Crypto Concepts

Symmetric encryption, Public key encryption, and TLS

Cryptography

Is:

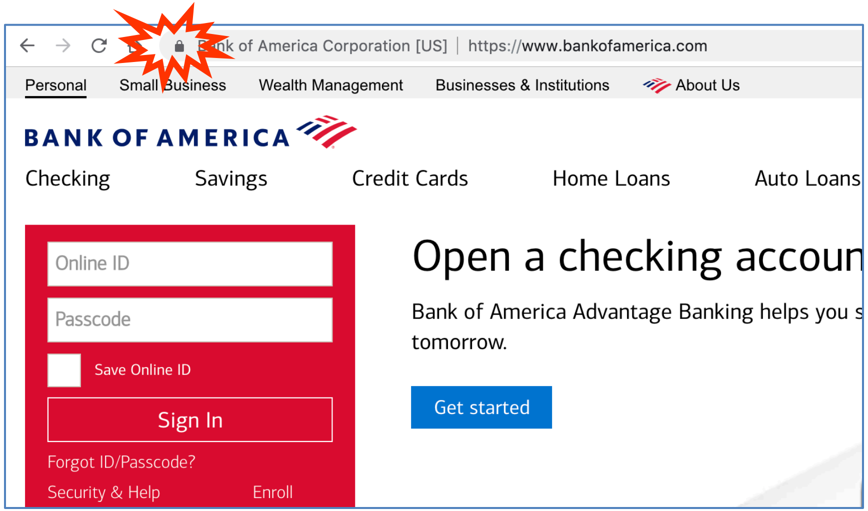
- A tremendous tool
- The basis for many security mechanisms

Is not:

- The solution to all security problems
- Reliable unless implemented and used properly
- Something you should try to invent yourself

Goal 1: Secure communication

(protecting data in motion)



no eavesdropping
no tampering

Transport Layer Security / TLS

Standard for Internet security

– Goal: "... provide privacy and reliability between two communicating applications"

Two main parts

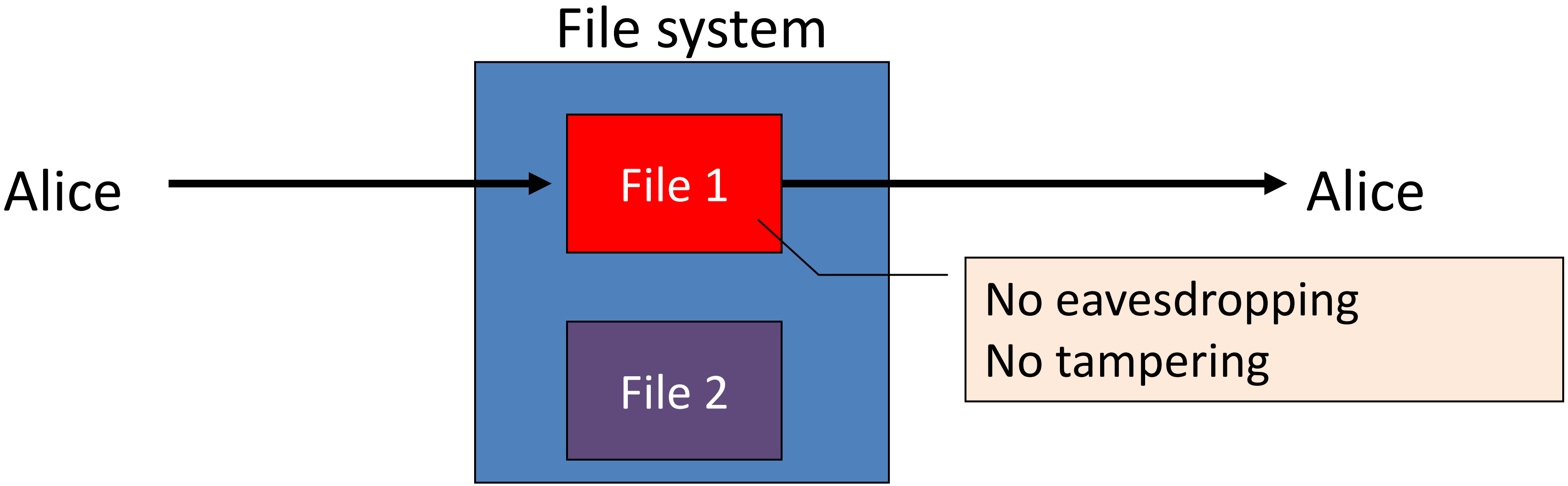
1. Handshake Protocol: **Establish shared secret key using public-key cryptography**

2. Record Layer: **Transmit data using negotiated key**

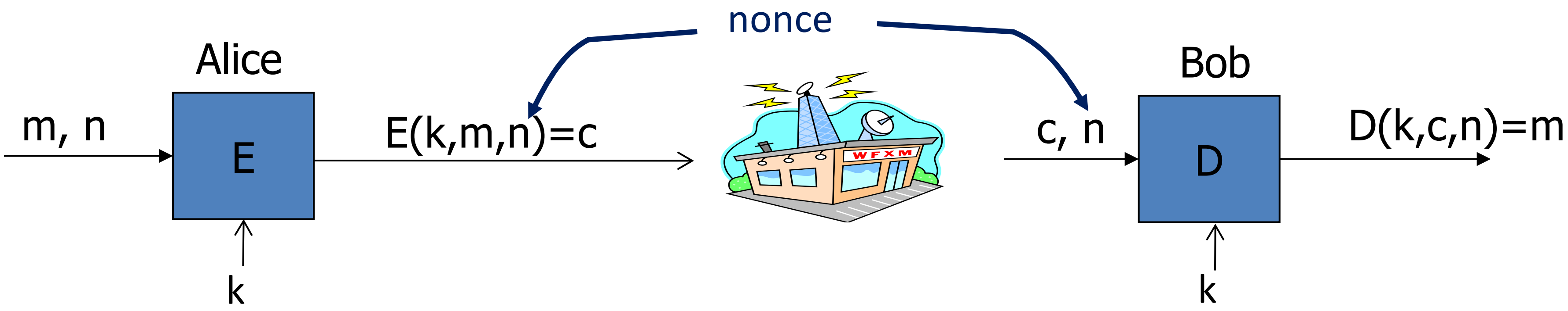
Our starting point: Using a key for encryption and integrity

Goal 2: protected files

(protecting data at rest)



Building block: symmetric cipher



E, D: cipher   k: secret key (e.g. 128 bits)

m, c: plaintext, ciphertext   n: nonce (non-repeating)

Encryption algorithm is publicly known

$\Rightarrow$ never use a proprietary cipher

Use Cases

Single use key: (one time key)

- Key is only used to encrypt one message
  - encrypted email: new key generated for every email
- No need for nonce (set to 0)

Multi use key: (many time key)

- Key used to encrypt multiple messages
  - TLS: same key used to encrypt many packets
- Use either a *unique* nonce or a *random* nonce

First example: One Time Pad (single use key)

Vernam (1917)

Key:	0	1	0	1	1	1	0	0	1	0
Plaintext:	1	1	0	0	0	1	1	0	0	0
Ciphertext:	1	0	0	1	1	0	1	0	1	0

$\oplus$

Encryption: $c = E(k, m) = m \oplus k$

Decryption: $D(k, c) = c \oplus k \quad = (m \oplus k) \oplus k = m$

One Time Pad (OTP) Security

Shannon (1949):

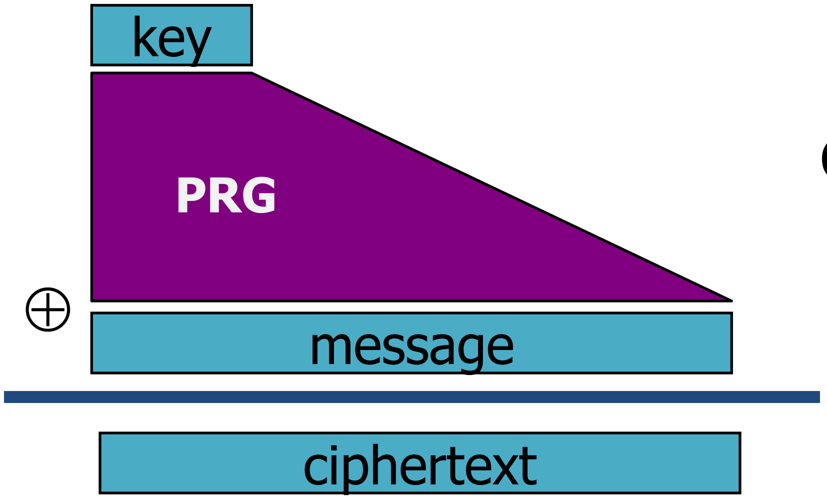
- OTP is “secure” against one-time eavesdropping
- without key, ciphertext reveals no “information” about plaintext

Problem: OTP key is as long as the message

Stream ciphers (single use key)

Problem: OTP key is as long as the message

Solution: Pseudo random key -- stream ciphers



$c \leftarrow \mathbf{PRG}(k) \oplus m$

Example: **ChaCha20** (one-time if no nonce) key: 128 or 256 bits.

Dangers in using stream ciphers

One time key !!   "Two time pad" is insecure:

$$c_1 \leftarrow m_1 \oplus PRG(k)$$

$$c_2 \leftarrow m_2 \oplus PRG(k)$$

What if want to use same key to encrypt two files?

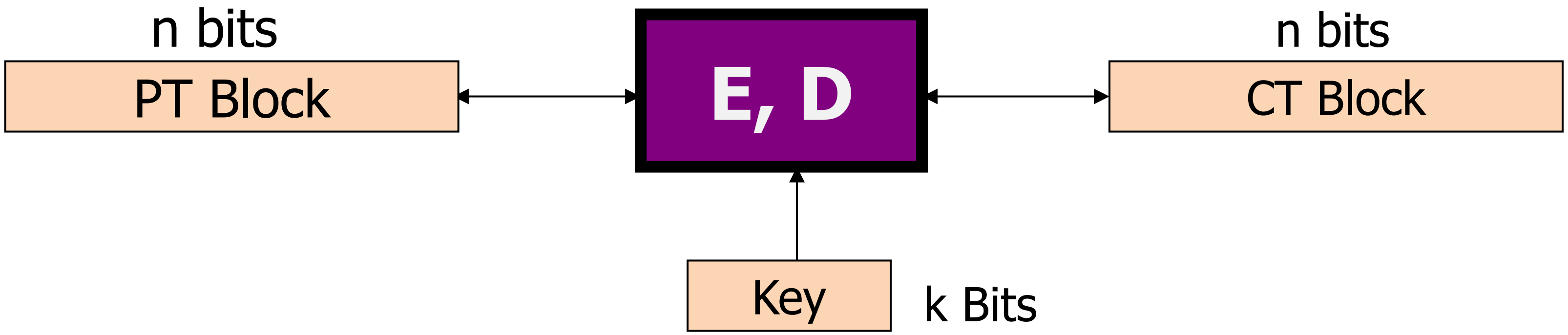
Eavesdropper does:

$$c_1 \oplus c_2 \rightarrow m_1 \oplus m_2$$

Enough redundant information in English that:

$$m_1 \oplus m_2 \rightarrow m_1 , m_2$$

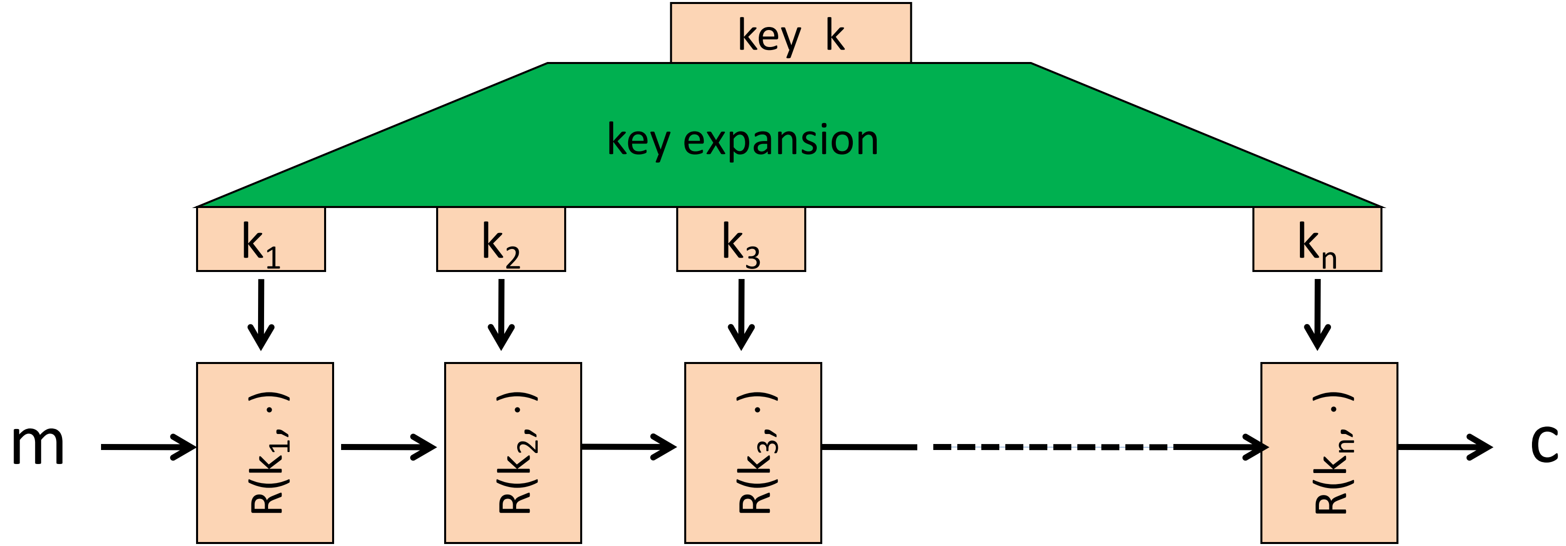
Block ciphers: crypto work horse



Canonical examples:

1. 3DES: n= 64 bits, k = 168 bits
2. AES: n=128 bits, k = 128, 192, 256 bits

Block Ciphers Built by Iteration

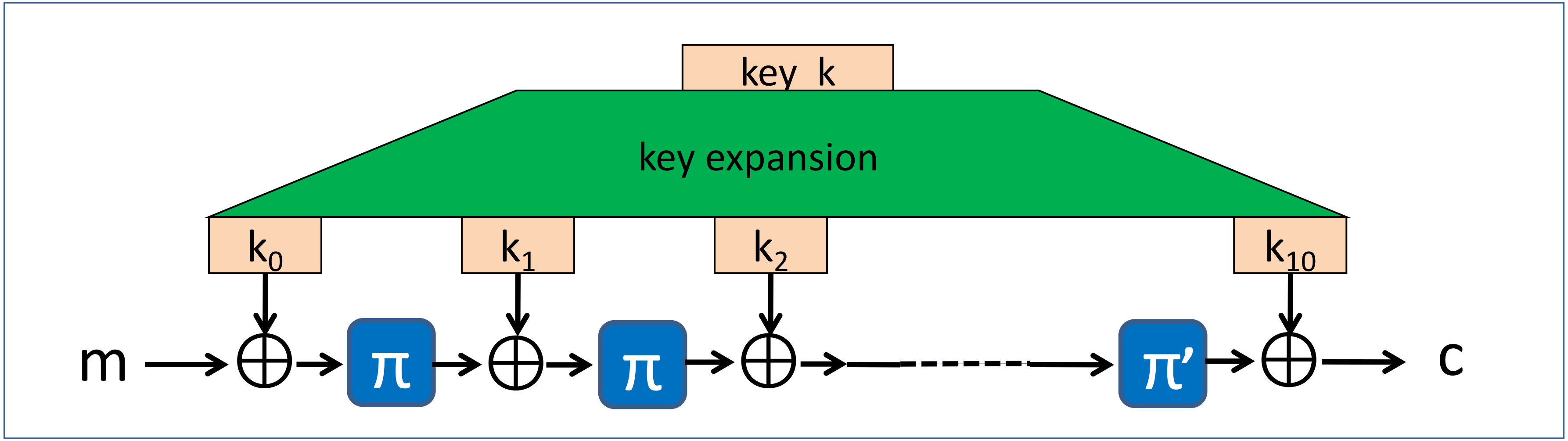


R(k,m): round function

for 3DES (n=48), for AES-128 (n=10)

Example: AES128

input: 128-bit block m, 128-bit key k. **output**: 128-bit block c.

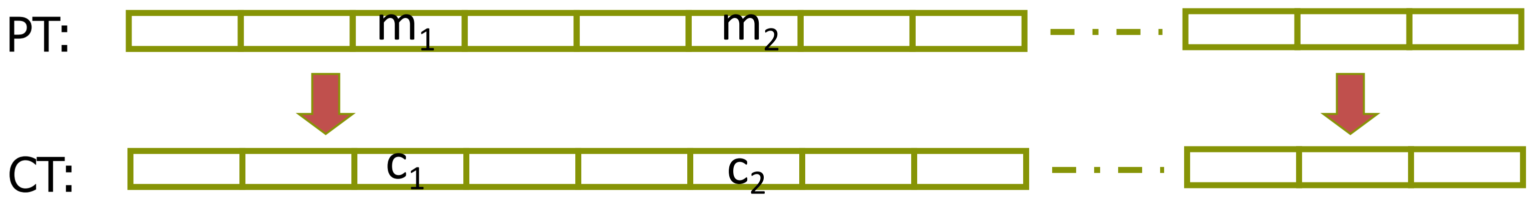


Difficult to design: must resist subtle attacks

- differential attacks, linear attacks, brute-force, …

Incorrect use of block ciphers

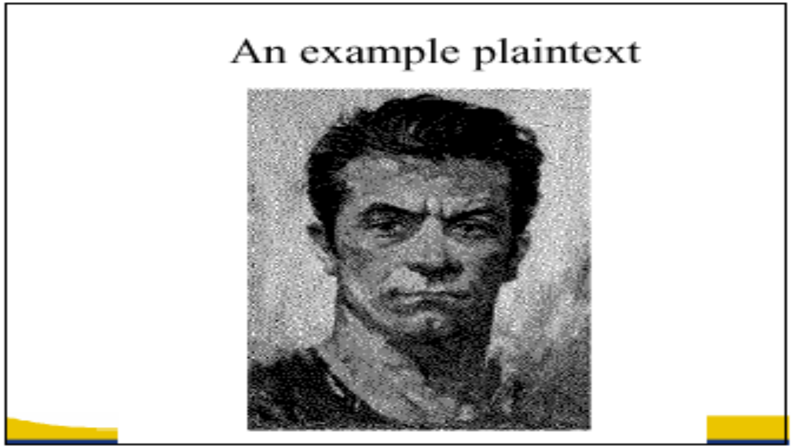
Electronic Code Book (ECB):

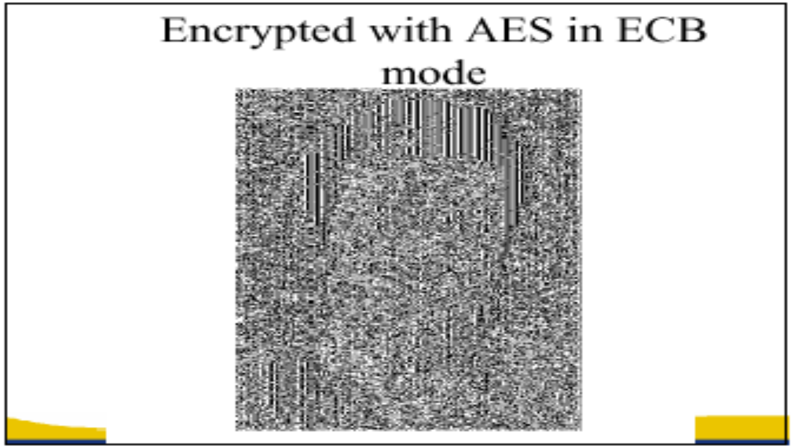


Problem:

– if $m_1=m_2$ then $c_1=c_2$

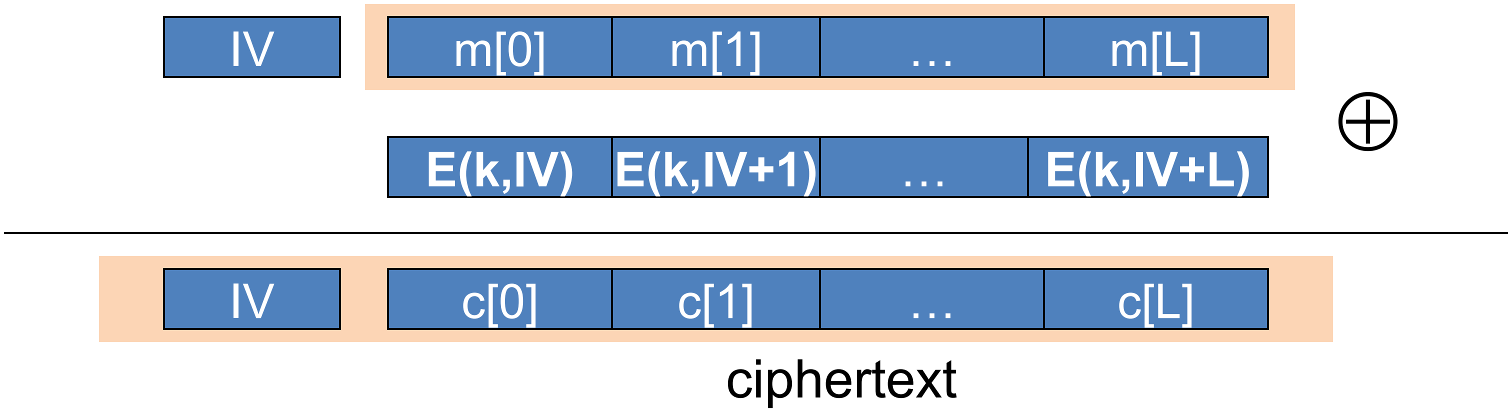
In pictures





CTR mode encryption (eavesdropping security)

Counter mode with a random IV: (parallel encryption)



Why is this secure for multiple messages? See the crypto course (cs255)

Performance

OpenSSL on Intel Haswell, 2.3 GHz (Linux)

	Cipher	Block/key size	Speed (MB/sec)
stream	ChaCha		408
block	3DES	64/168	30
	AES128	128/128	176
	AES256	128/256	135

(w/o AES-NI)

A Warning

eavesdropping security is insufficient for most applications

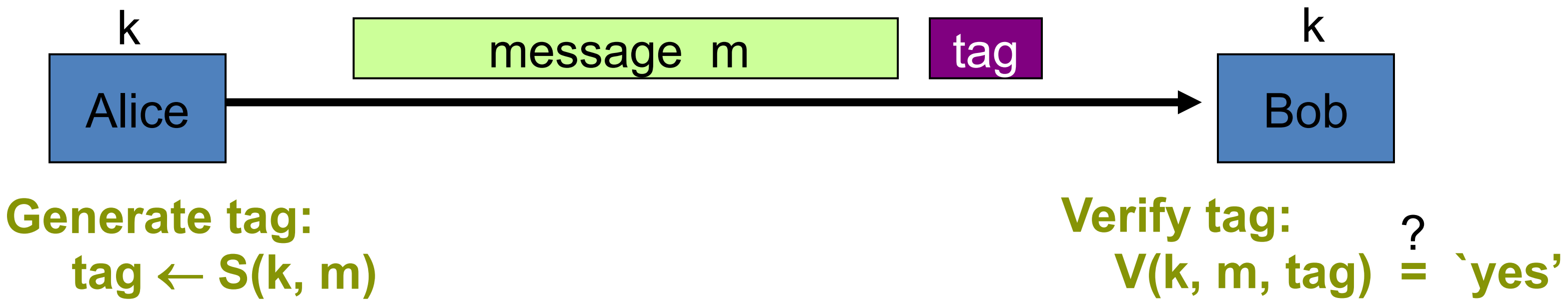
Need also to defend against active (tampering) attacks.

CTR mode is insecure against active attacks!

Next: methods to ensure message integrity

Message Integrity: MACs

- Goal: provide message integrity. No confidentiality.
  - ex: Protecting public binaries on disk.



Construction: HMAC (Hash-MAC)

Most widely used MAC on the Internet.

H: hash function.
example: SHA-256 ; output is 256 bits

Building a MAC out of a hash function:

– Standardized method: HMAC

$$S(k, msg) = H\big(k \oplus opad \;\|\; \mathbf{H(k \oplus ipad \,\|\, msg)} \big)$$

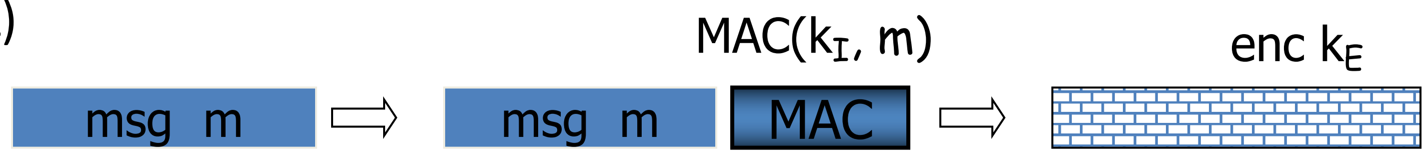
Why is this MAC construction secure?

… see the crypto course (cs255)

Combining MAC and ENC (Auth. Enc.)

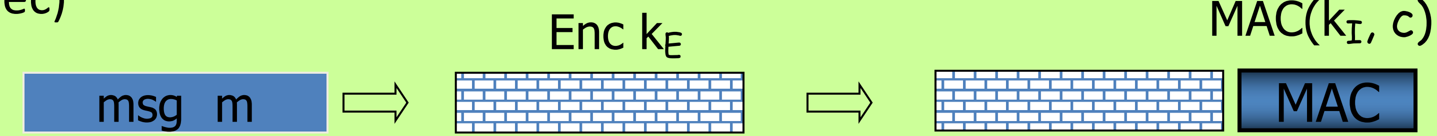
Encryption key k_E. MAC key = k_I

Option 1: (SSL)

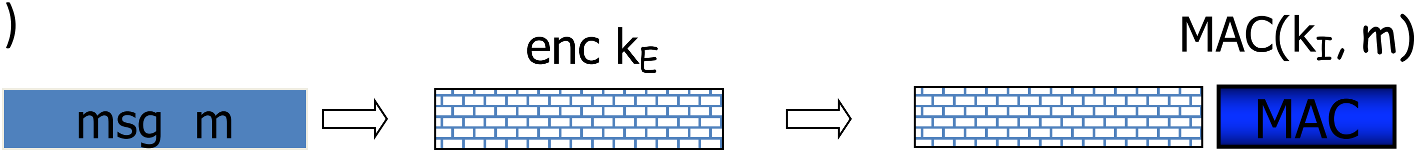


Option 2: (IPsec)

always correct

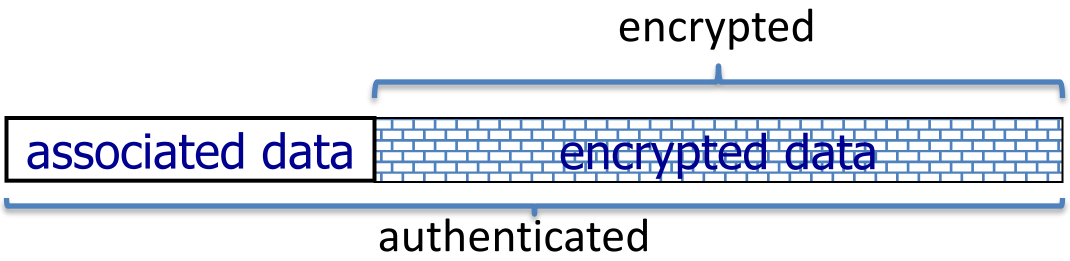


Option 3: (SSH)



AEAD: Auth. Enc. with Assoc. Data

AEAD:



AES-GCM: CTR mode encryption then MAC

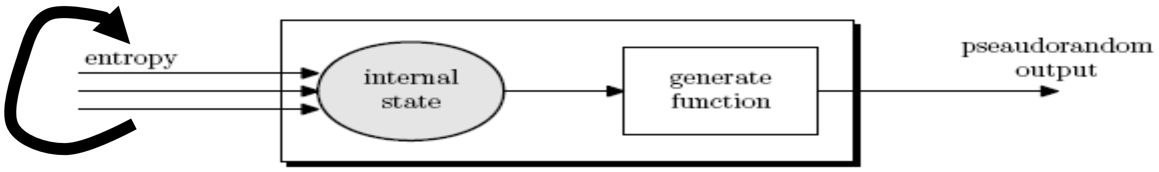
(MAC accelerated via Intel's PCLMULQDQ instruction)

Example AES-GCM encryption function

```
int encrypt(
    unsigned char *key,                                         // key
    unsigned char *iv,   int iv_len,                            // nonce
    unsigned char *plaintext,    int plaintext_len,             // plaintext
    unsigned char *aad,   int aad_len,                          // assoc. data

    unsigned char *ciphertext                                   // output ct
)
```

Generating Randomness (e.g. keys, nonces)



Pseudo random generators in practice: (e.g. /dev/random)

- Continuously add entropy to internal state
- Entropy sources:
    - Hardware RNG: Intel **RdRand** inst. (Ivy Bridge). 3Gb/sec.
    - Timing: hardware interrupts (keyboard, mouse)

Summary

Shared secret key:

- Used for secure communication and document encryption

Encryption: (eavesdropping security) **[should not be used standalone]**

- One-time key: stream ciphers, CTR with fixed IV
- Many-time key: CTR with random IV

Integrity: HMAC or CW-MAC

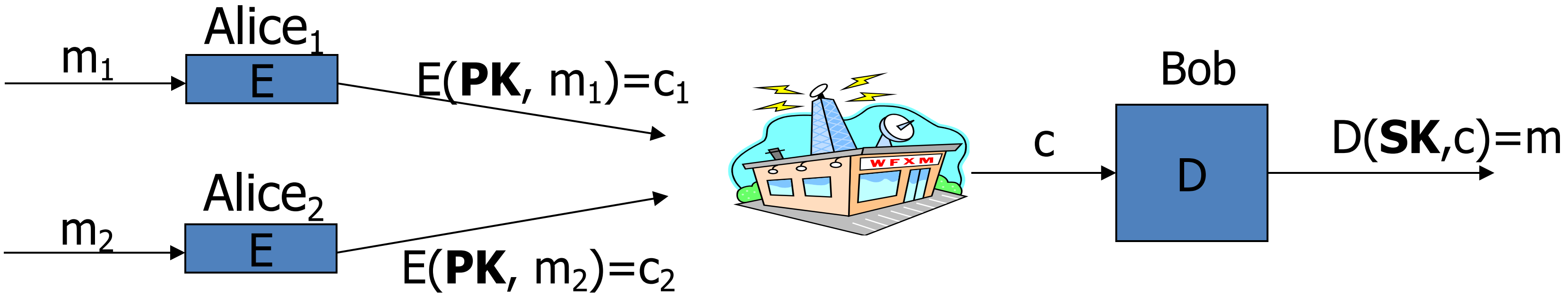
Authenticated encryption: encrypt-then-MAC using GCM

Crypto Concepts

Public key cryptography

Public-key encryption

Tool for managing or generating symmetric keys



- E – Encryption alg. PK – Public encryption key
- D – Decryption alg. SK – Private decryption key

Algorithms E, D are publicly known.

Building block: trapdoor permutations

1. Algorithm KeyGen: outputs pk and sk

2. Algorithm $F(pk, \cdot)$: a one-way function
   - Computing $y = F(pk, x)$ is easy
   - One-way: given random y finding x s.t. $y = F(pk,x)$ is difficult

3. Algorithm $F^{-1}(sk, \cdot)$: Invert $F(pk, \cdot)$ using trapdoor SK

$$F^{-1}(sk, y) = x$$

Example: RSA

1. KeyGen: generate two equal length primes p, q

   set $N \leftarrow p \cdot q$ (3072 bits $\approx$ 925 digits)

   set $e \leftarrow 2^{16}+1 = 65537$; $d \leftarrow e^{-1} \pmod{\varphi(N)}$

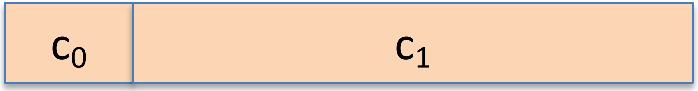
   $pk = (N, e)$; $sk = (N, d)$

2. RSA(pk, x) : $x \rightarrow (x^e \bmod N)$

   Inverting this function is believed to be as hard as factoring N

3. $RSA^{-1}(pk, y)$: $y \rightarrow (y^d \bmod N)$

Public Key Encryption with a TDF



KeyGen: generate pk and sk

Encrypt(pk, m):

- choose random $x \in$ domain(F) and set $k \leftarrow H(x)$
- $c_0 \leftarrow F(pk, x)$, $c_1 \leftarrow E(k, m)$ (E: symmetric cipher)
- send $c = (c_0, c_1)$

Decrypt$\big($sk, $c=(c_0,c_1)$ $\big)$: $x \leftarrow F^{-1}(sk, c_0)$, $k \leftarrow H(x)$, $m \leftarrow D(k, c_1)$

security analysis in crypto course

Digital signatures

Goal: bind document to author

- Problem: attacker can copy Alice's sig from one doc to another

Main idea: make signature depend on document

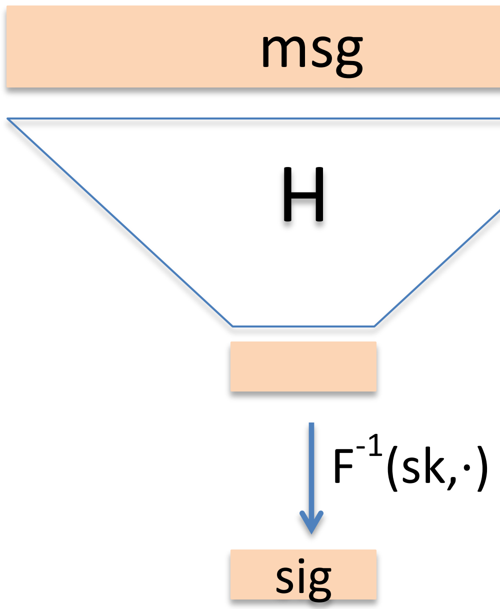
Example: signatures from trapdoor functions (e.g. RSA)

sign(sk, m) := F^{-1} (sk, H(m))

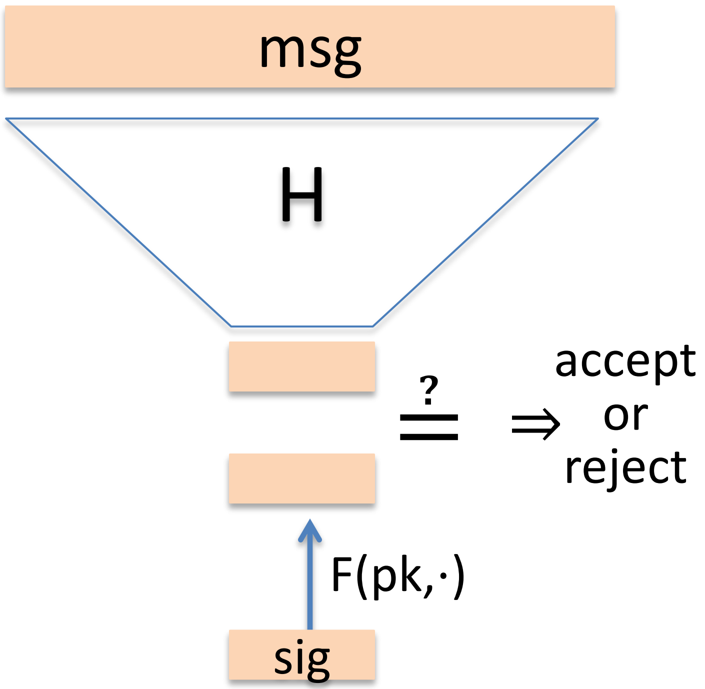
verify(pk, m, sig) := accept if F(pk, sig) = H(m)

Digital Sigs. from Trapdoor Functions

sign(sk, msg):

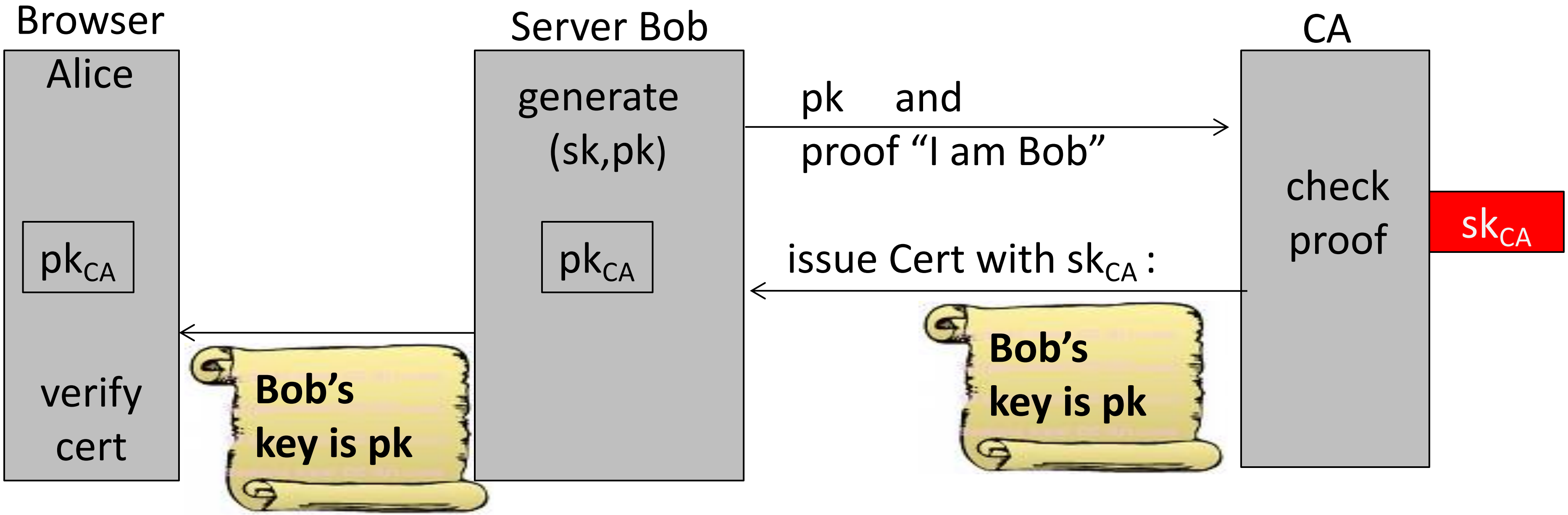


verify(pk, msg, sig):



Certificates: bind Bob's ID to his PK

How does Alice (browser) obtain Bob's public key pk_{Bob} ?



Bob uses Cert for an extended period (e.g. one year)

Sample certificate:

mail.google.com
Issued by: Google Internet Authority G3
Expires: Wednesday, June 20, 2018 at 6:25:00 AM Pacific Daylight Time
✔ This certificate is valid

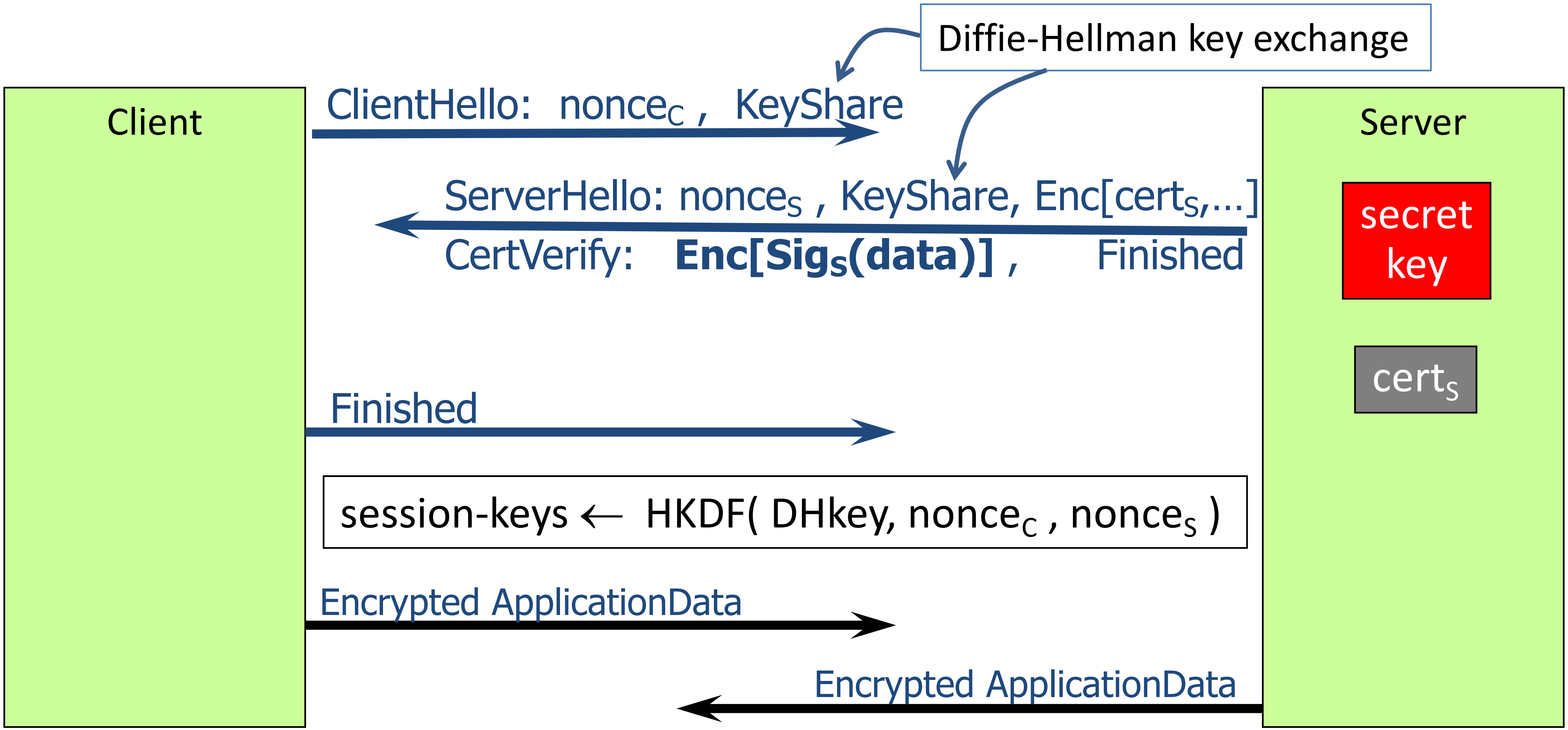
▼ Details

Subject Name	
Country	US
State/Province	California
Locality	Mountain View
Organization	Google Inc
Common Name	mail.google.com

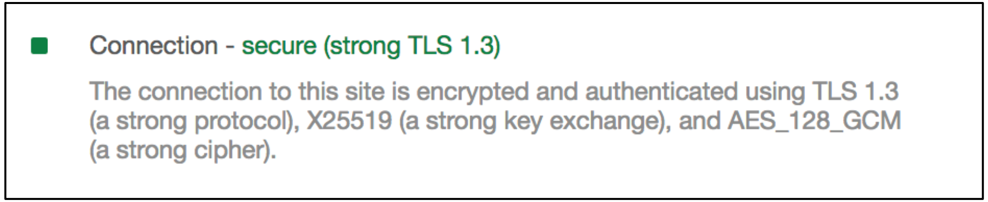
Issuer Name	
Country	US
Organization	Google Trust Services
Common Name	Google Internet Authority G3
Serial Number	3495829599616174946
Version	3
Signature Algorithm	SHA-256 with RSA Encryption

Public Key Info	
Algorithm	Elliptic Curve Public Key (1.2.840.10045.2.1)
Parameters	Elliptic Curve secp256r1 (1.2.840.10045.3.1.7)
Public Key	65 bytes : 04 D5 63 FC 4D F9 4E 91 ...
Key Size	256 bits
Key Usage	Encrypt, Verify, Derive
Signature	256 bytes : 3F FE 04 7B BE B0 32 1D ...

TLS 1.3 session setup (simplified)



Properties



Gmail

Nonces: prevent replay of an old session

Forward secrecy: server compromise does not expose old sessions

Some identity protection: certificates are sent encrypted

One sided authentication:

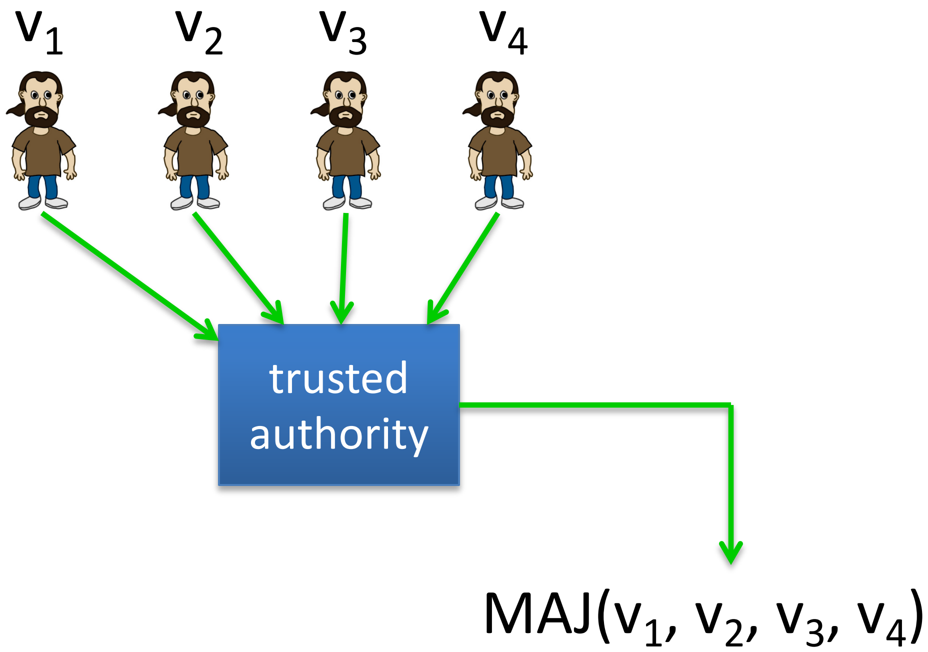
- Browser identifies server using server-cert
- TLS has support for mutual authentication
  - Rarely used: requires a client pk/sk and client-cert

Crypto Concepts

A brief sample of advanced crypto

Protocols

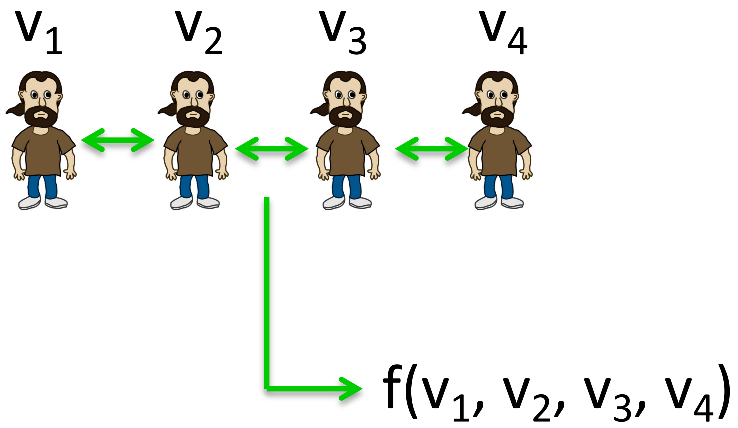
- Elections



Can we do the same without a trusted party?

Protocols

- Elections
- Private auctions



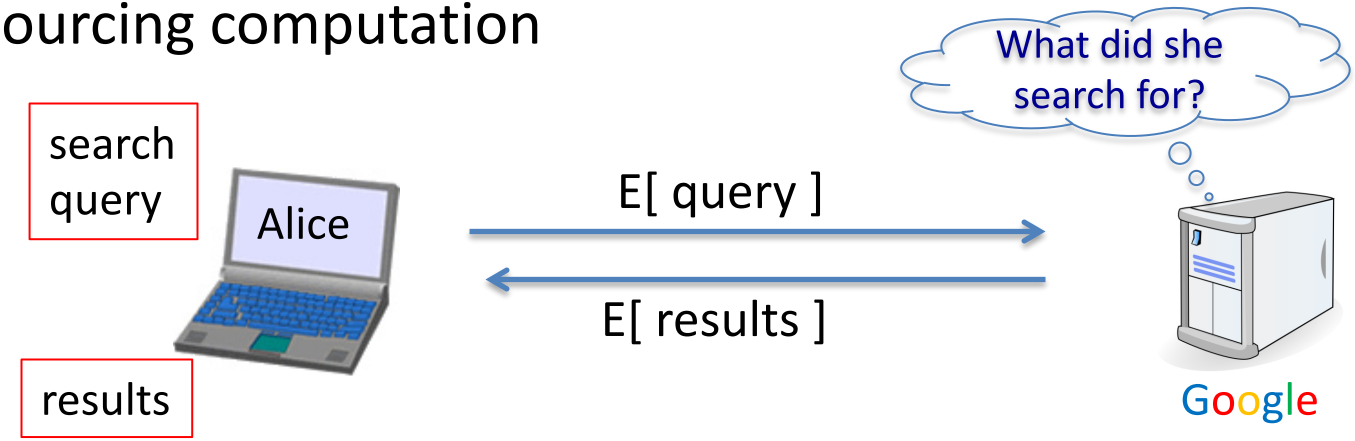
Goal: compute $f(v_1, v_2, v_3, v_4)$

"Thm:" anything that can be done with a trusted authority can also be done without

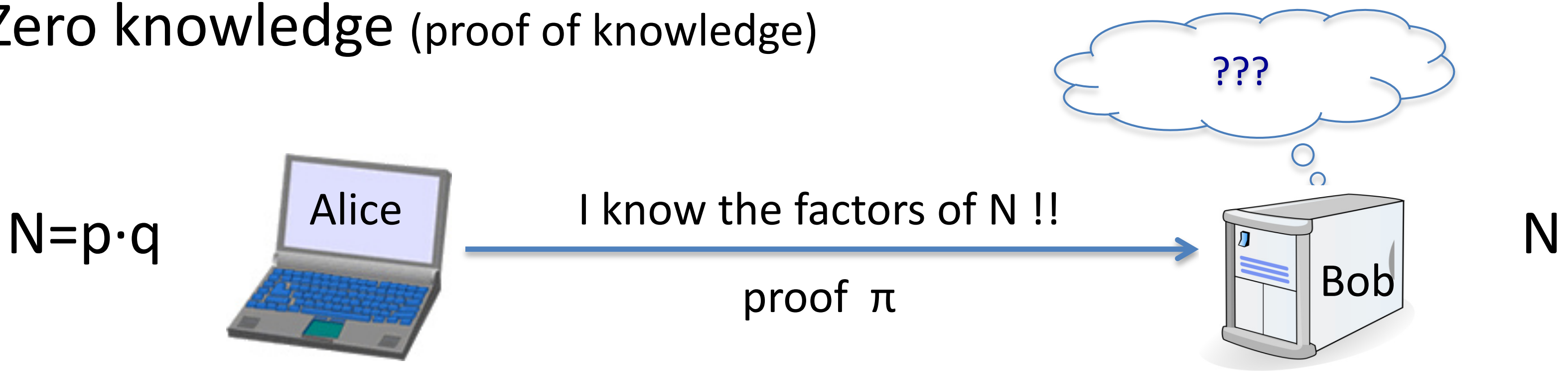
- Secure multi-party computation

Magical applications

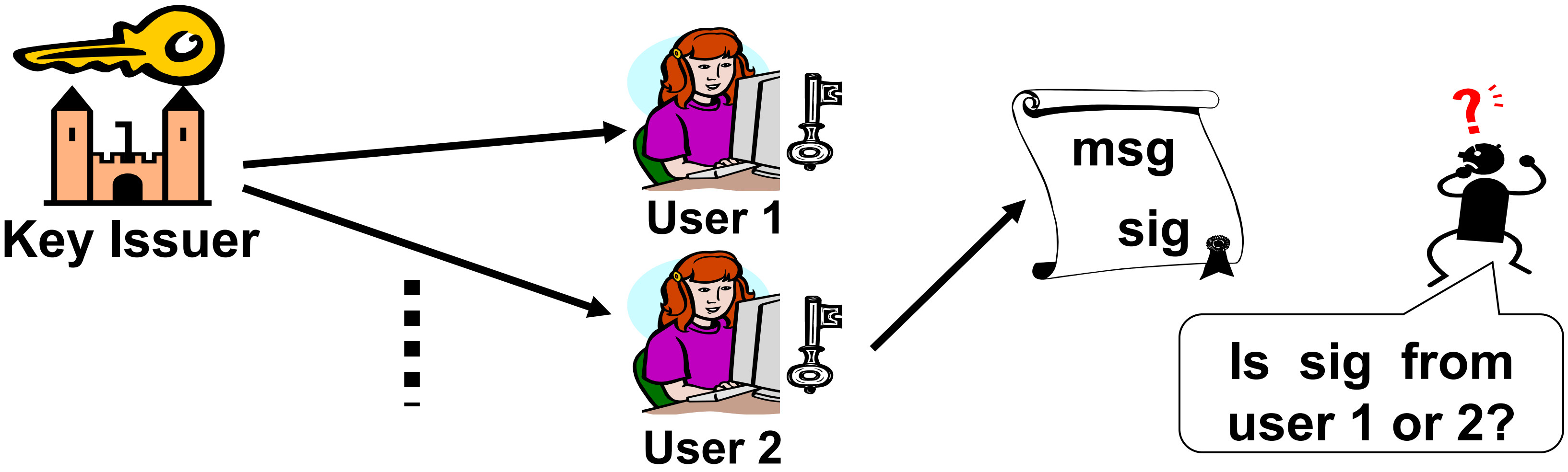
- Privately outsourcing computation



- Zero knowledge (proof of knowledge)



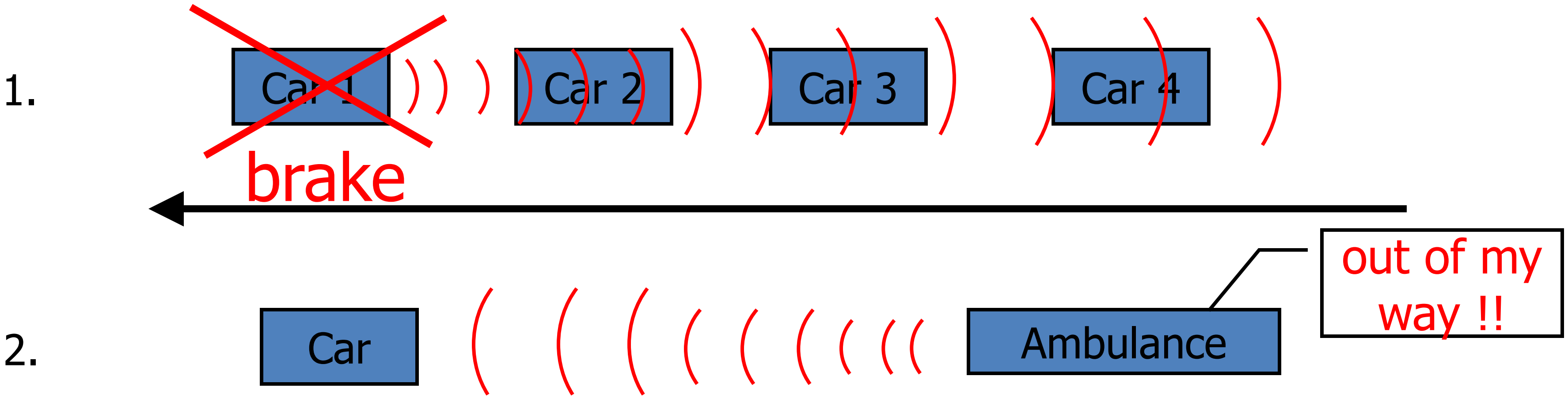
Privacy: Group Signatures



Simple solution: give all users same private key

… but also need to revoke signers when they misbehave

Example: Vehicle Safety Comm. (VSC)



Require authenticated (signed) messages from cars.

- Prevent impersonation and DoS on traffic system.

<u>Privacy problem</u>: cars broadcasting <u>signed</u> (x,y, **V**).

Clean solution: group sigs. Group = set of all cars.

Summary: crypto concepts

Symmetric cryptography:

Authenticated Encryption (AE) and message integrity

Public-key cryptography:

Public-key encryption, digital signatures, key exchange

Certificates: bind a public key to an identity using a CA

- Used in TLS to identify server (and possibly client)

Modern crypto: goes far beyond basic encryption and signatures