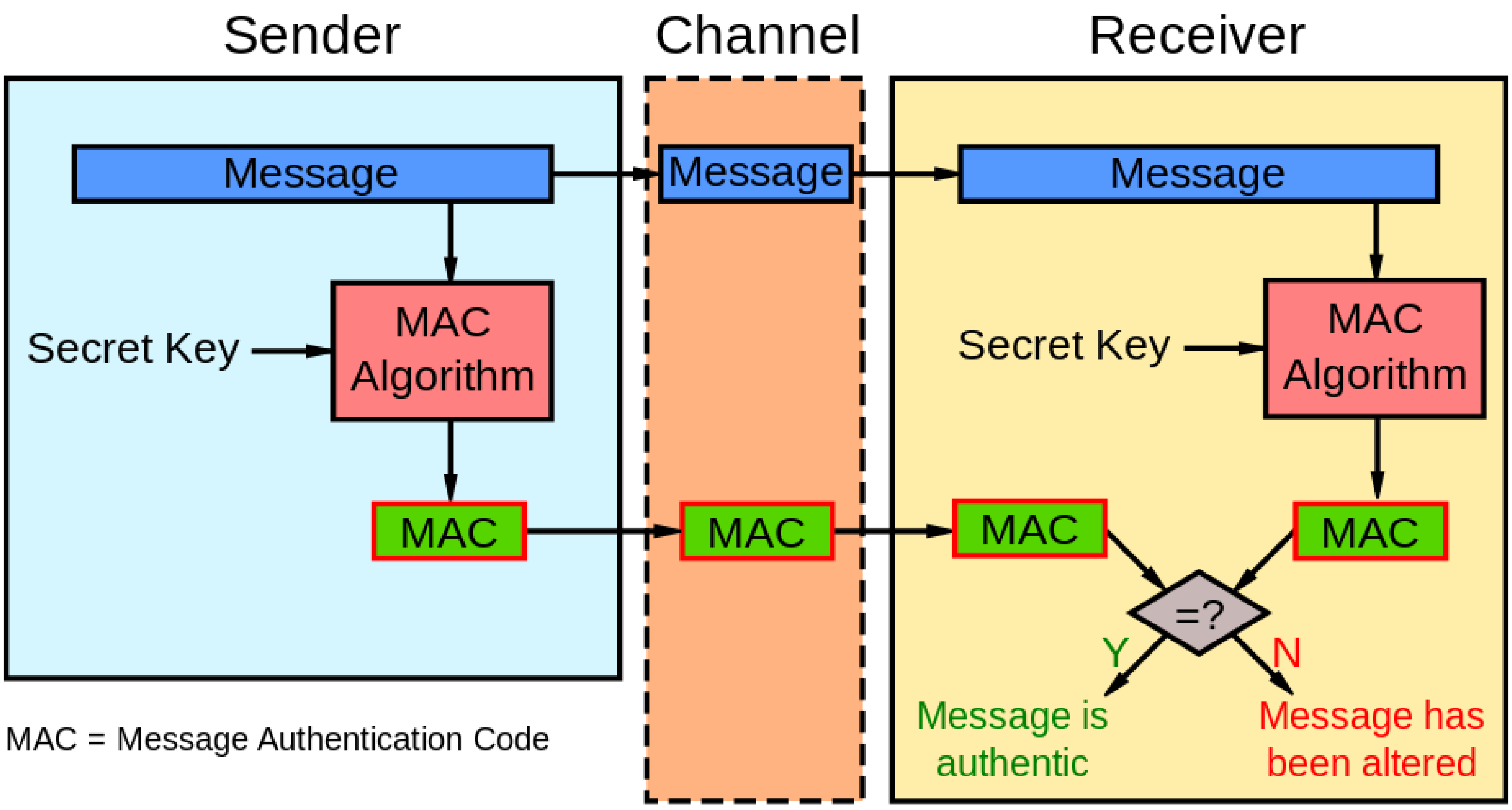
2006: Rechberger, attack with 2^{35} compression function evaluations

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

Use it? Use SHA-256 or better instead

Message Authentication Codes (MACs)

Verify both message integrity and authenticity



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

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)

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

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 $E(P) \| M(E(P))$

- **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*

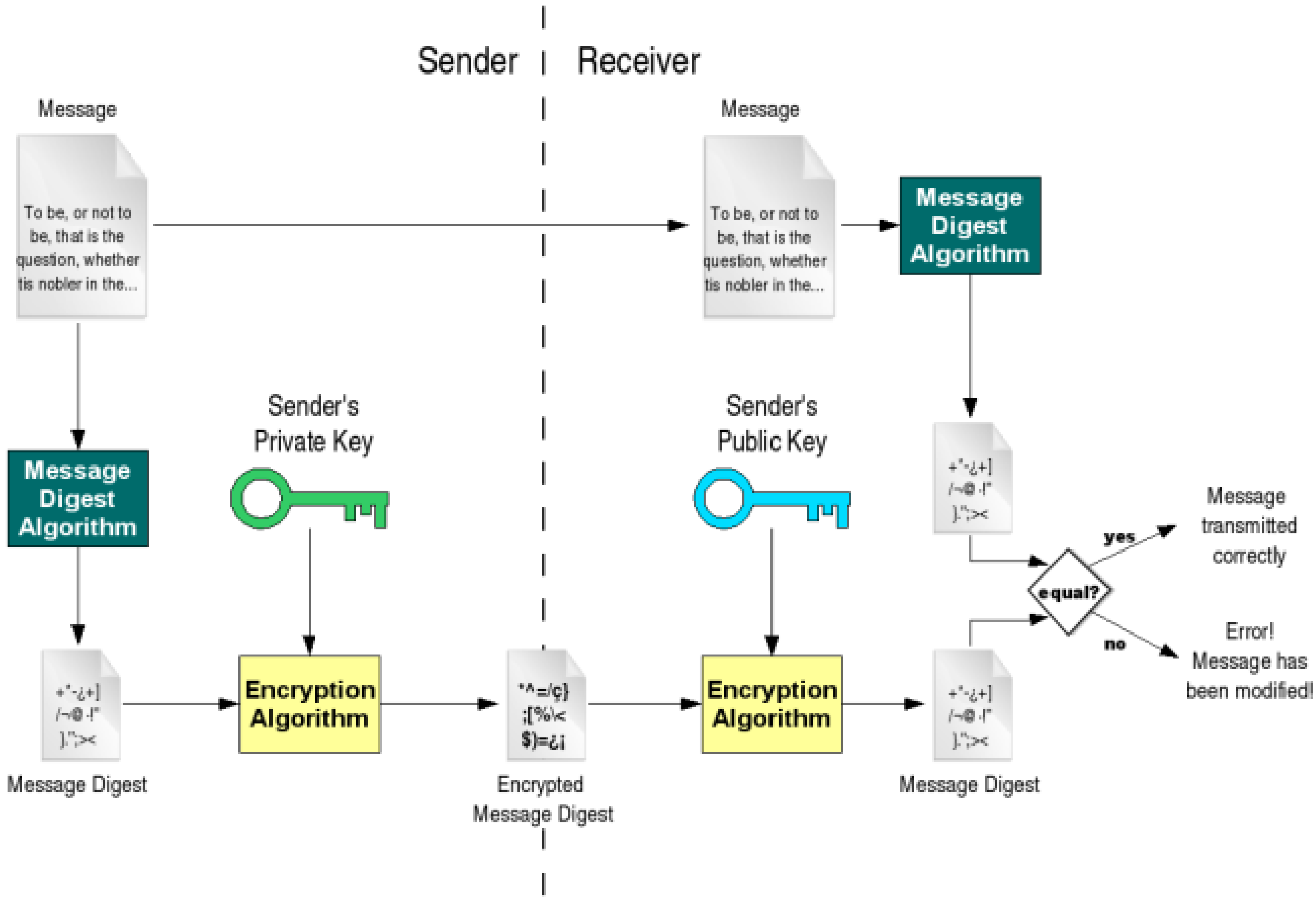
Again, too expensive to sign the whole message

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

What if a private key was stolen or deliberately leaked?

- All the 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 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 gain confidence or proof that a particular 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

- TOFU: trust on first use (e.g., SSH)
- Web of trust – decentralized trust model (e.g., PGP)
- PKI: public key infrastructure (e.g., SSL)
- (subject of future lecture)

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