Cryptography Part 2

CMSC 23200/33250, Winter 2022, Lecture 8

David Cash & Blase Ur

University of Chicago

Outline

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

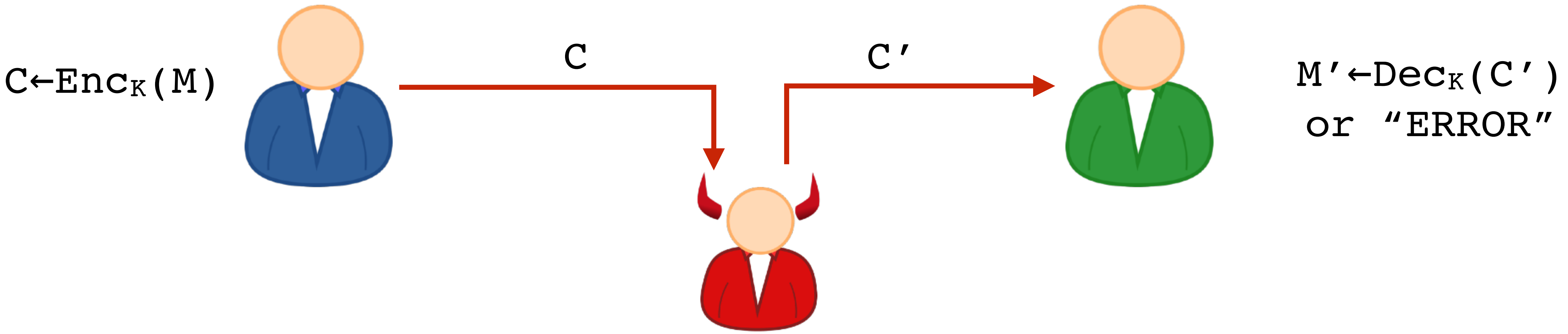
Outline

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

Next Up: Integrity and Authentication

- Authenticity: Guarantee that adversary cannot change or insert ciphertexts
- Achieved with MAC = “Message Authentication Code”

Encryption Integrity: An abstract setting



Encryption satisfies **integrity** if it is infeasible for an adversary to send a new C′ such that $\text{Dec}_K(C') \neq \text{ERROR}$.

Stream ciphers do not give integrity

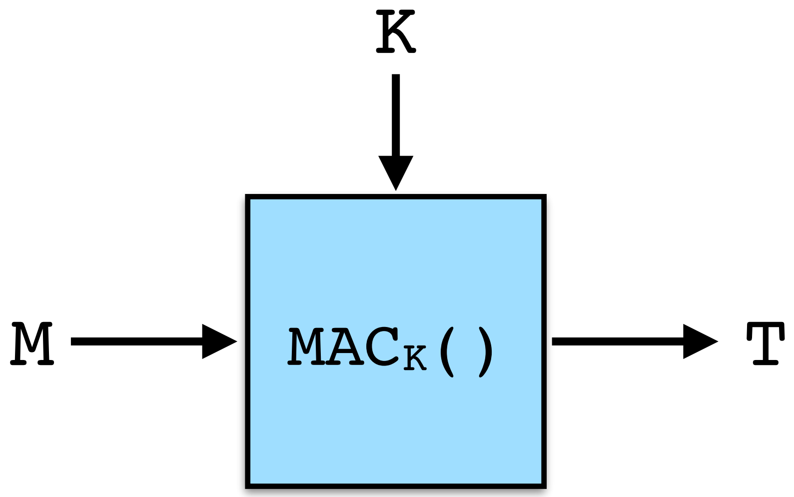
```
M = please pay ben 20 bucks

C = b0595fafd05df4a7d8a04ced2d1ec800d2daed851ff509b3e446a782871c2d
```

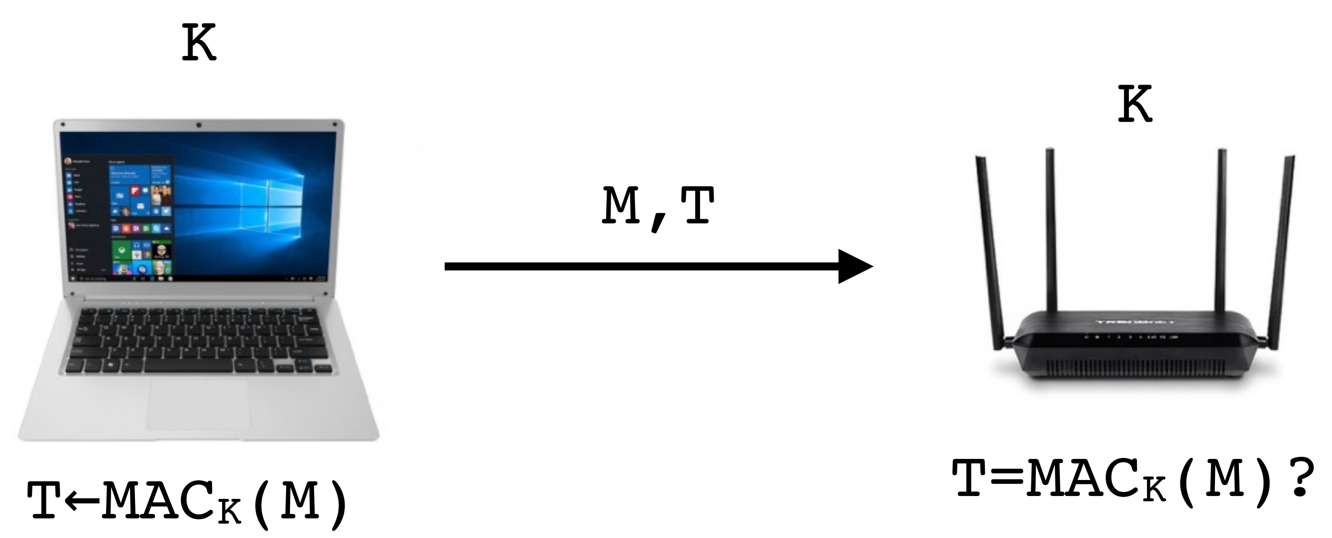
```
C’= b0595fafd05df4a7d8a04ced2d1ec800d2daed851ff509b3e546a782871c2d

M’ = please pay ben 21 bucks
```

Inherent to stream-cipher approach to encryption.

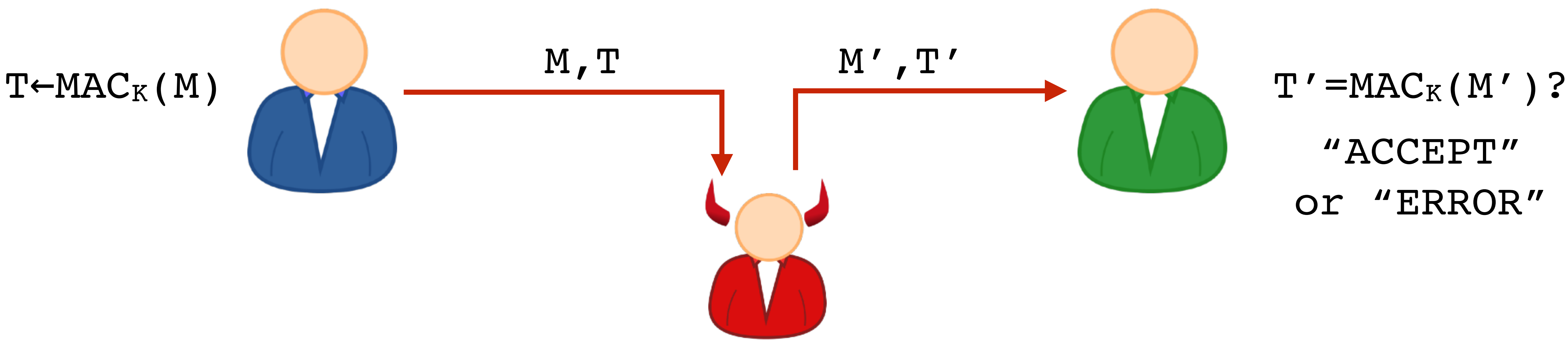
Message Authentication Code

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





MAC Security Goal: Unforgeability



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

MAC Security Goal: Unforgeability

Note: No encryption on this slide.

`M = please pay ben 20 bucks`

`T = 827851dc9cf0f92ddcdc552572ffd8bc`

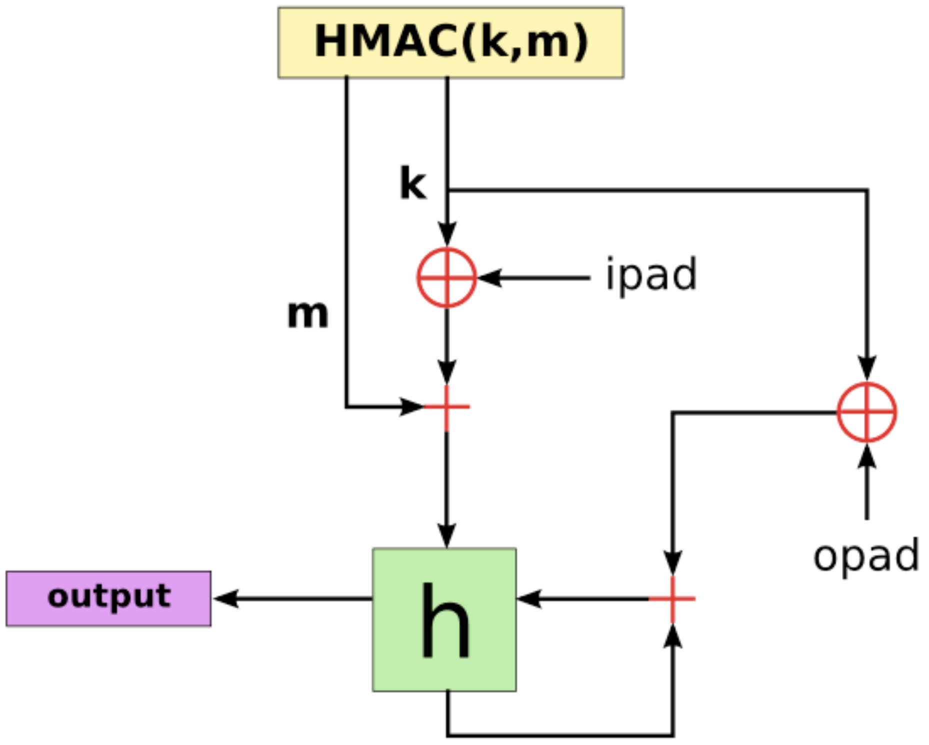
`M’= please pay ben 21 bucks`

`T’= baeaf48a891de588ce588f8535ef58b6`

Should be hard to predict **`T’`** for any new **`M’`**.

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**.

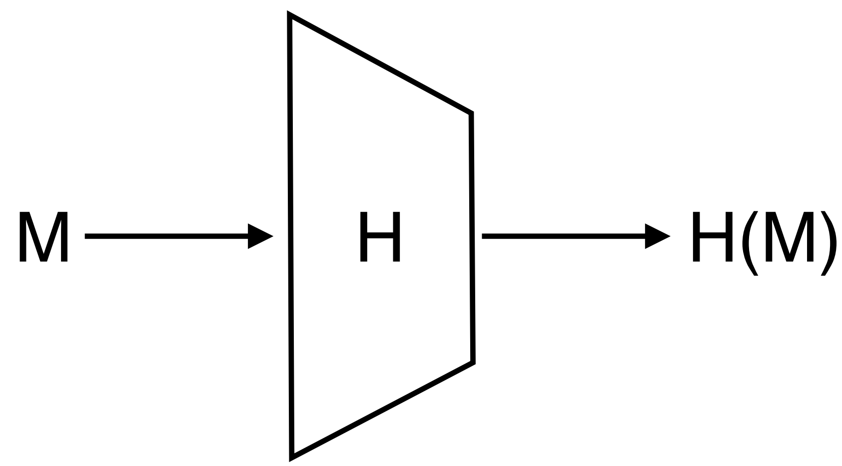
- Built using a good stream cipher and a MAC.
  - Ex: Salsa20 with HMAC-SHA2
- Best solution: Use ready-made Authenticated Encryption
  - Ex: AES-GCM is the standard

Outline

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

Next Up: Hash Functions

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

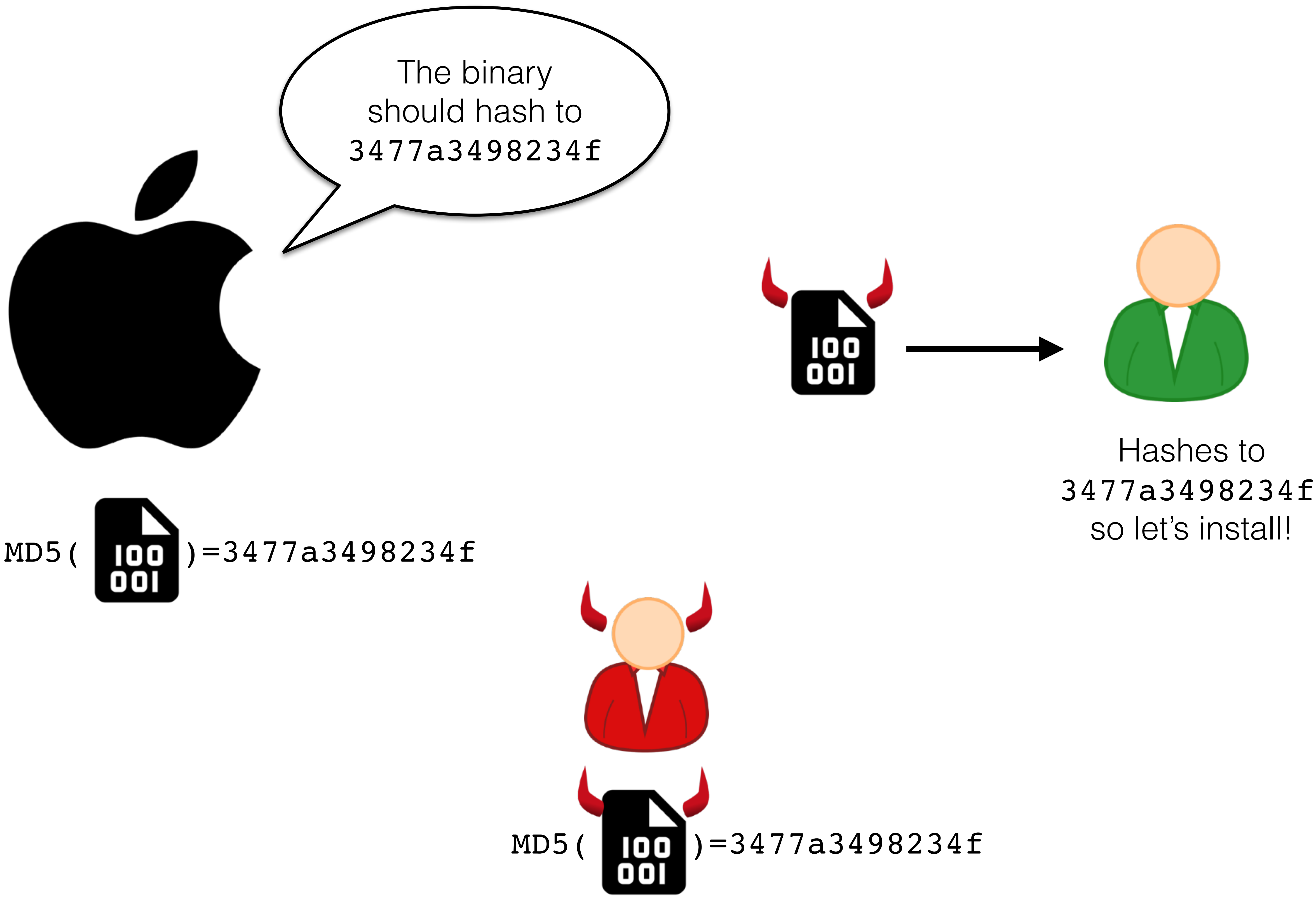


Some security goals:

- 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?

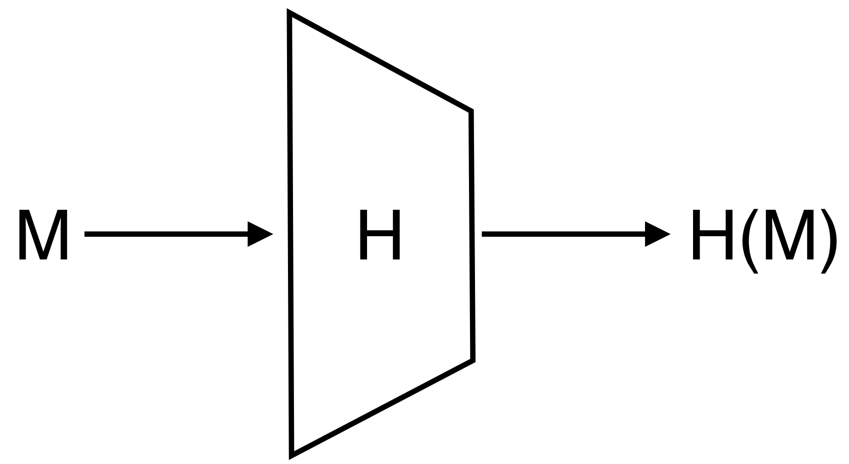


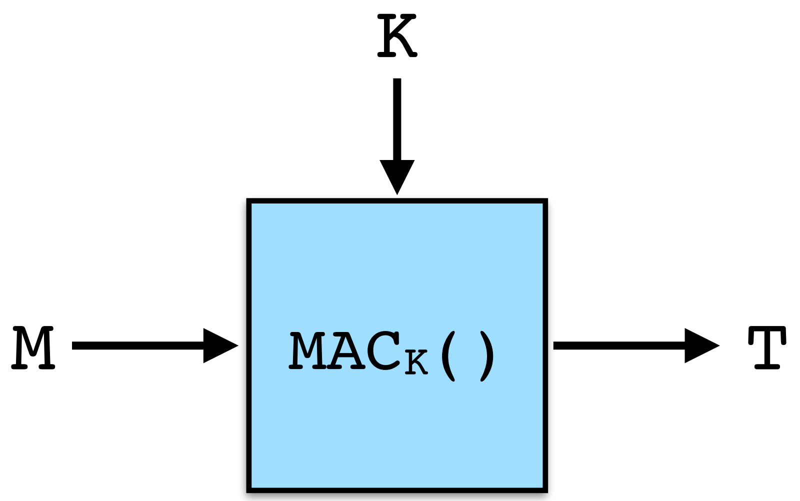
Practical Hash Functions

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: $MAC(K, M) = H(K \| M)$ **Warning: Broken**

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

Construction: $MAC(K, M) = H(M \| K)$ **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, M) = H(K || M) **Warning: Broken**

Adversary goal: Find new message M' and a valid tag T' for M'

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

In Assignment 3: Break this construction!

Outline

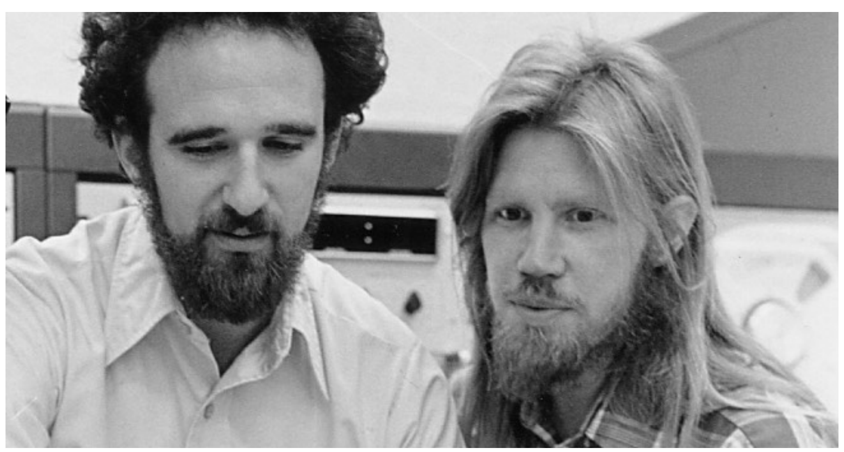
- Message Authentication
- Hash Functions
- **Public-Key Encryption**
- Digital Signatures

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?

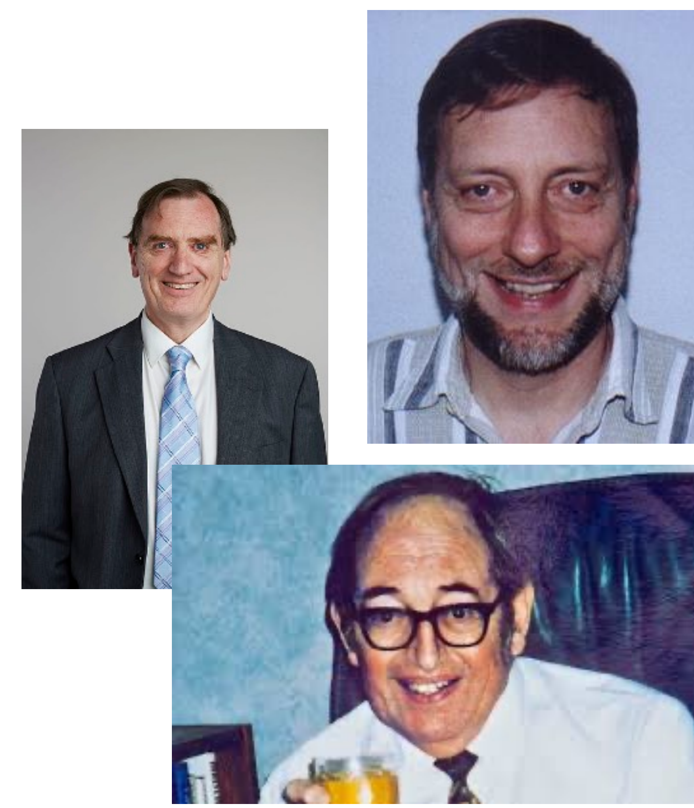


Diffie and Hellman
in 1976: **Yes!**

Turing Award, 2015,
+ Million Dollars

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

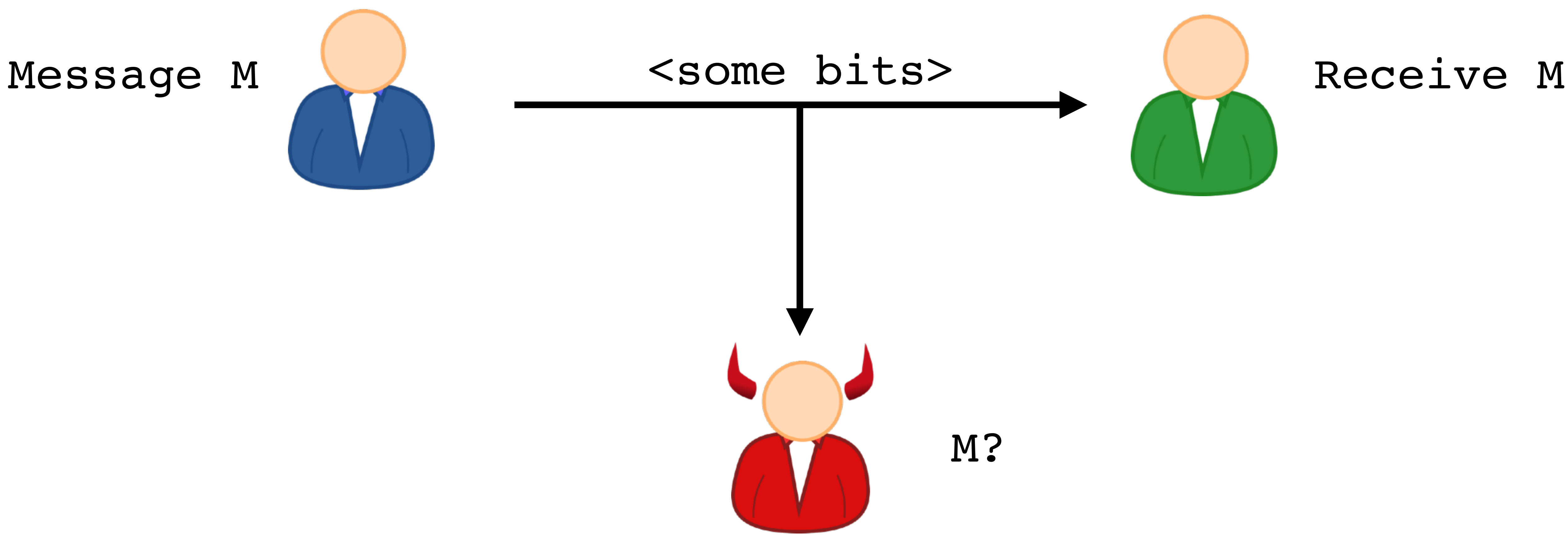
Turing Award, 2002,
+ no money



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

The Seed of Public-Key Cryptography

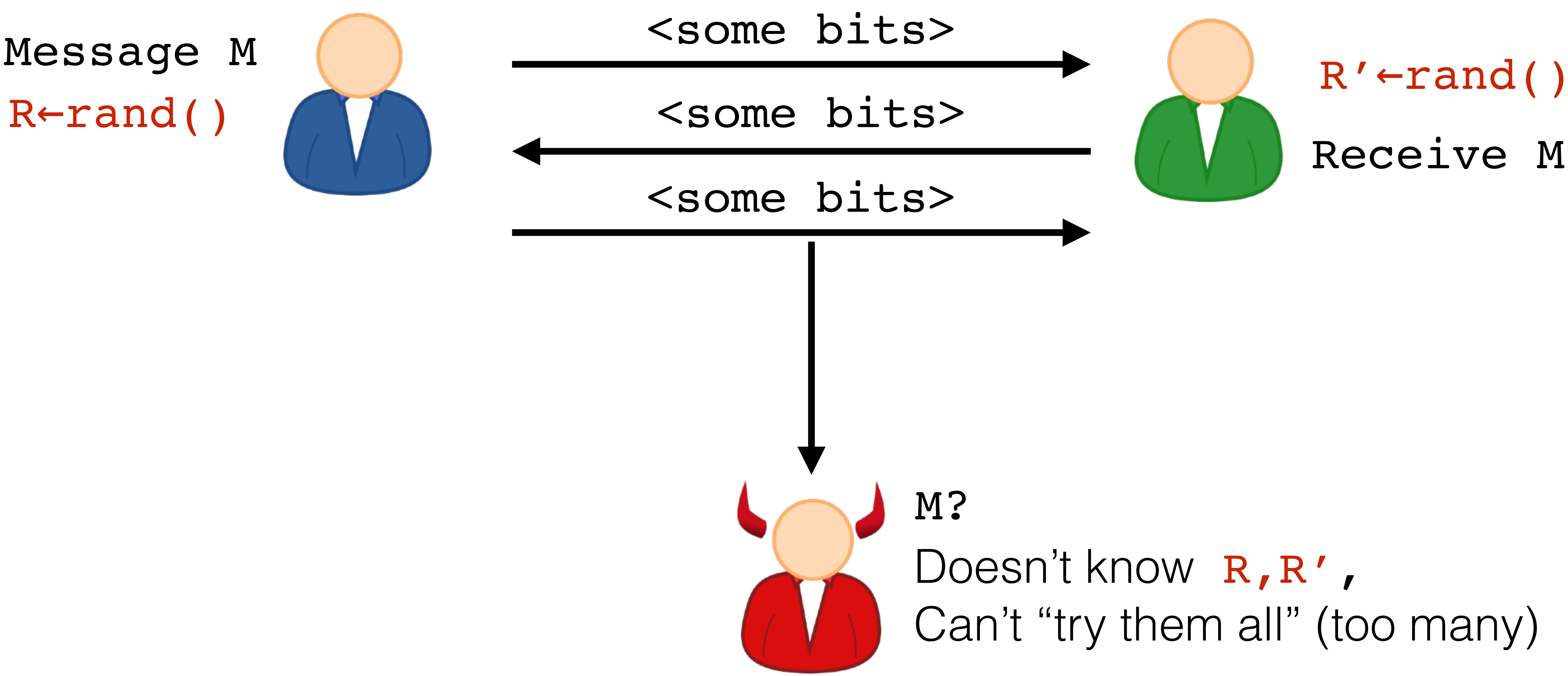
Basic question: If two people are talking in the presence of an eavesdropper, and they don't have pre-shared a key, is there any way they can send private messages?



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

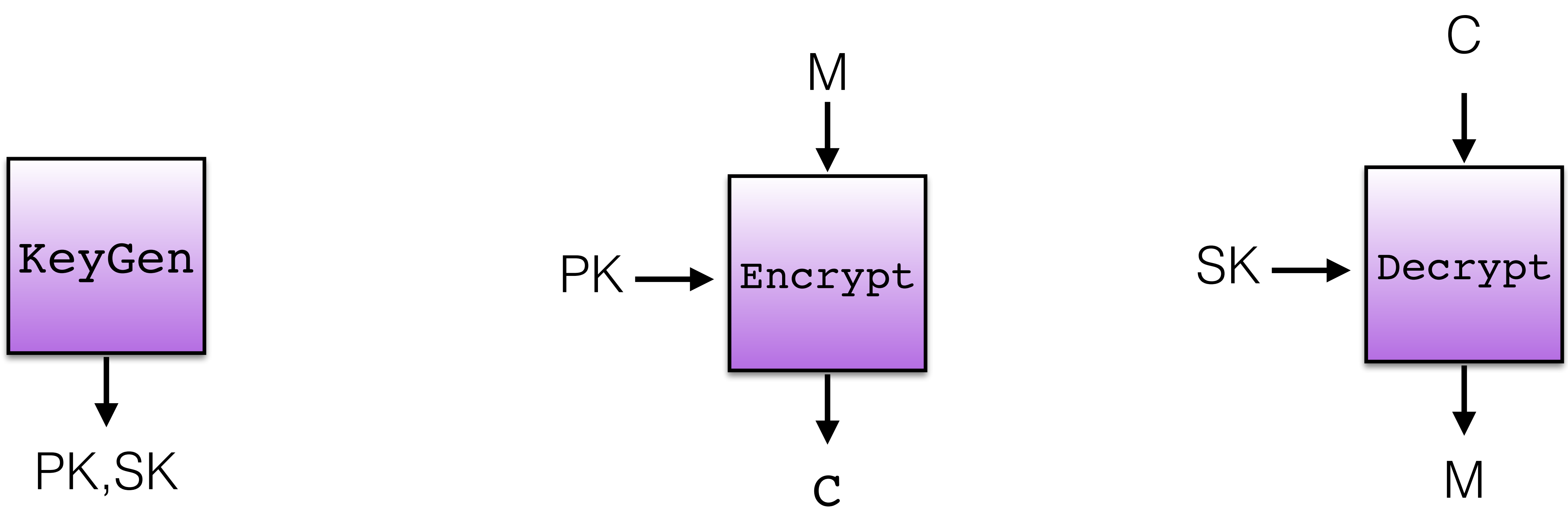
The Seed of Public-Key Cryptography

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**

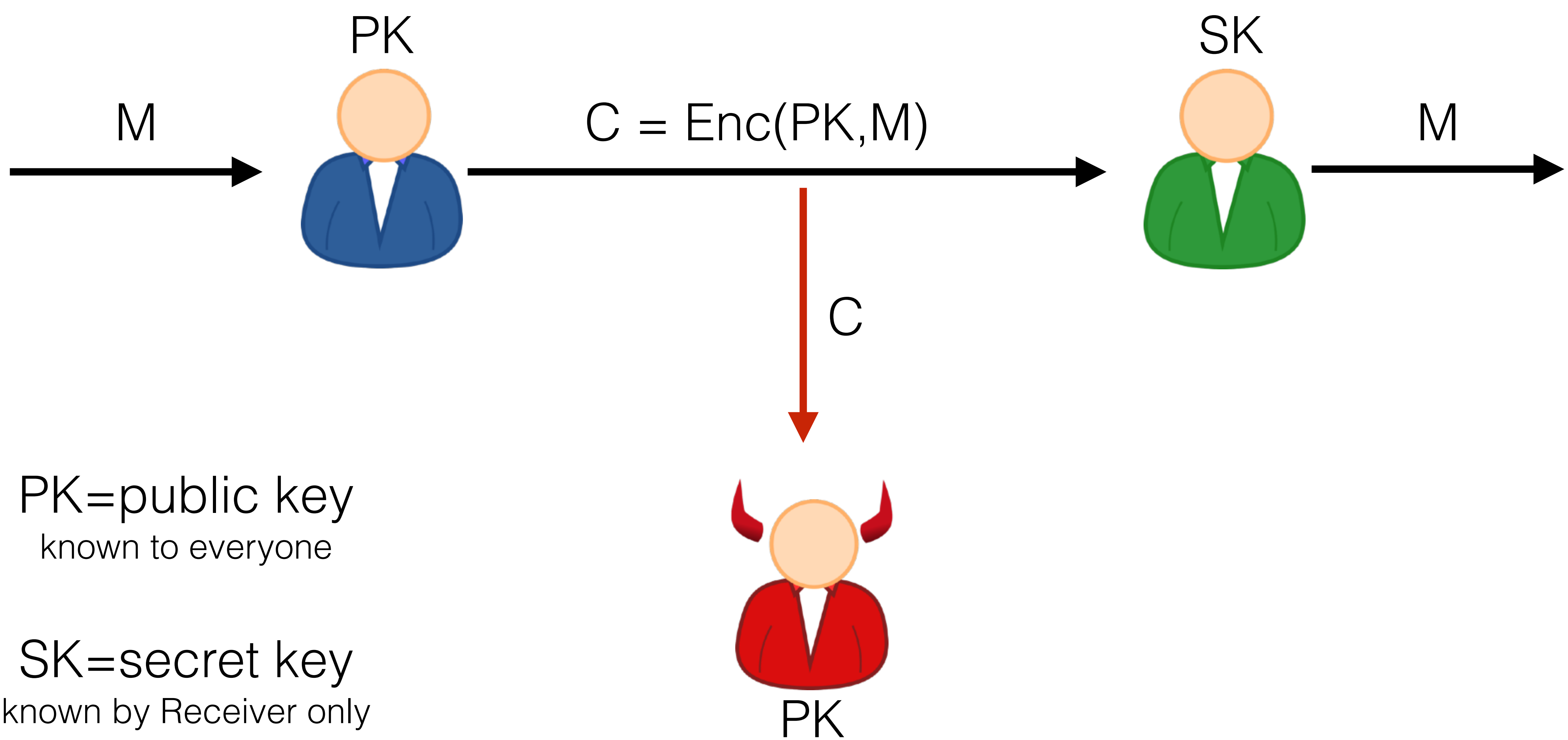


KeyGen: Outputs two keys. PK published openly, and SK kept secret.

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

Some RSA Math

Called "2048-bit primes"

RSA setup

p and q be large prime numbers (e.g. around 2^{2048})

$N = pq$

N is called the **modulus**

p=7, q=11 gives N=77

p=17 q=61 gives N=1037

RSA "Trapdoor Function"

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

$$\text{RSA}((N, e), x) = x^e \bmod N$$

$$\text{RSA}^{-1}((N, d), y) = y^d \bmod N$$

Setting up RSA:
- Need two large random primes
- Have to pick e and then find d
- Not covered in 232/332: How this really works.

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

Encrypting with the RSA Trapdoor Function

"Hybrid Encryption":
- Apply RSA to random x
- Hash x to get a symmetric key k
- Encrypted message under k

```
Enc((N,e),M):

1. Pick random x // 0 <= x < N
2. c0←(x^e mod N)
3. k←H(x)
4. c1←SymEnc(k,M) // symmetric enc.
5. Output (c0,c1)
```

```
Dec((N,d), (c0,c1)):

1. x←(c0^d mod N)
2. k←H(x)
3. M←SymDec(k,c1)
4. Output M
```

Do not implement yourself!

Warning: Broken

- Use RSA-OAEP, which uses hash in more complicated way.

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

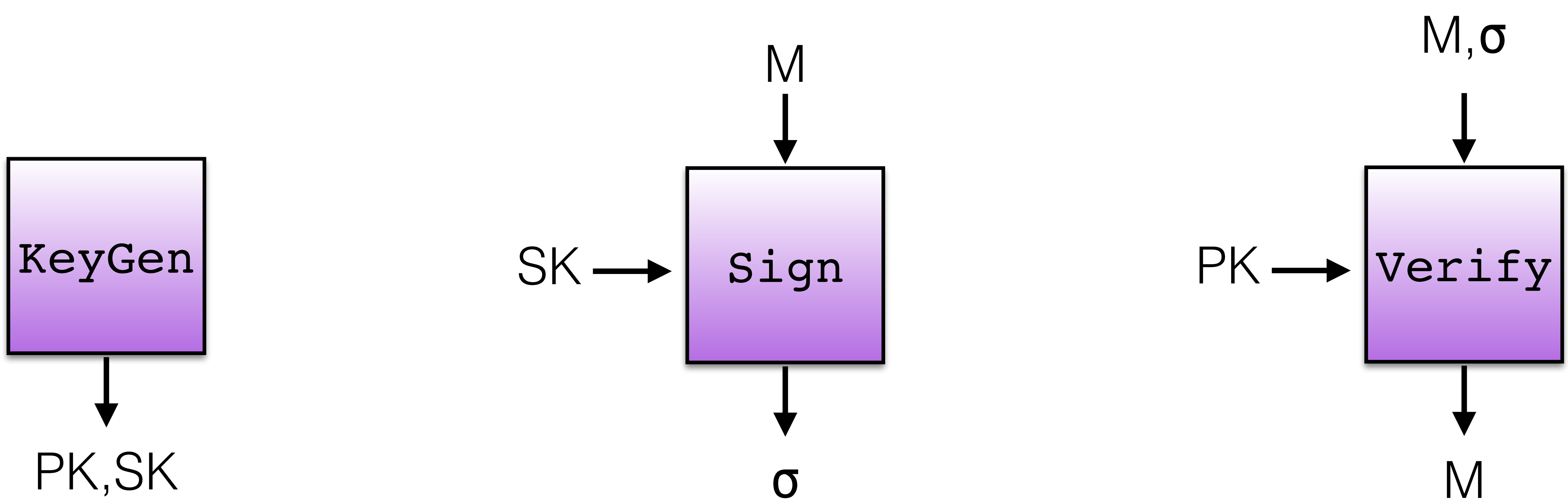
- Recommended bit-length today: 2048
- Note that fast algorithms force such a large key.
  - 512-bit N defeats naive factoring

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`**

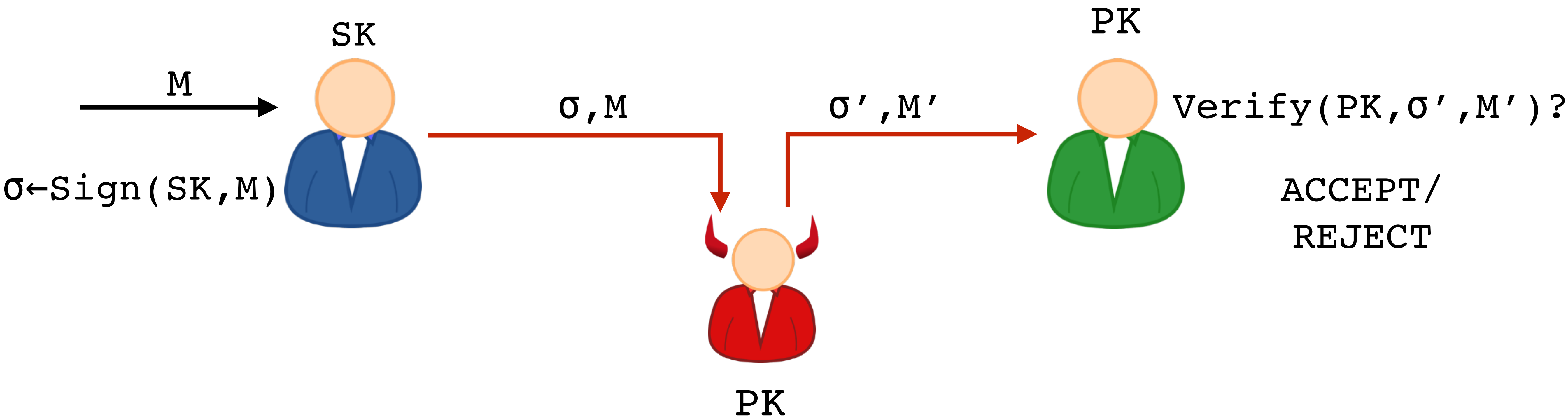


<u>`KeyGen`</u>: Outputs two keys. PK published openly, and SK kept secret.

<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 $N = pq$, $ed = 1 \bmod \phi(N)$

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

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

$e = 3$ 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

Example RSA Signature: Full Domain Hash

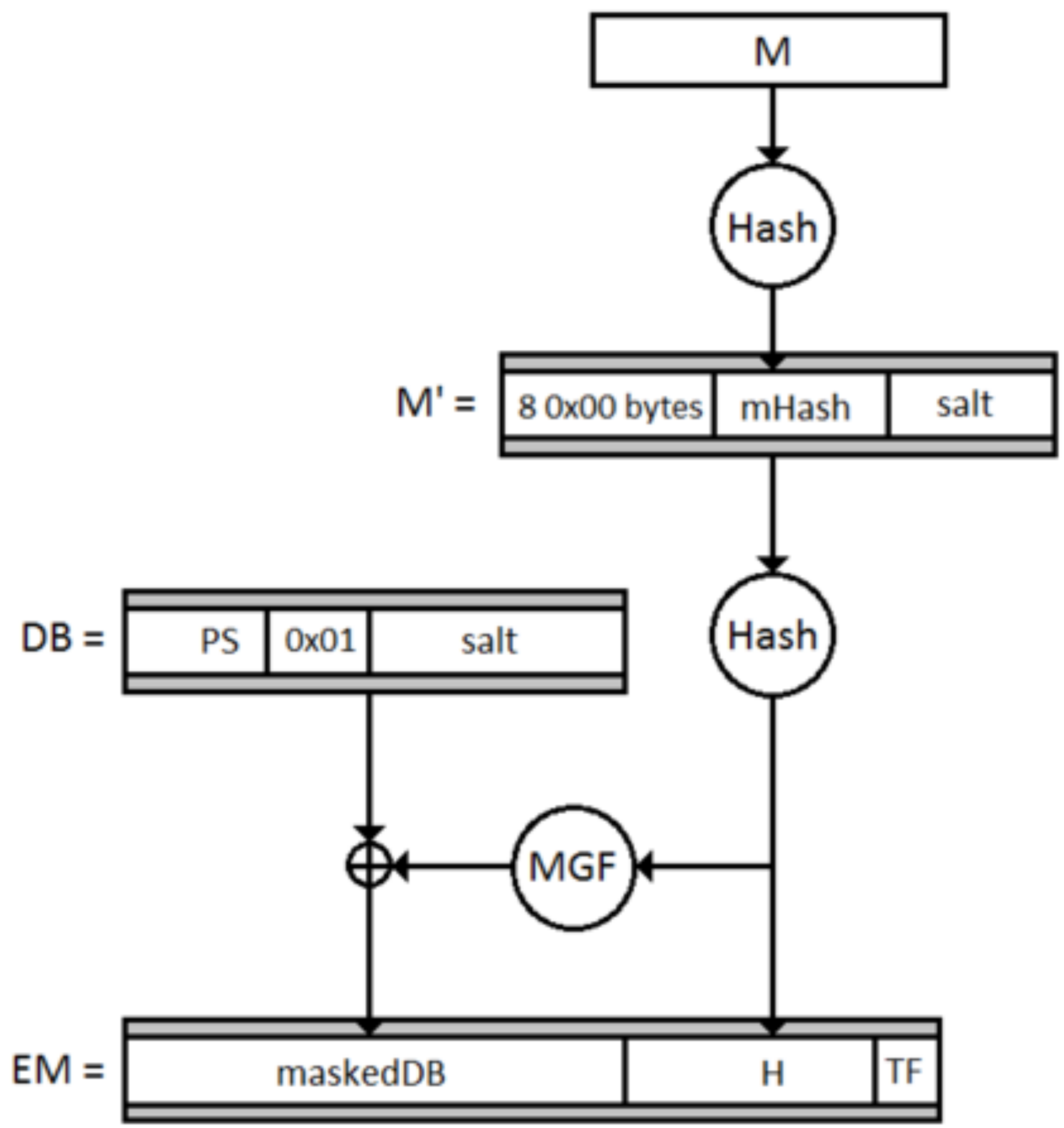
N: **n**-byte long integer.
H: Hash fcn with **m**-byte output. ← Ex: **SHA-256, m=32**
k = ceil((n-1)/m)

```
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!

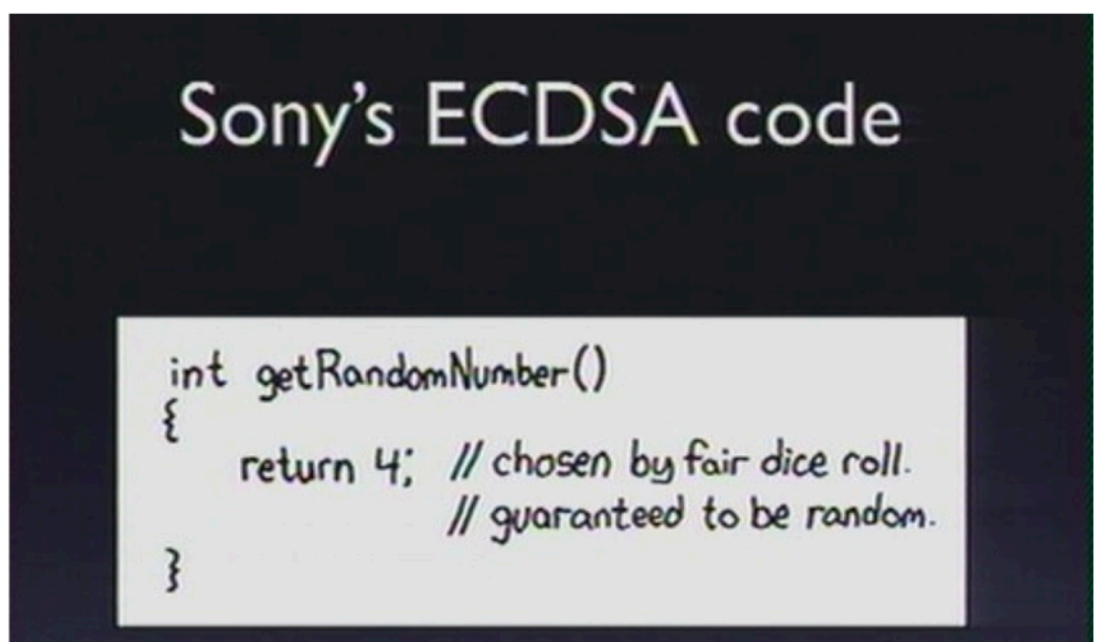
Other Practical Signatures: DSA/ECDSA

- Based on ideas related to Diffie-Hellman key exchange
- 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