Cryptography Part 2

CMSC 23200/33250, Winter 2023, Lecture 10

David Cash & Blase Ur

University of Chicago

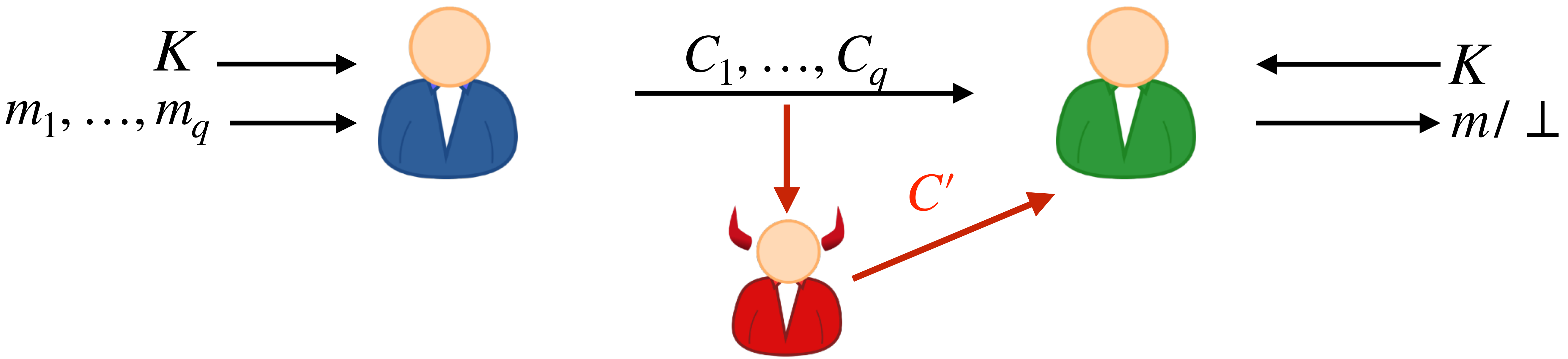
Outline

- Message Authentication
- Hash Functions
- Public-Key Encryption
- Digital Signatures

Outline

- **Message Authentication**
- Hash Functions
- Public-Key Encryption
- Digital Signatures

Adversary Goal #2: Break Authenticity



The adversary sees ciphertexts and attempts to create and inject a new ciphertext without being detected by receiver.

Other attack settings are important here too.

Stream ciphers do not give integrity

M = please pay ben 20 bucks

C = b0595fafd05df4a7d8a04ced2d1ec800d2daed851ff509b3e446a782871c2d

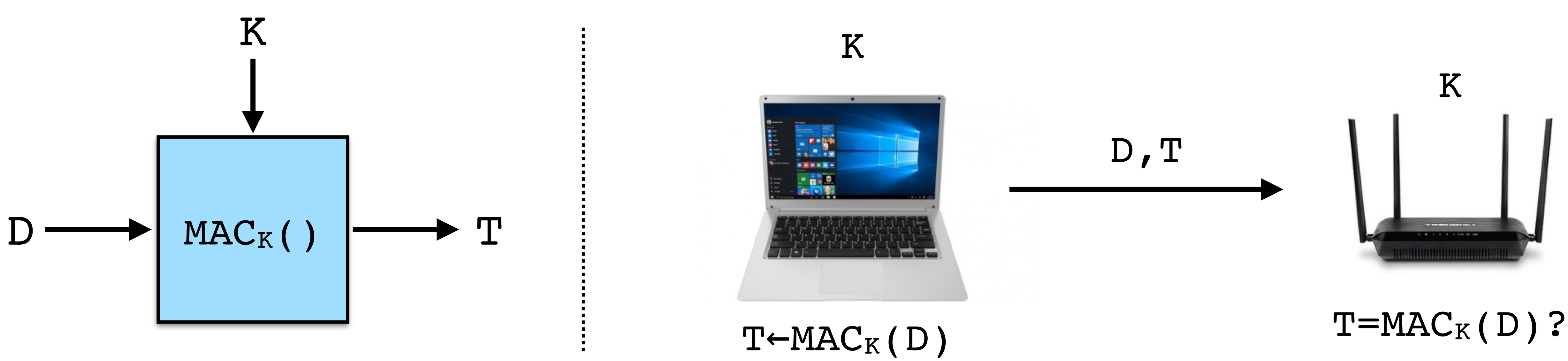
C′= b0595fafd05df4a7d8a04ced2d1ec800d2daed851ff509b3e546a782871c2d

M′ = please pay ben 21 bucks

Inherent to stream-cipher approach to encryption.

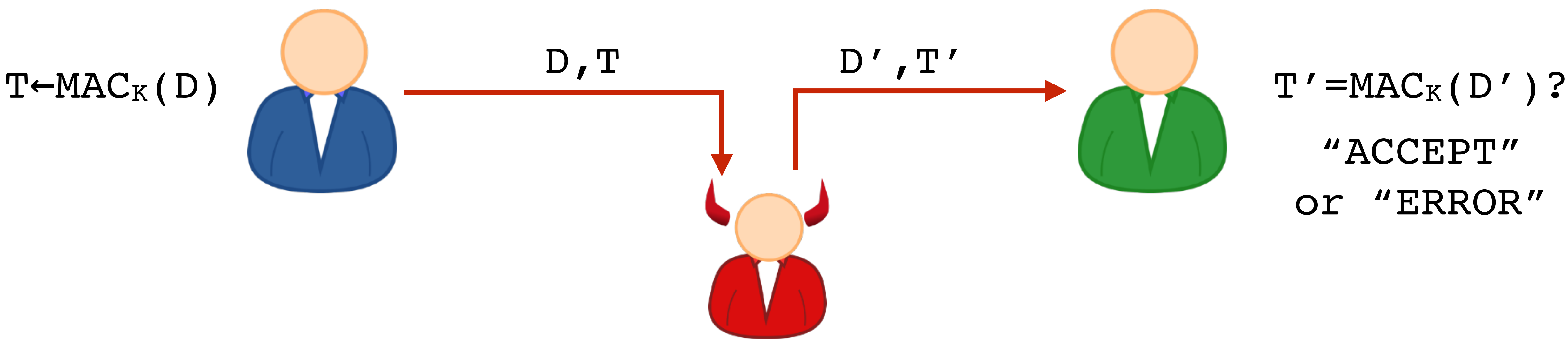
Message Authentication Codes

A **message authentication code (MAC)** is an algorithm that takes as input a key and a message, and outputs an “unpredictable” **tag.**



D will usually be a ciphertext, but is often called a “message”.

MAC Security Goal: Unforgeability



MAC satisfies **unforgeability** if it is infeasible for Adversary to fool Bob into accepting `D'` not previously sent by Alice.

MAC Security Goal: Unforgeability

Note: No encryption on this slide.

`D = please pay ben 20 bucks`

`T = 827851dc9cf0f92ddcdc552572ffd8bc`

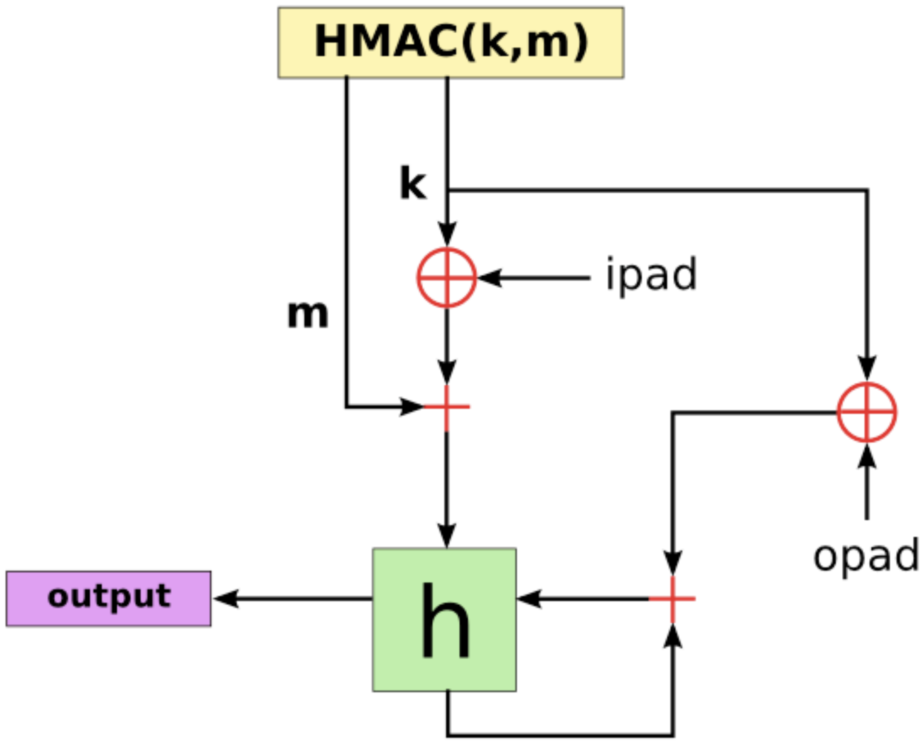
`D'= please pay ben 21 bucks`

`T'= baeaf48a891de588ce588f8535ef58b6`

Should be hard to predict **T'** for any new **D'**.

MACs In Practice: Use HMAC or Poly1305-AES

- More precisely: Use HMAC-SHA2. More on hashes and MACs in a moment.



- Other, less-good option: AES-CBC-MAC (bug-prone)

Authenticated Encryption

Encryption that provides **confidentiality** and **integrity** is called **Authenticated Encryption**.

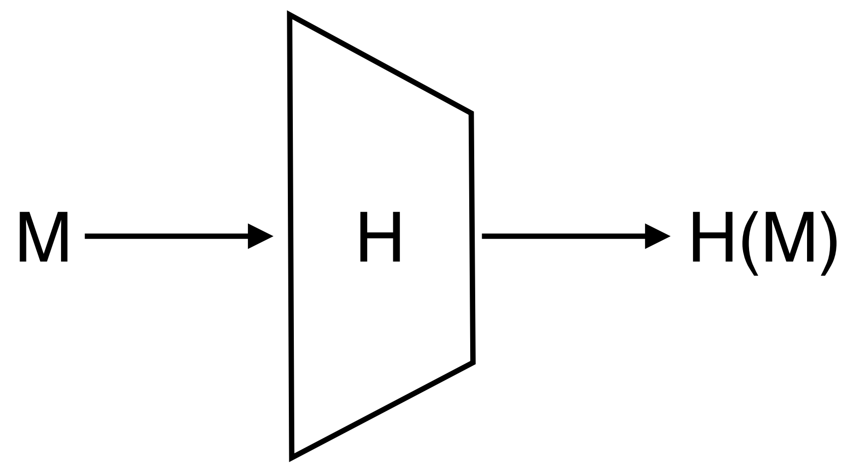
- Built using a good stream cipher and a MAC.
  - Ex: Salsa20 with HMAC-SHA2
- Best solution: Use ready-made Authenticated Encryption
  - Ex: AES-GCM is the standard

Outline

- Message Authentication
- **Hash Functions**
- Public-Key Encryption
- Digital Signatures

Next Up: Hash Functions

Definition: A hash function is a deterministic function H that reduces arbitrary strings to fixed-length outputs.

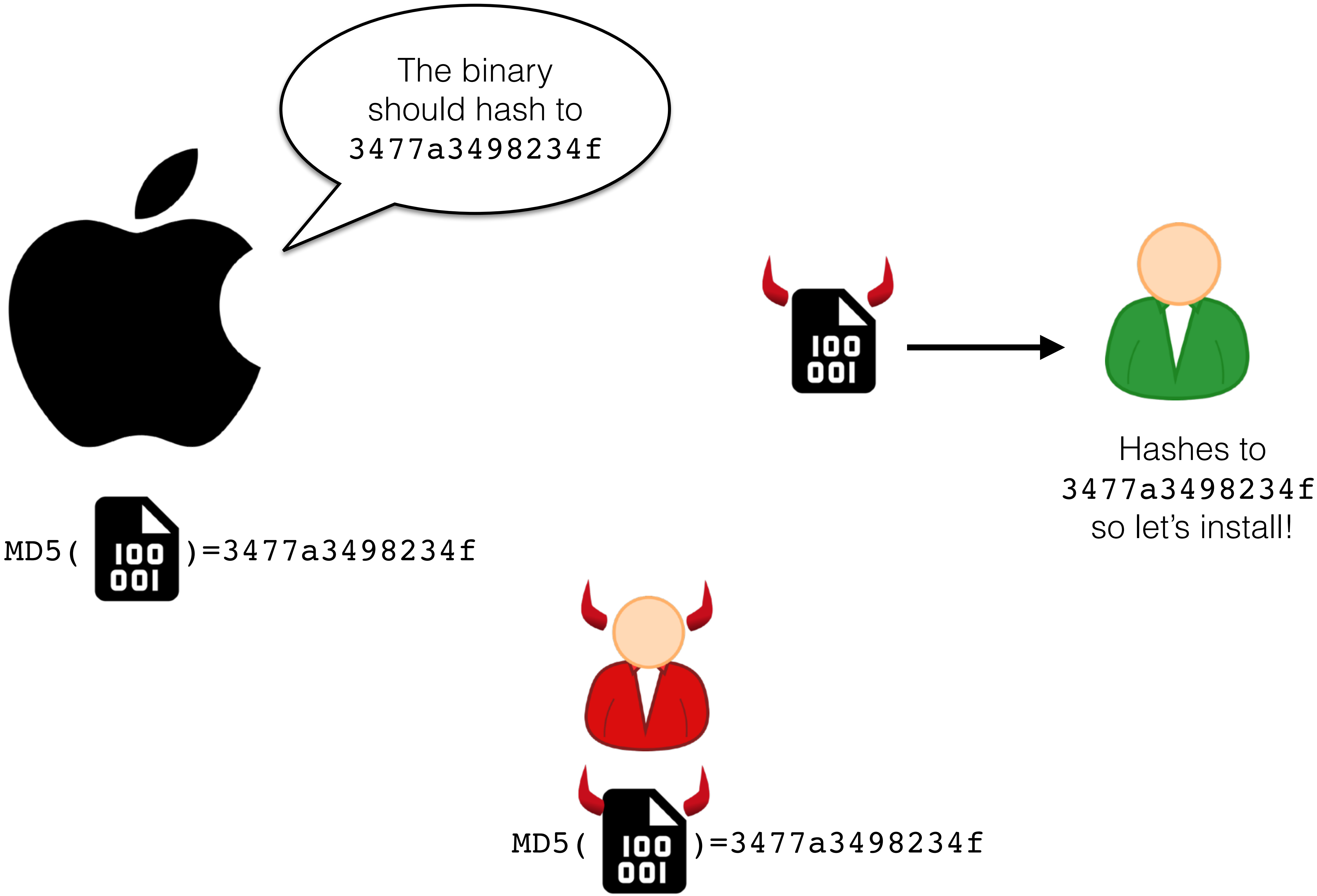


Some security goals:

- collision resistance: can't find M != M' such that H(M) = H(M')
- preimage resistance: given H(M), can't find M
- second-preimage resistance: given H(M), can't find M' s.t. H(M') = H(M)

Note: Very different from hashes used in data structures!

Why are collisions bad?

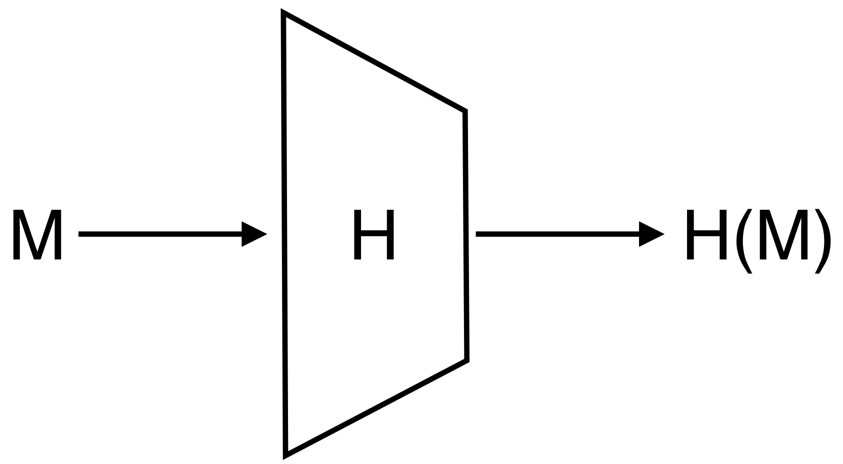


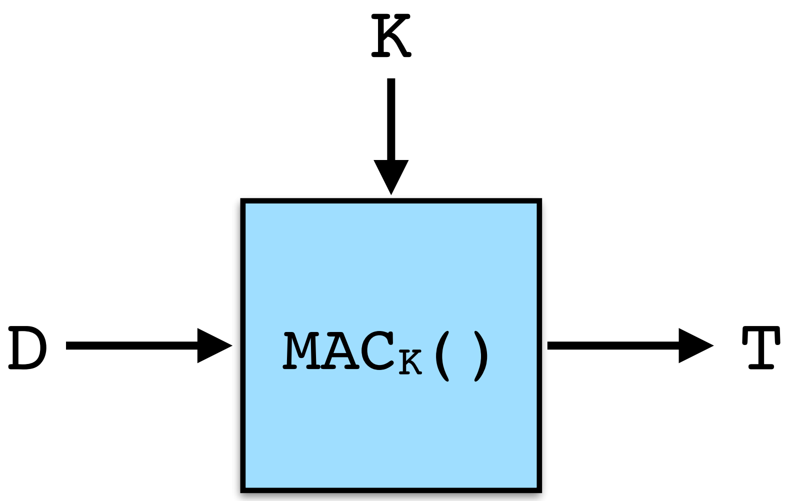
Practical Hash Functions

Name	Year	Output Len (bits)	Broken?
MD5	1993	128	Super-duper broken
SHA-1	1994	160	Yes
SHA-2 (SHA-256)	1999	256	No
SHA-2 (SHA-512)	2009	512	No
SHA-3	2019	>=224	No

Confusion over “SHA” names leads to vulnerabilities.

Hash Functions are not MACs





Both map long inputs to short outputs… but a hash function does not take a key.

Intuition: a MAC is like a hash function, that only the holders of key can evaluate.

MACs from Hash Functions

Goal: Build a secure MAC out of a good hash function.

Construction: MAC(K, D) = H(K || D) **Warning: Broken**

- Totally insecure if H = MD5, SHA1, SHA-256, SHA-512
- May be secure with SHA-3 (but don't do it)

Construction: MAC(K, D) = H(D || K) **Just don't**

Upshot: Use HMAC; It's designed to avoid this and other issues.

Later: Hash functions and certificates

Length Extension Attack

Construction: MAC(K, D) = H(K || D) **Warning: Broken**

Adversary goal: Find new message D’ and a valid tag T’ for D’

Need to find: Given T=H(K || D), find T’=H(K || D’) without knowing K.

In Assignment 4: Break this construction!

Outline

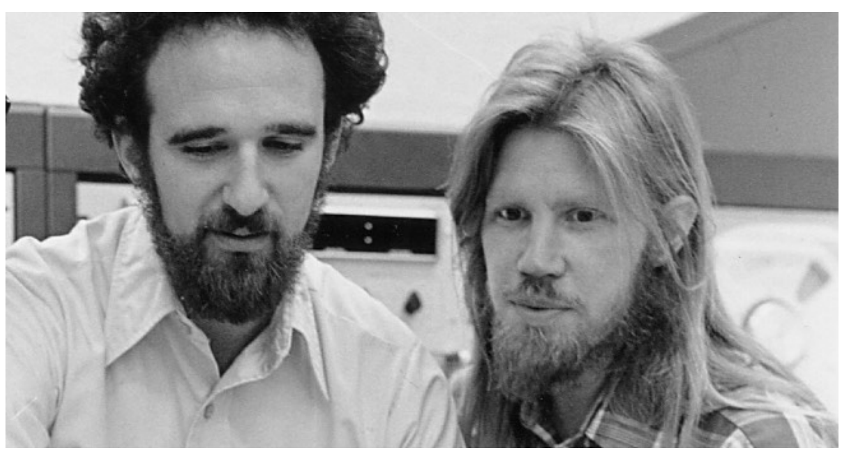
- Message Authentication
- Hash Functions
- **Public-Key Encryption**
- Digital Signatures

The Seed of Public-Key Cryptography

Basic question: If two people are talking in the presence of an eavesdropper, and they don't have pre-shared a key, is there any way they can send private messages?

The Seed of Public-Key Cryptography

Basic question: If two people are talking in the presence of an eavesdropper, and they don't have pre-shared a key, is there any way they can send private messages?

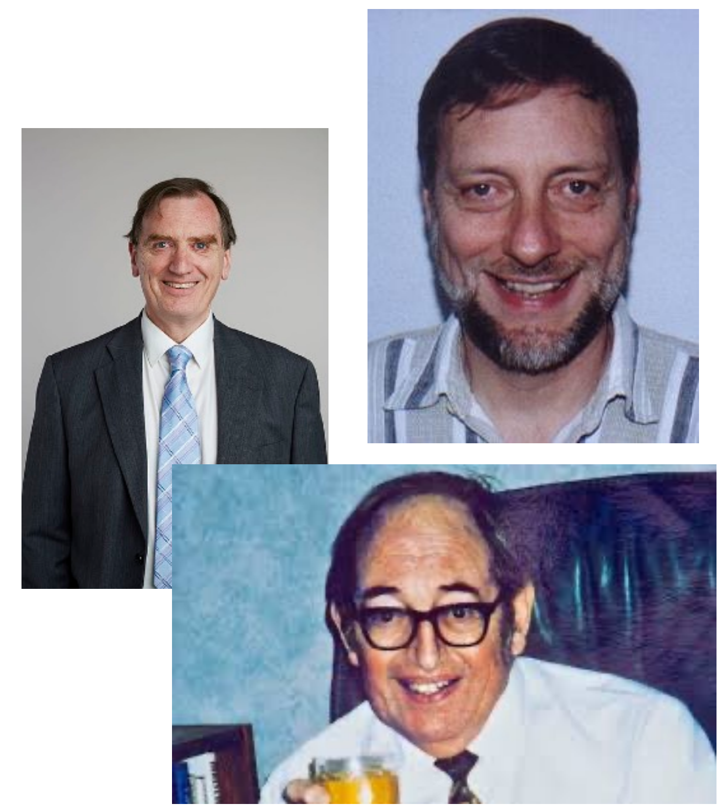


Diffie and Hellman
in 1976: **Yes!**

Turing Award, 2015,
+ Million Dollars

Rivest, Shamir, Adleman
in 1978: **Yes, differently!**

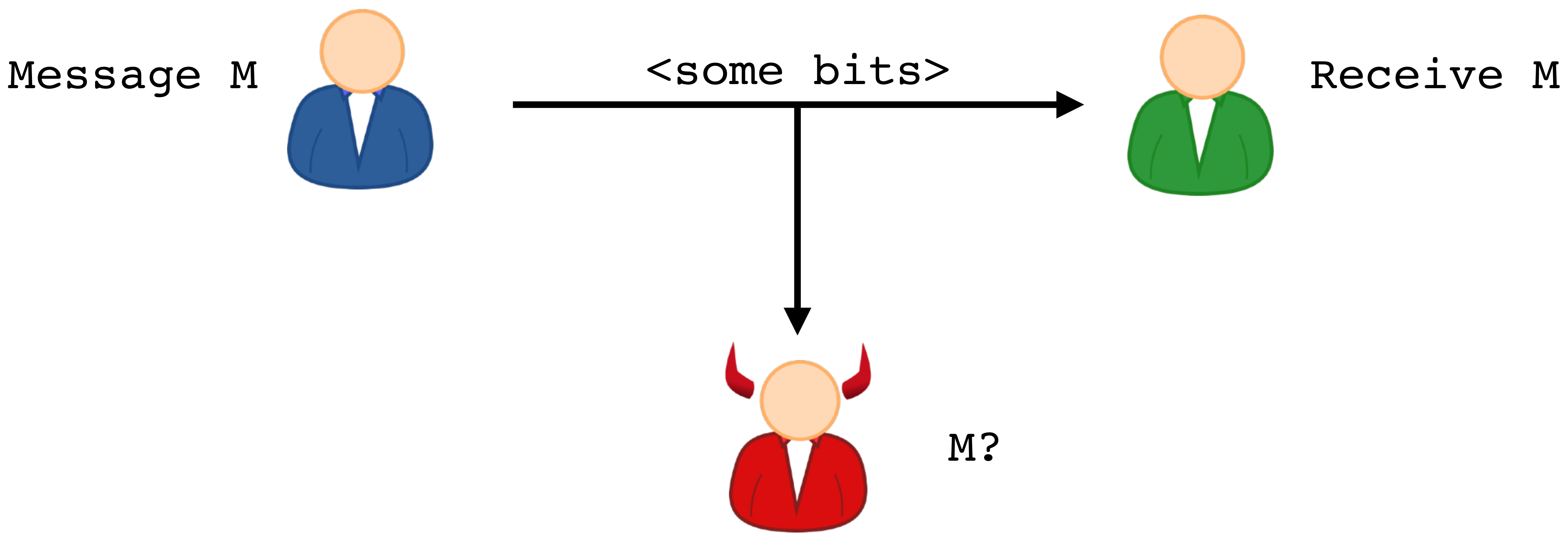
Turing Award, 2002,
+ no money



Cocks, Ellis, Williamson
in 1969, at GCHQ:
Yes…

The Seed of Public-Key Cryptography

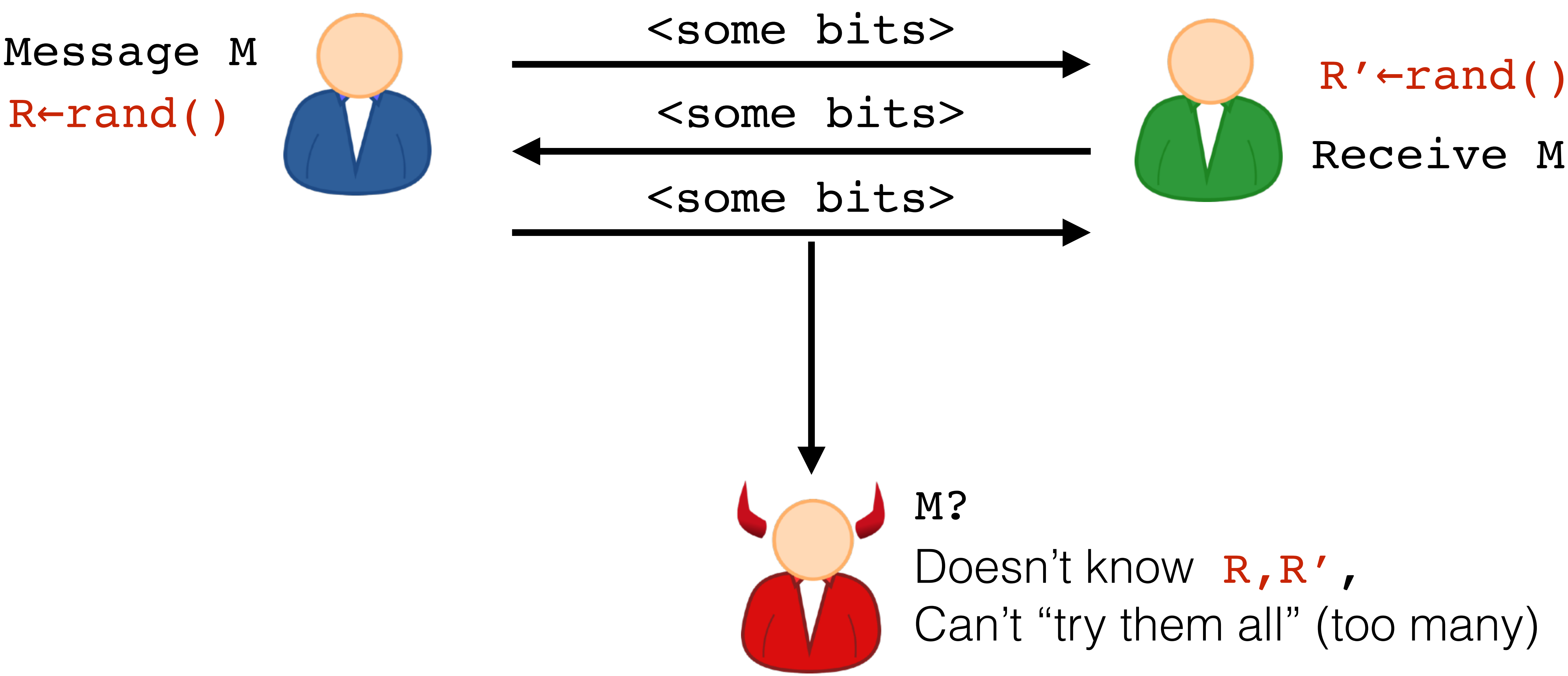
Basic question: If two people are talking in the presence of an eavesdropper, and they don't have pre-shared a key, is there any way they can send private messages?



Formally impossible (in some sense):
No difference between receiver and adversary.

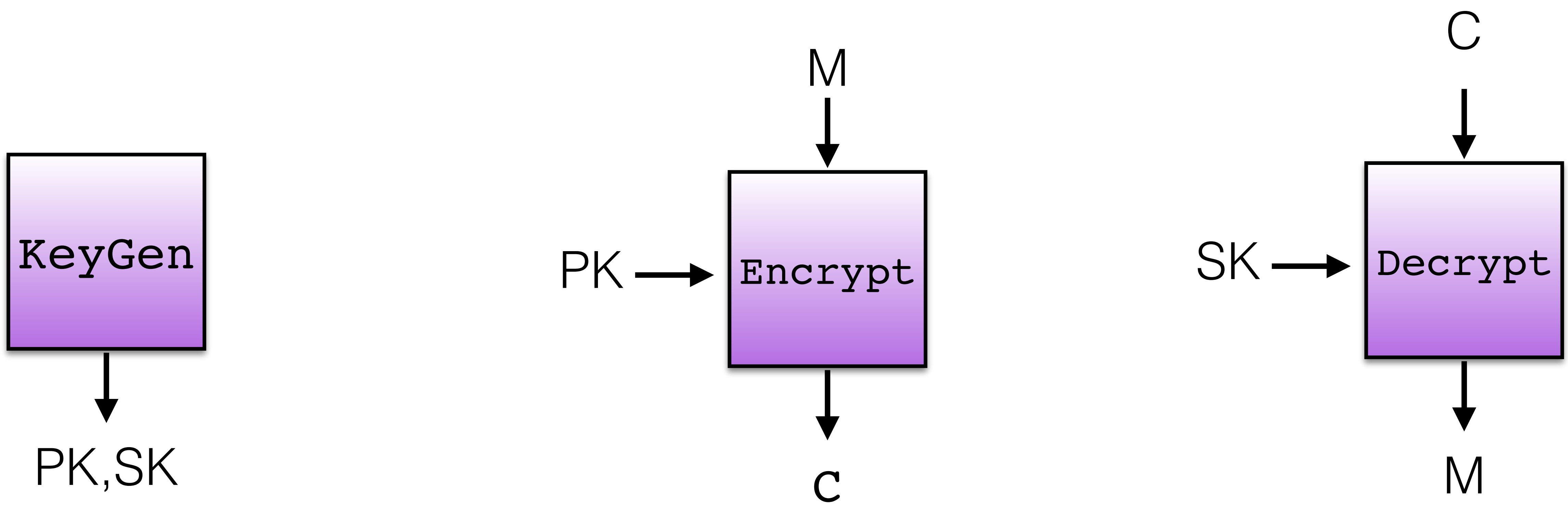
The Seed of Public-Key Cryptography

Basic question: If two people are talking in the presence of an eavesdropper, and they don't have pre-shared a key, is there any way they can send private messages?



Public-Key Encryption Schemes

A public-key encryption scheme consists of three algorithms **KeyGen**, **Encrypt**, and **Decrypt**

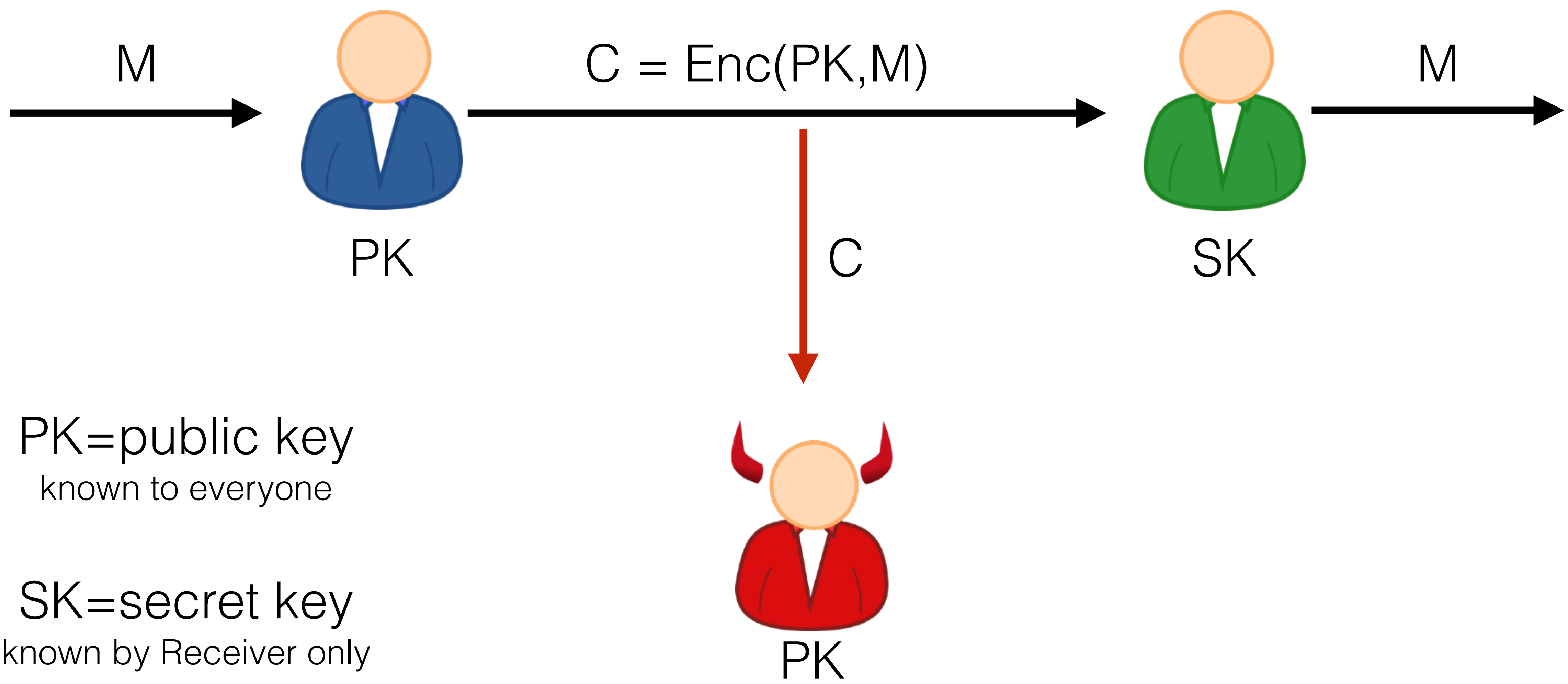


KeyGen: Outputs two keys. PK published openly, and SK kept secret.

Encrypt: Uses PK and M to produce a ciphertext C.

Decrypt: Uses SK and C to recover M.

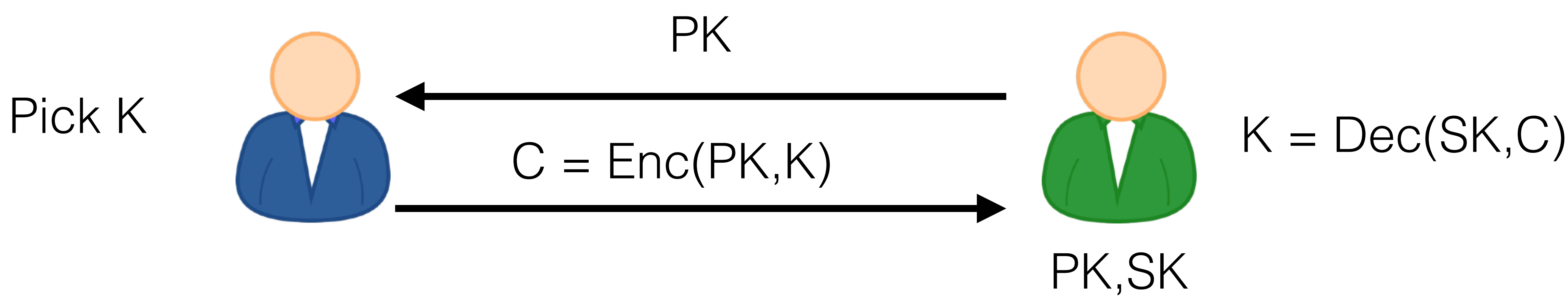
Public-Key Encryption in Action



PK=public key
known to everyone

SK=secret key
known by Receiver only

Establishing a Shared Key



- This and similar ideas used in SSH, TLS, etc

```
davidcash@hofbraeuhaus:~|⇒  ssh cs232-33.c.cs.uchicago.edu
The authenticity of host 'cs232-33.c.cs.uchicago.edu (128.135.37.172)' can't be established.
ED25519 key fingerprint is SHA256:hw3ERhLhD97AFxQTfHdyONeKJchxySqfxZQ66JqLBSI.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:56: cs232main.c.cs.uchicago.edu
    ~/.ssh/known_hosts:58: a2bailey.c.cs.uchicago.edu
    ~/.ssh/known_hosts:59: 128.135.37.128
    ~/.ssh/known_hosts:60: cs232-02.c.cs.uchicago.edu
    ~/.ssh/known_hosts:61: cs232-10.c.cs.uchicago.edu
    ~/.ssh/known_hosts:62: cs232-53.c.cs.uchicago.edu
    ~/.ssh/known_hosts:63: cs232-52.c.cs.uchicago.edu
    ~/.ssh/known_hosts:64: cs232-01.c.cs.uchicago.edu
    (19 additional names omitted)
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

A Glimpse at Public-Key Encryption: RSA

RSA Key Generation

- Pick p and q be *large* random prime numbers (around 2^{1024})
- Compute $N \leftarrow pq$
- Set e to a default value ($e = 3$ and $e = 65537$ are common)
- Compute d such that $ed = 1 \bmod (p-1)(q-1)$
- Output
  - Public key $pk = (N, e)$
  - Secret key $sk = (N, d)$

Example:

- $p = 5,\ q = 11,\ N = 55$
- $e = 3,\ d = 27$

Plain RSA Encryption

$PK = (N, e)$ $SK = (N, d)$ where $N = pq, \ \ ed = 1 \bmod \phi(N)$

$$\mathrm{Enc}((N, e), x) = x^e \bmod N$$

$$\mathrm{Dec}((N, d), y) = y^d \bmod N$$

Using number theory from CMSC 27100, can show:

$$\mathrm{Dec}(\mathrm{Enc}((N, e), x)) = (x^e)^d = x \bmod N$$

Never use directly as encryption! **Warning: Broken**

Factoring Records and RSA Key Length

- Factoring N allows recovery of secret key
- Challenges posted publicly by RSA Laboratories

Bit-length of N	Year
400	1993
478	1994
515	1999
768	2009
795	2019

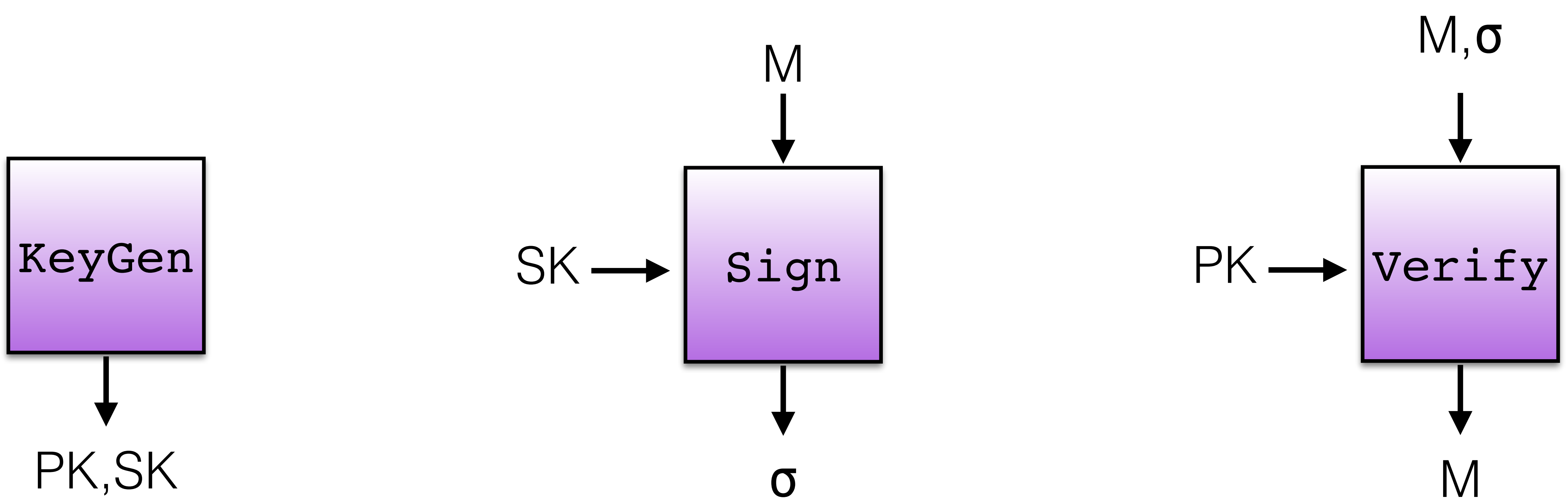
- Recommended bit-length today: 2048 or greater
- Note that fast algorithms force such a large key.
  - 512-bit N defeats naive factoring

Outline

- Message Authentication
- Hash Functions
- Public-Key Encryption
- **Digital Signatures**

Digital Signatures Schemes

A <u>digital signature scheme</u> consists of three algorithms **`KeyGen`**, **`Sign`**, and **`Verify`**

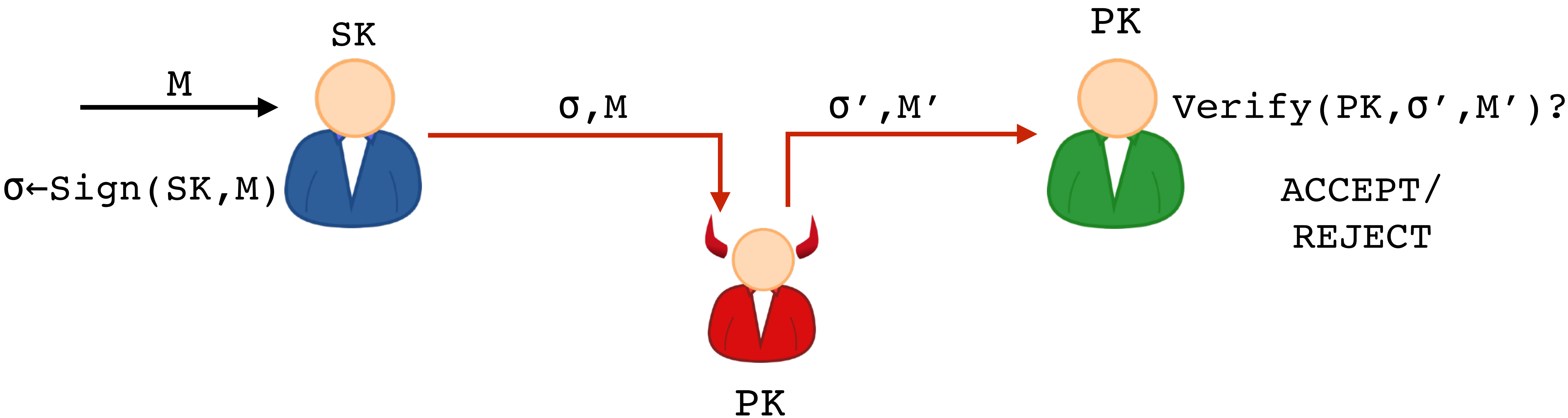


<u>`KeyGen`</u>: Outputs two keys. PK published openly, and SK kept secret.

<u>`Sign`</u>: Uses SK to produce a "signature" **σ** on M.

<u>`Verify`</u>: Uses PK to check if signature **σ** is valid for M.

Digital Signature Security Goal: Unforgeability



Scheme satisfies **unforgeability** if it is unfeasible for Adversary (who knows `PK`) to fool Bob into accepting `M′` not previously sent by Alice.

"Plain" RSA with No Encoding

Broken

$PK = (N, e)$ $SK = (N, d)$ where $N = pq, \;\; ed = 1 \bmod \phi(N)$

$$\text{Sign}((N,d), M) = M^d \bmod N$$

$$\text{Verify}((N,e), M, \sigma) : \sigma^e = M \bmod N?$$

$e = 3$ is common for fast verification.

RSA Signatures with Encoding

$PK = (N, e)$ $SK = (N, d)$ where $N = pq,\ ed = 1 \bmod \phi(N)$

$$\text{Sign}((N, d), M) = \text{encode}(M)^d \bmod N$$

$$\text{Verify}((N, e), M, \sigma) : \sigma^e = \text{encode}(M) \bmod N?$$

encode maps bit strings to numbers between 0 and N

Encoding must be chosen with extreme care.

Broken

Forging RSA Signatures with Encoding

To forge a signature on M, and adversary must find a integer σ between 0 and N such that:

$$\sigma^e = \text{encode}(M) \bmod N$$

When $e = 3$, this is just

$$\sigma^3 = \text{encode}(M) \bmod N$$

<u>Easy</u>: Find a *real number* σ such that

$$\sigma^3 = \text{encode}(M) \bmod N$$

In fact, we can find σ such that

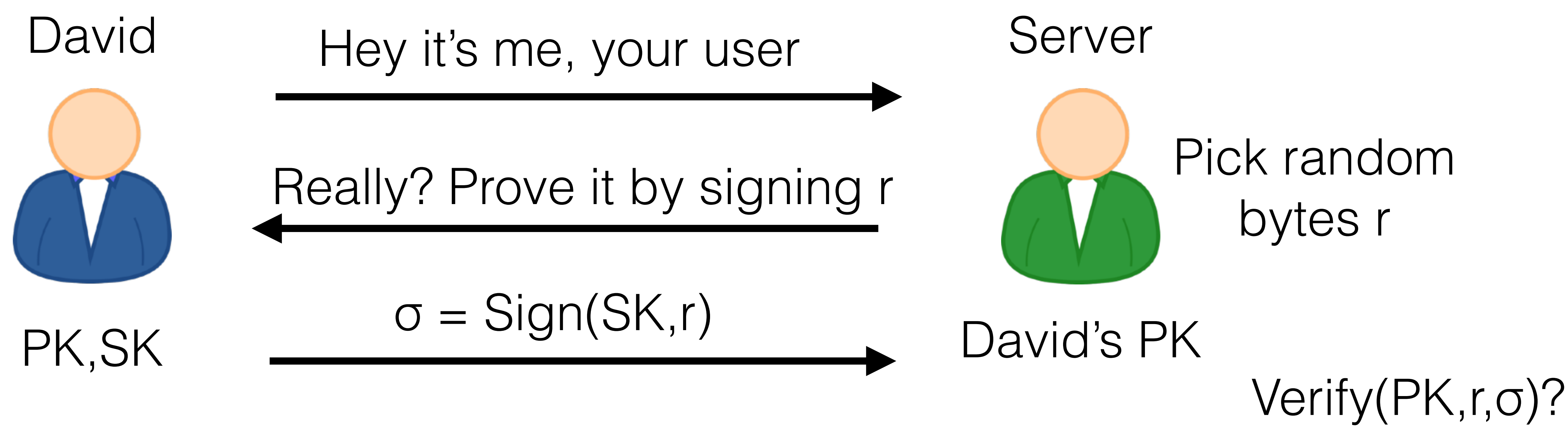
$$\sigma^3 = \text{encode}(M).$$

It's just $\sigma = \sqrt[3]{\text{encode}(M)}$, which is easy to compute even if the numbers involved are large.

<u>Hard</u>: Find an *integer* σ such that

$$\sigma^3 = \text{encode}(M) \bmod N$$

Signatures for Authentication



- This and similar ideas used in SSH, TLS, etc
- Contrast with passwords?

Example RSA Signature Encoding: Full Domain Hash

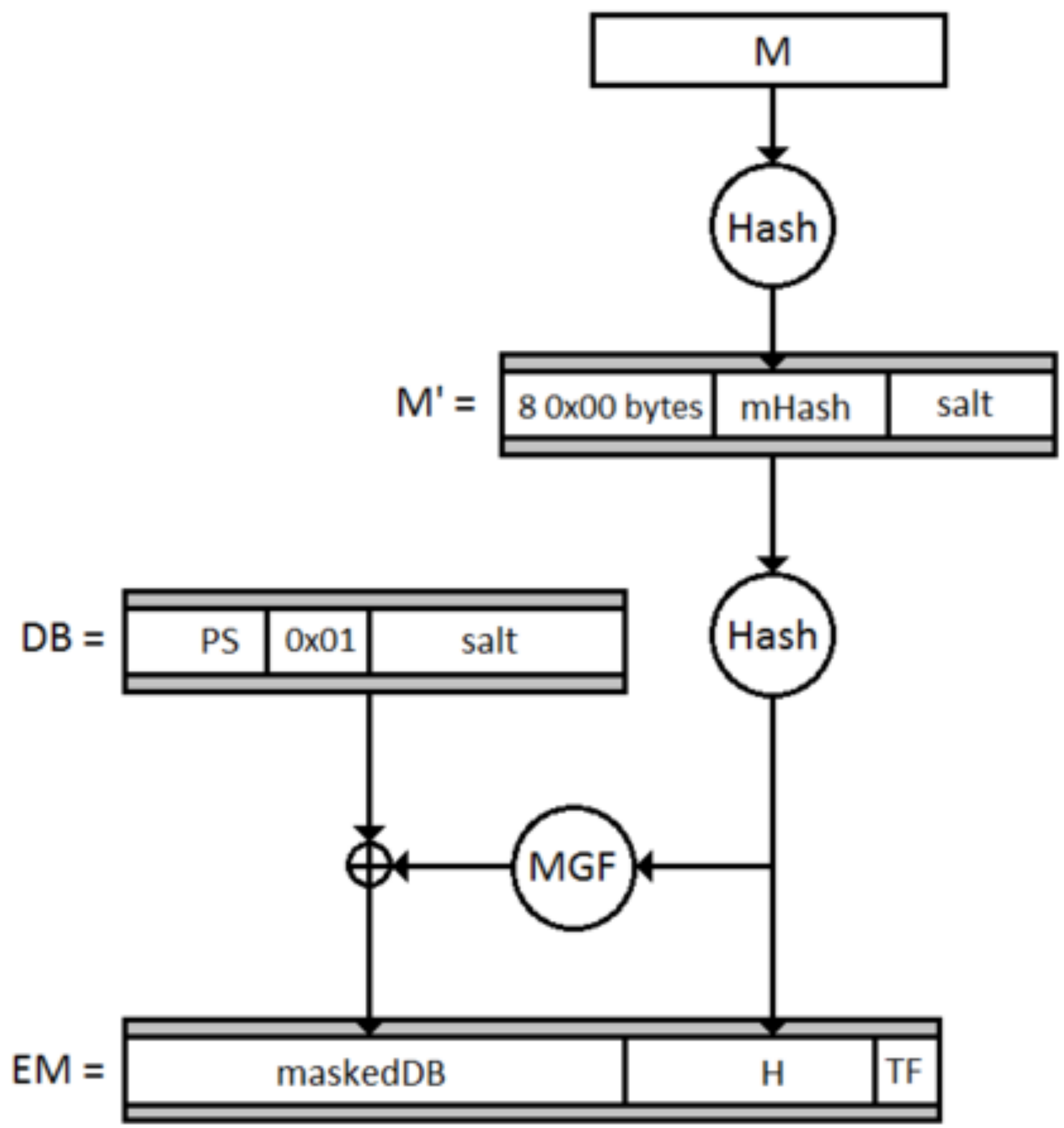
N: **n**-byte long integer.
H: Hash fcn with **m**-byte output. ← Ex: **SHA-256, m=32**
k = ceil((n-1)/m)

```
Sign((N,d),M):
1. X←00||H(1||M)||H(2||M)||…||H(k||M)
2. Output σ = X^d mod N
```

```
Verify((N,e),M,σ):
1. X←00||H(1||M)||H(2||M)||…||H(k||M)
2. Check if σ^e = X mod N
```

Other RSA Padding Schemes: PSS (In TLS 1.3)

- Somewhat complicated
- *Randomized* signing



RSA Signature Summary

- Plain RSA signatures are very broken
- PKCS#1 v.1.5 is widely used, in TLS, and fine if implemented correctly
- Full-Domain Hash and PSS should be preferred
- Don’t roll your own RSA signatures!

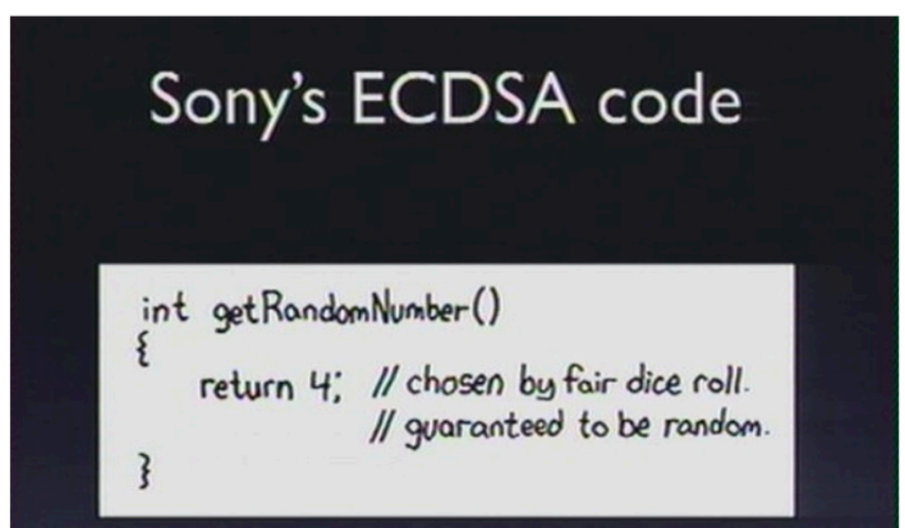
Other Practical Signatures: DSA/ECDSA

- Based on ideas related to Diffie-Hellman key exchange
- EC version has shorter keys
- Secure, but even more ripe for implementation errors

Hackers obtain PS3 private cryptography key due to epic programming fail? (update)

Sean Hollister
12.29.10

2
Shares



The End