Crypto Concepts

Symmetric encryption, Public key encryption, and TLS

Cryptography

Is:

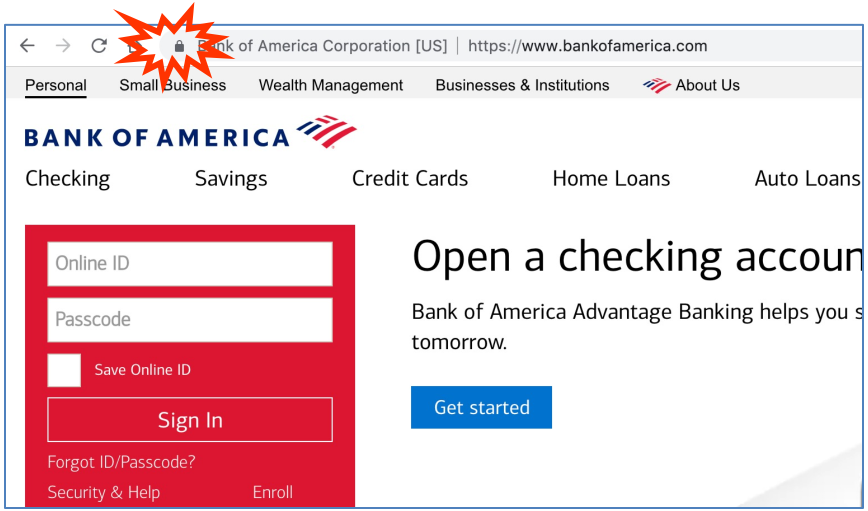
- A tremendous tool for protecting information
- The basis for many security mechanisms

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

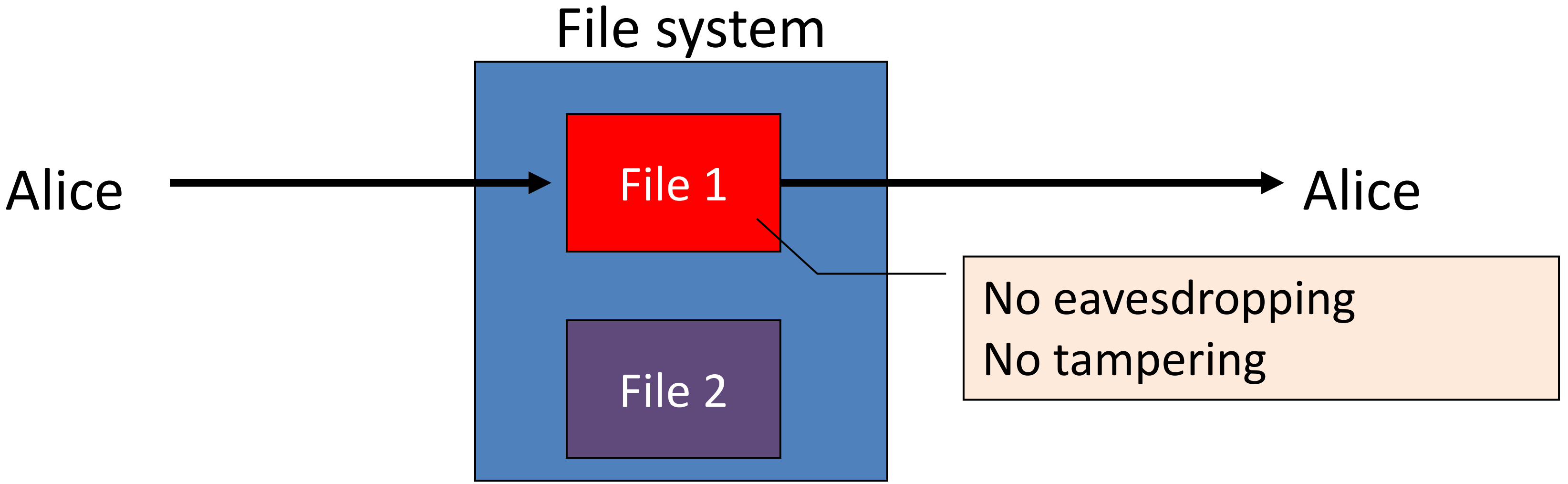
1. Handshake Protocol: **Establish shared secret key using public-key cryptography**

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

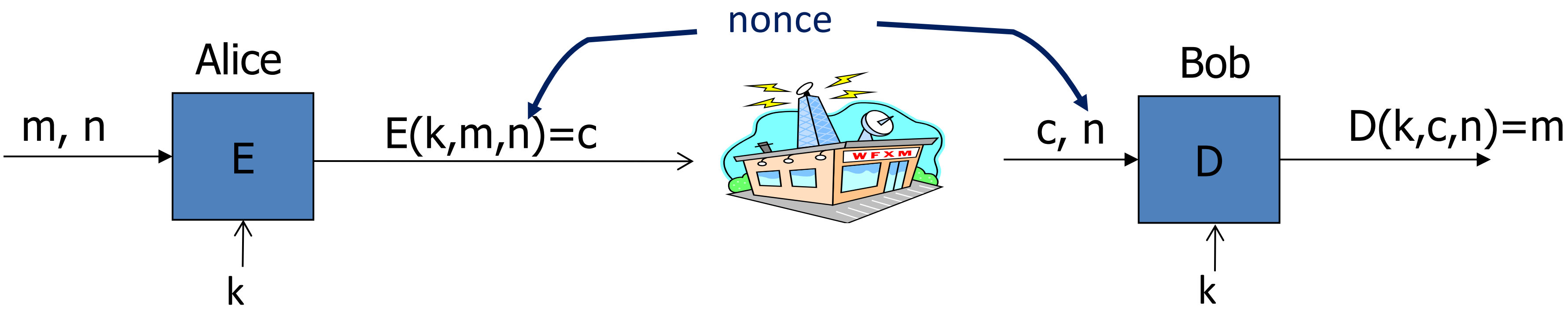
Our starting point: Using a key for encryption and integrity

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

⇒   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 is used to encrypt multiple messages or multiple files
  - TLS: same key used to encrypt many frames
- 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: $D(k, c) = c \oplus k = (m \oplus k) \oplus k = m$

One Time Pad (OTP) Security

Shannon (1949):

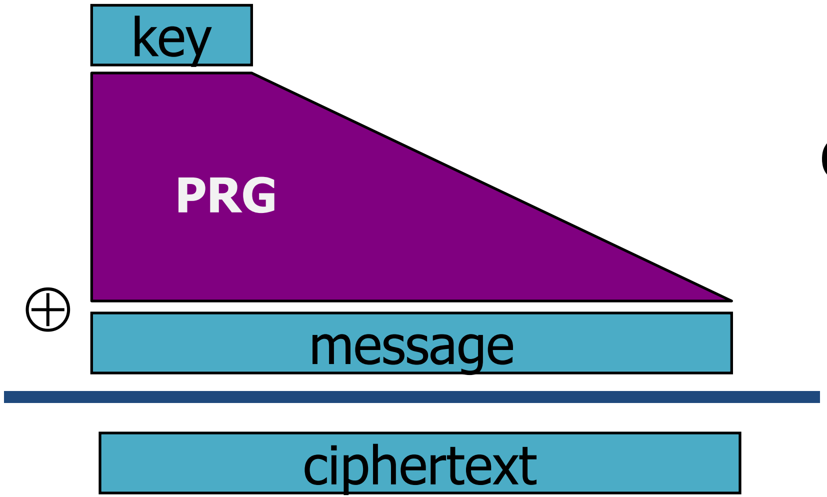
- OTP is “secure” against one-time eavesdropping
- without key, ciphertext reveals no “information” about plaintext

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?

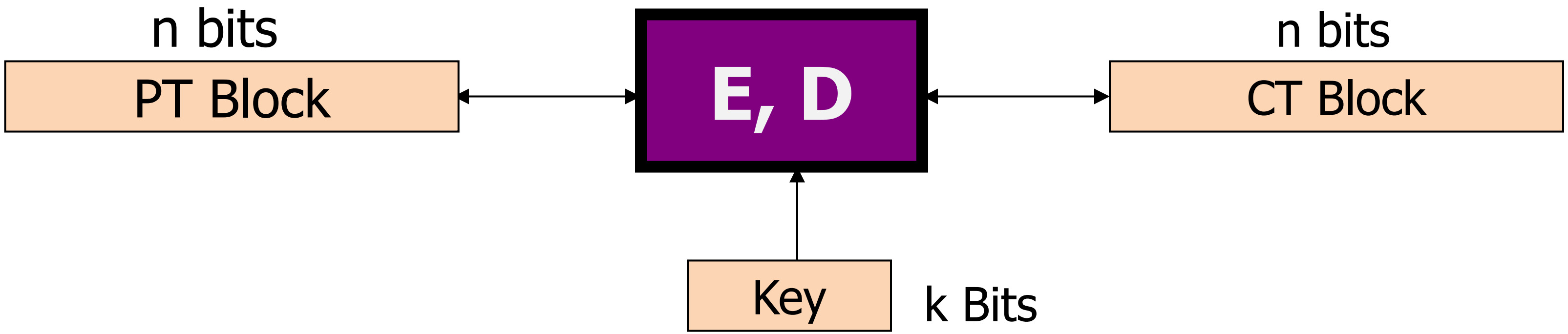
Eavesdropper does:

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

Enough redundant information in English that:

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

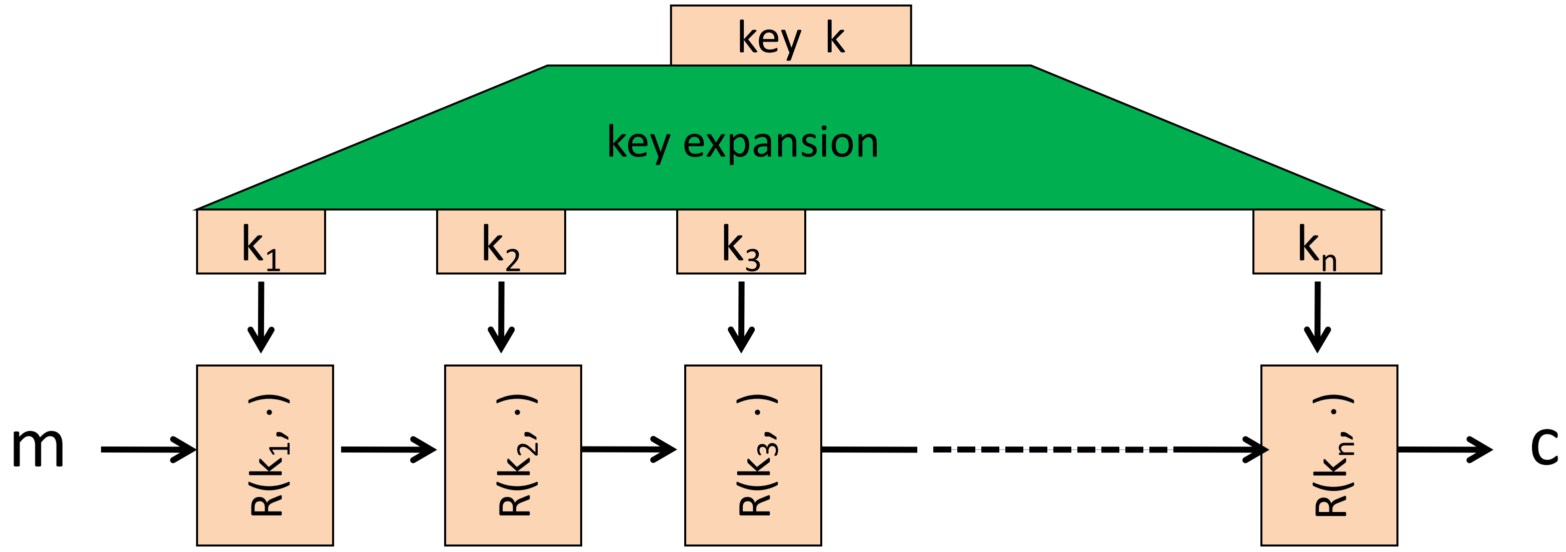
Block ciphers: crypto work horse



Canonical examples:

1. 3DES (old):   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 AES128: 10 rounds, AES256: n=14 rounds

AES-NI: AES in hardware (Intel, AMD, ARM)

New x86 hardware instructions used to implement AES:

- **aesenc**, **aesenclast**: one round of AES

  aesenc xmm1, xmm2 (result written to xmm1)

  xmm1 = state, xmm2 = round key

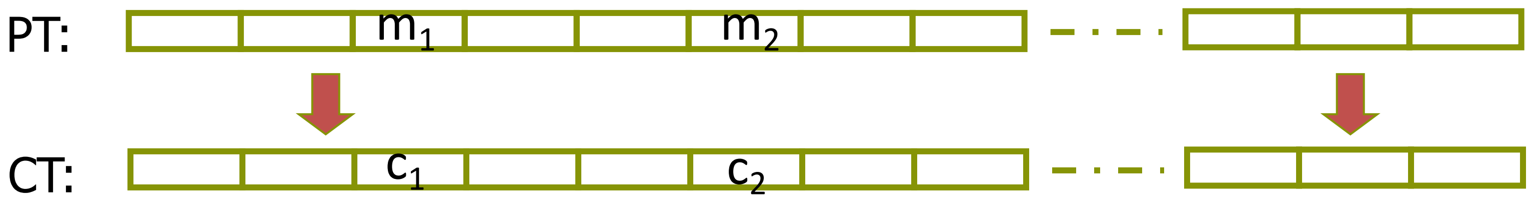
- **aesdec**, **aesdeclast**: one round of AES
- **aeskeygenassist**: do AES key expansion

$\Longrightarrow$ more than 10x speedup over a software AES

$\Longrightarrow$ better security: all AES instructions are **constant time**

Incorrect use of block ciphers

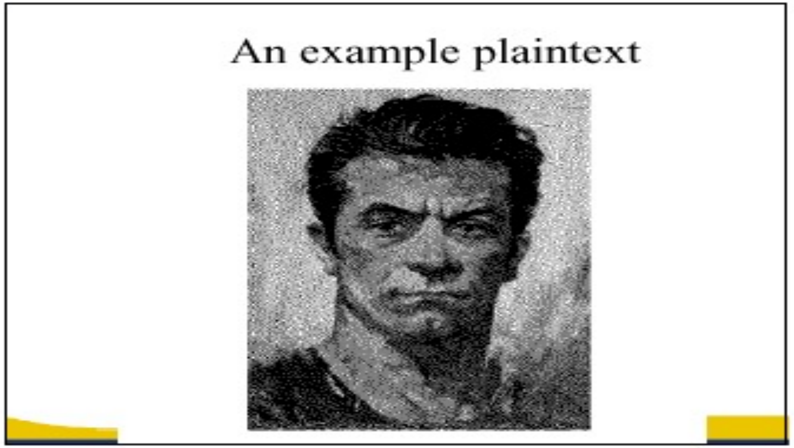
Electronic Code Book (ECB):

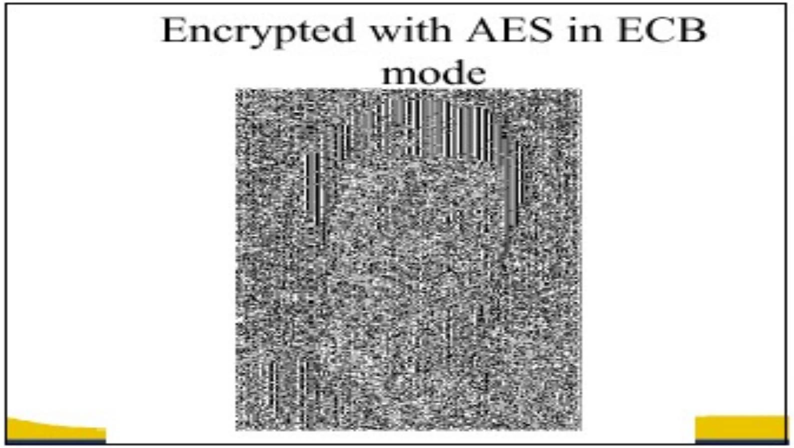


Problem:

– if $m_1=m_2$ then $c_1=c_2$

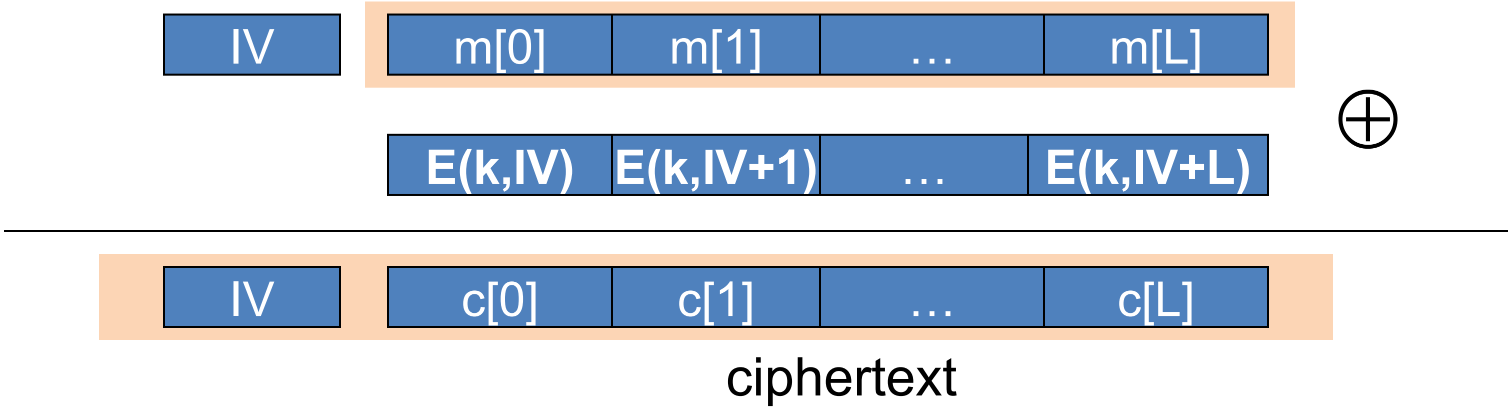
In pictures





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)

A Warning

eavesdropping security is insufficient for most applications

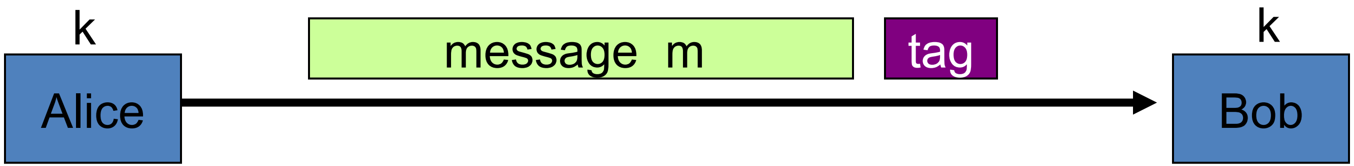
Need also to defend against active (tampering) attacks.

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.



Generate tag:
tag ← S(k, m)

Verify tag:
V(k, m, tag) $\stackrel{?}{=}$ `yes'

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

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

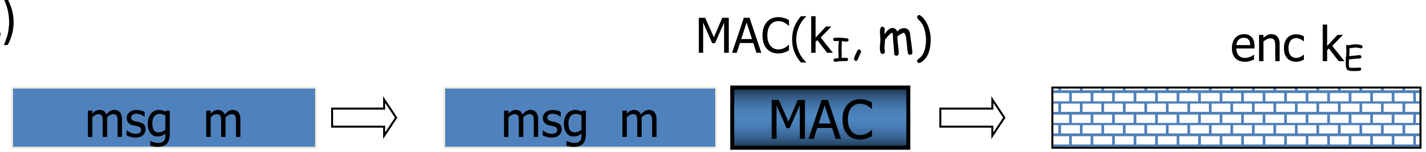
Why is this MAC construction secure?

… see the crypto course (cs255)

Combining MAC and ENC (Auth. Enc.)

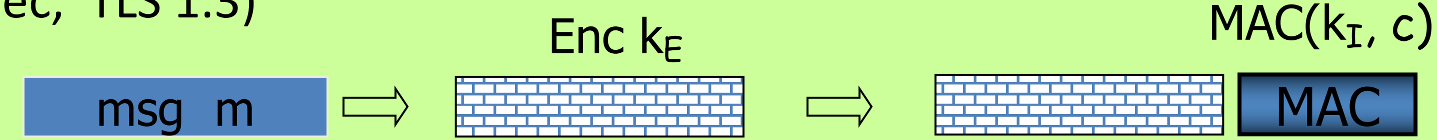
Encryption key k_E. MAC key = k_I

Option 1: (SSL)

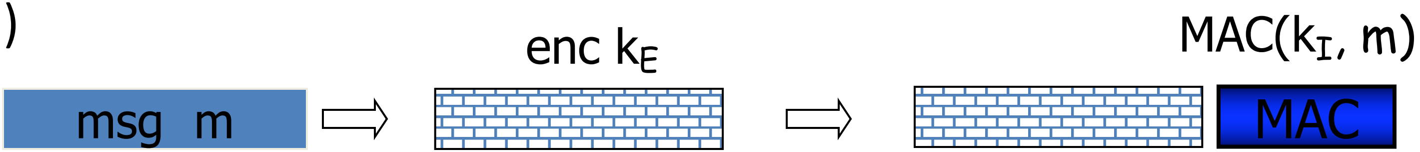


Option 2: (IPsec, TLS 1.3)

always correct

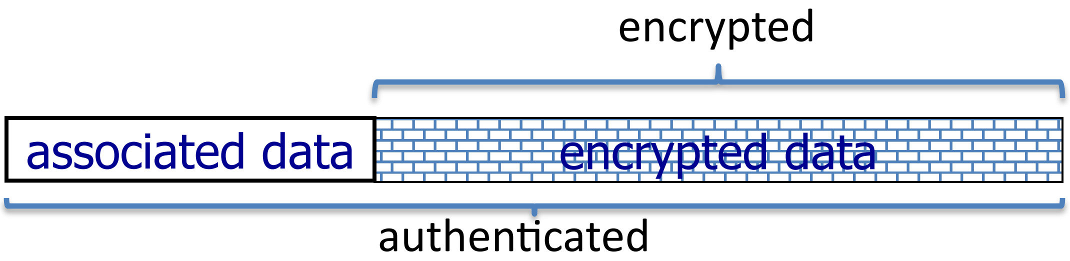


Option 3: (SSH)



AEAD: Auth. Enc. with Assoc. Data

AEAD:



AES-GCM: CTR mode encryption then MAC

(MAC accelerated via Intel's PCLMULQDQ instruction)

Example AES-GCM functions

```
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
```

```
int decrypt(                     // error if invalid MAC on  (aad, ciphertext)
      unsigned char *key,                                  // key
      unsigned char *ciphertext,   int ciphertext_len,     // plaintext
      unsigned char *aad,   int aad_len,                   // assoc. data
      - - - - - - -
      unsigned char *plainrtext )                          // output pt
```

Summary

Shared secret key:

- Used for secure communication and document encryption

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

- One-time key: ex: a stream cipher
- Many-time key: ex: AES-CTR with a unique/random nonce

Integrity: HMAC

Authenticated encryption: encrypt-then-MAC using AES-GCM

Crypto Concepts

encryption and compression problems

Encryption and compression: oil and vinegar

HTTP: uses compression to reduce bandwidth

Option 1: first encrypt and then compress

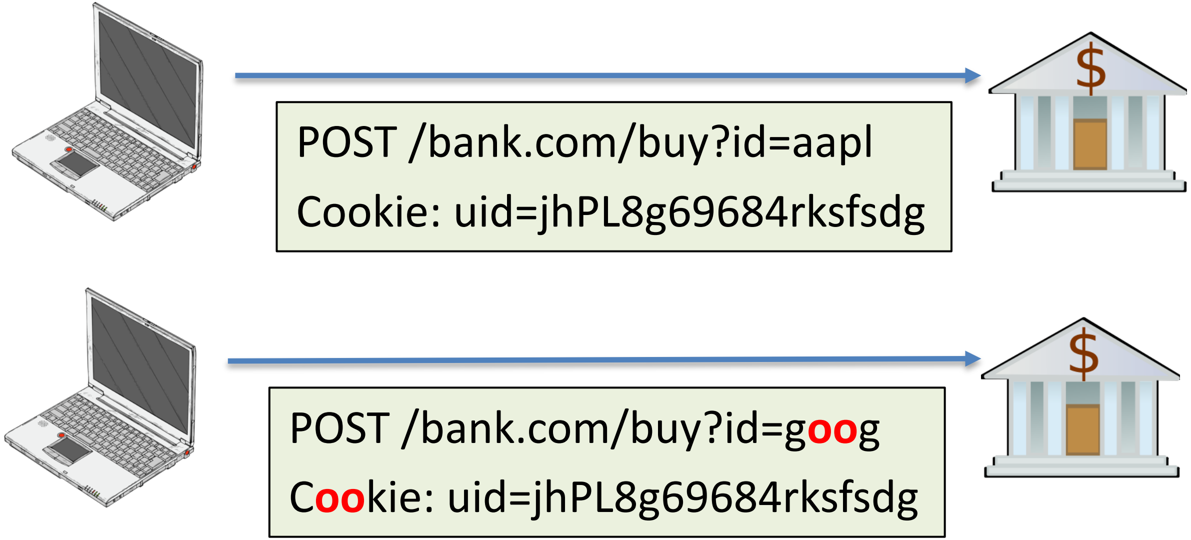
- Does not work … ciphertext looks like a random string

Option 2: first compress and then encrypt

- Used in many Internet protocols (TLS, HTTP, QUIC, …)
- Trouble …

Trouble … [Kelsey'02]

Compress-then-encrypt reveals information:

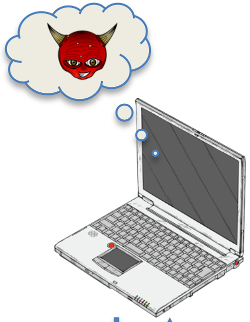


Second message compresses better than first:

network observer can distinguish the two messages!

Even worse: the CRIME attack [RD’2012]

Goal: steal user’s bank cookie



POST /bank.com/buy?id=aapl

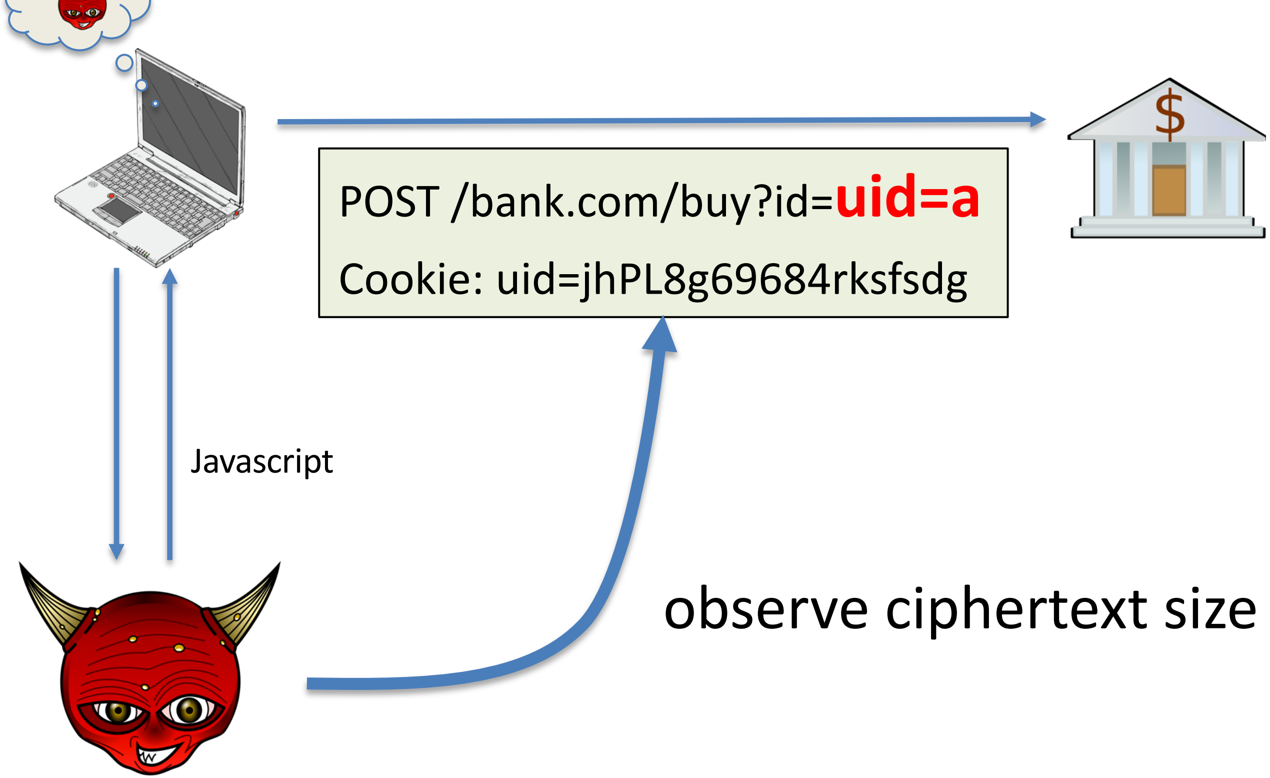
Cookie: uid=jhPL8g69684rksfsdg

Javascript

Javascript can issue requests to Bank, but cannot read Cookie value

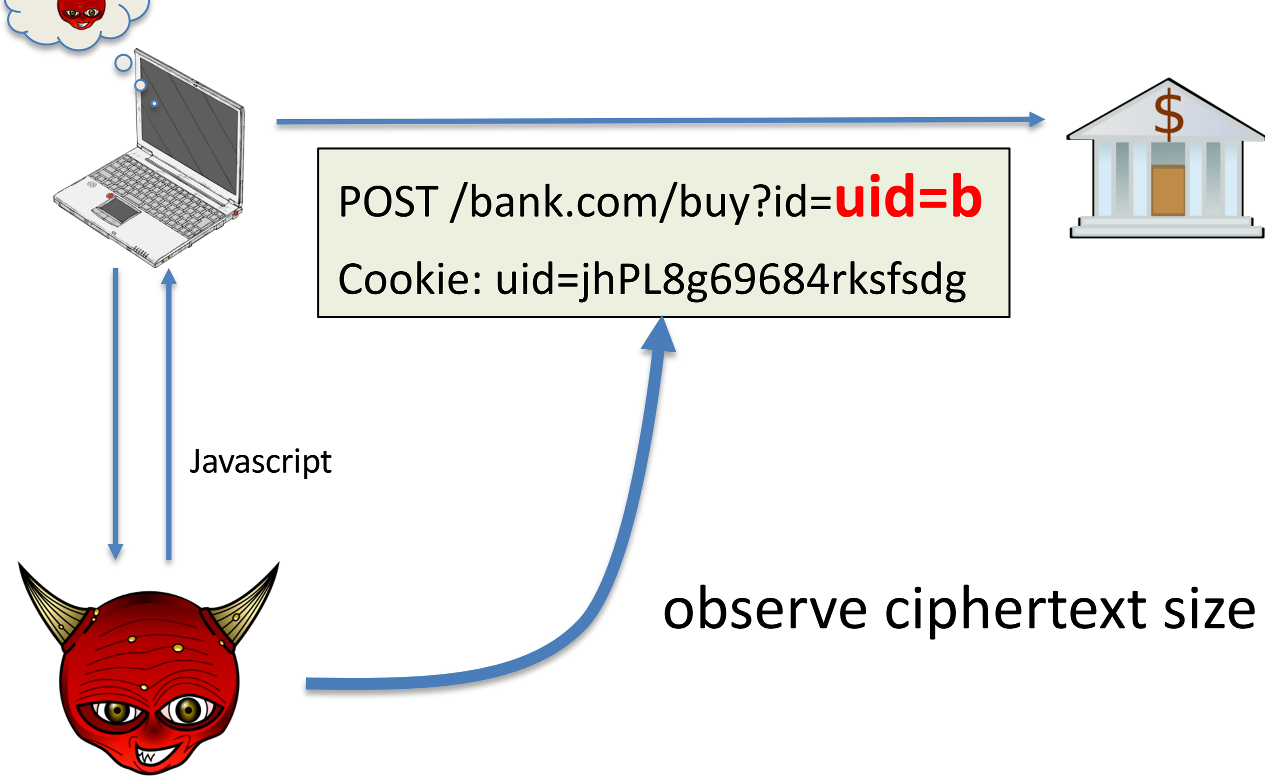
Even worse: the CRIME attack [RD'2012]

Goal: steal user's bank cookie



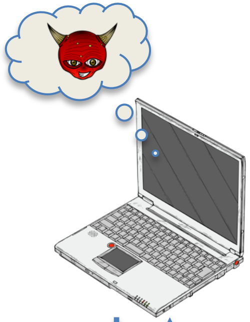
Even worse: the CRIME attack [RD’2012]

Goal: steal user’s bank cookie



Even worse: the CRIME attack [RD'2012]

Goal: steal user's bank cookie



POST /bank.com/buy?id=**uid=j**

Cookie: **uid=j**hPL8g69684rksfsdg

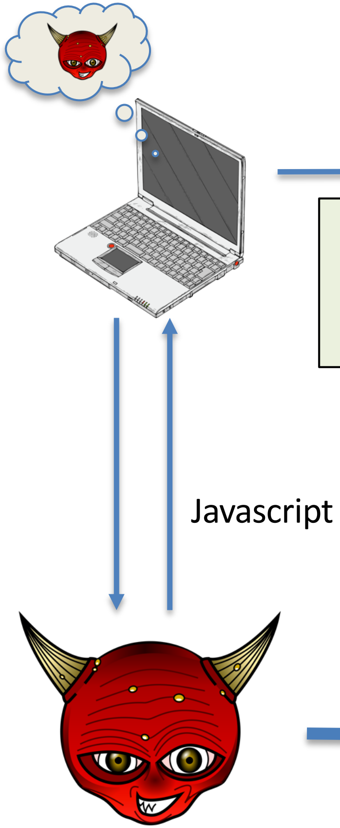
Javascript

ciphertext slightly shorter

$\Rightarrow$ first character of Cookie is "j"

Even worse: the CRIME attack [RD'2012]

Goal: steal user's bank cookie



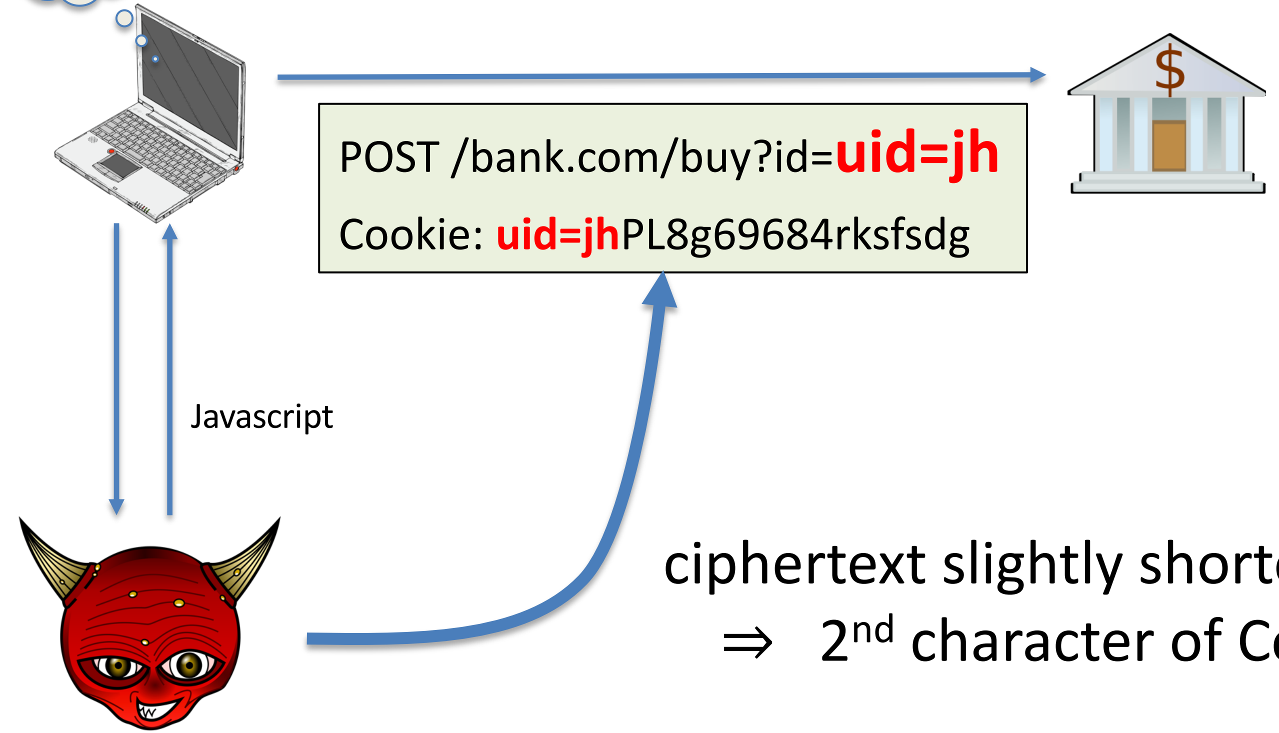
POST /bank.com/buy?id=**uid=ja**

Cookie: uid=jhPL8g69684rksfsdg

observe ciphertext size

Even worse: the CRIME attack [RD'2012]

Goal: steal user's bank cookie

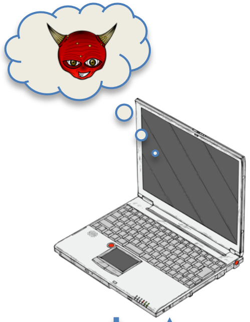


ciphertext slightly shorter

⇒ 2nd character of Cookie is "h"

Even worse: the CRIME attack [RD'2012]

Goal: steal user's bank cookie



POST /bank.com/buy?id=**uid=jh**

Cookie: **uid=jh**PL8g69684rksfsdg

Javascript

Recover entire cookie after
256 × |Cookie| tries

Takes several minutes (simplified)

What to do?

- Disable compression ☹️

- Use a different compression context for parts under Javascript control and parts that are not

- Change secret (Cookie) after every request

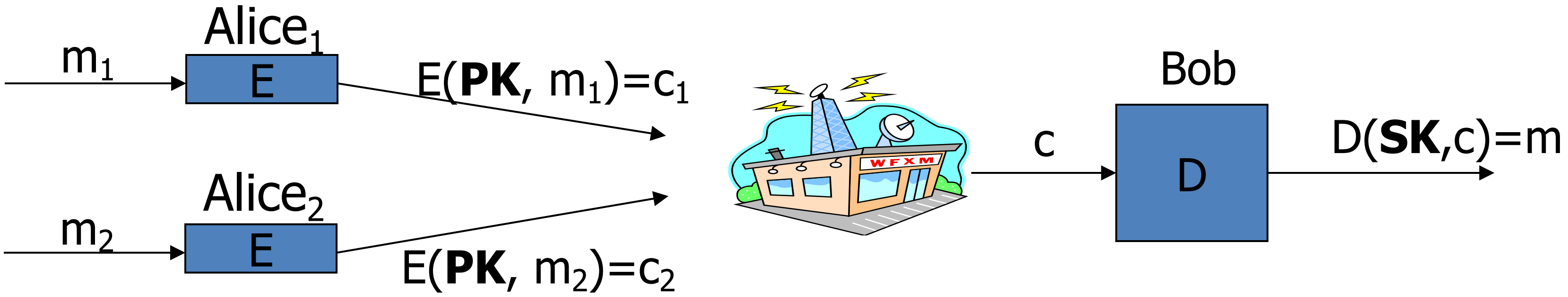
Does not eliminate inherent leakage due to compression

Crypto Concepts

Public key cryptography

(1) Public-key encryption

Tool for managing or generating symmetric keys



- E – Encryption alg. PK – <u>Public</u> encryption key
- D – Decryption alg. SK – <u>Private</u> 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 $e \leftarrow 2^{16}+1 = 65537$; $d \leftarrow e^{-1} \pmod{\varphi(N)}$

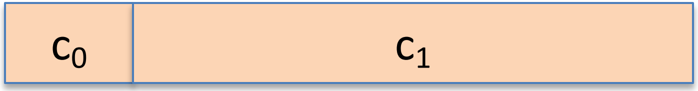
   $pk = (N, e)$; $sk = (N, d)$

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 (cs255)

(2) 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 a trapdoor permutation (e.g. RSA)

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

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

Digital signatures

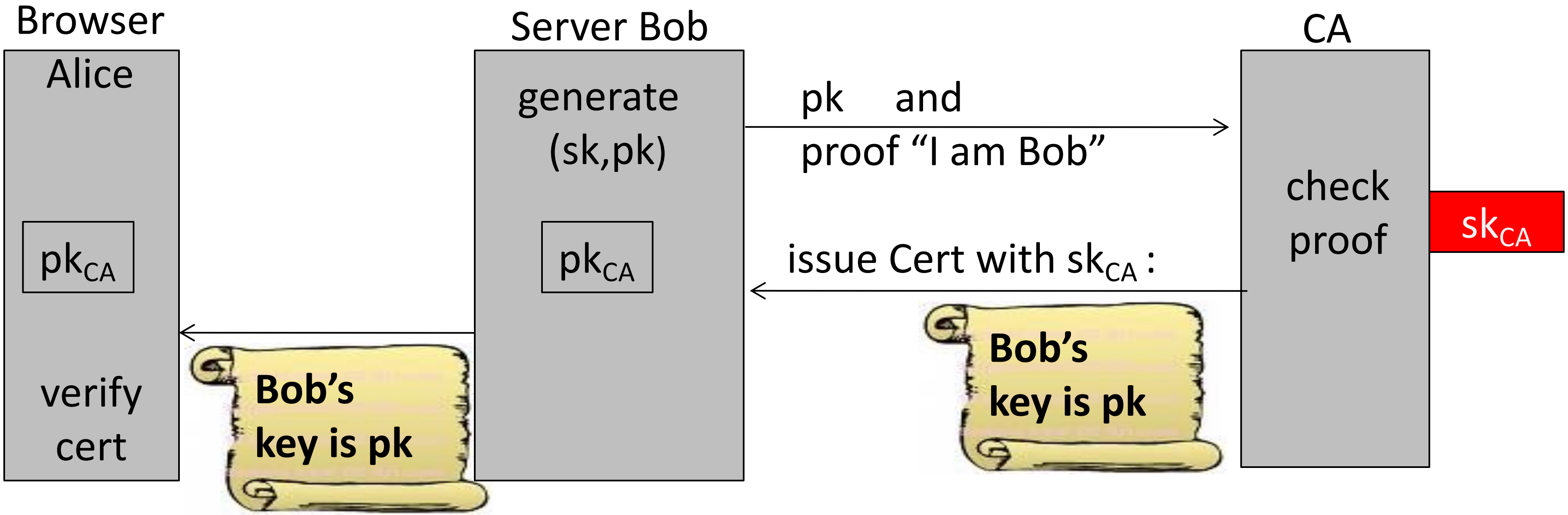
- Only someone who knows **sk** can sign a message m
- Anyone who has **pk** can verify a (msg, signature) pair

$$\text{sign}(sk, m) \quad := \quad F^{-1}(sk,\ H(m))$$

$$\text{verify}(pk, m, sig) \quad := \quad \text{accept if} \quad F(pk, sig) = H(m)$$

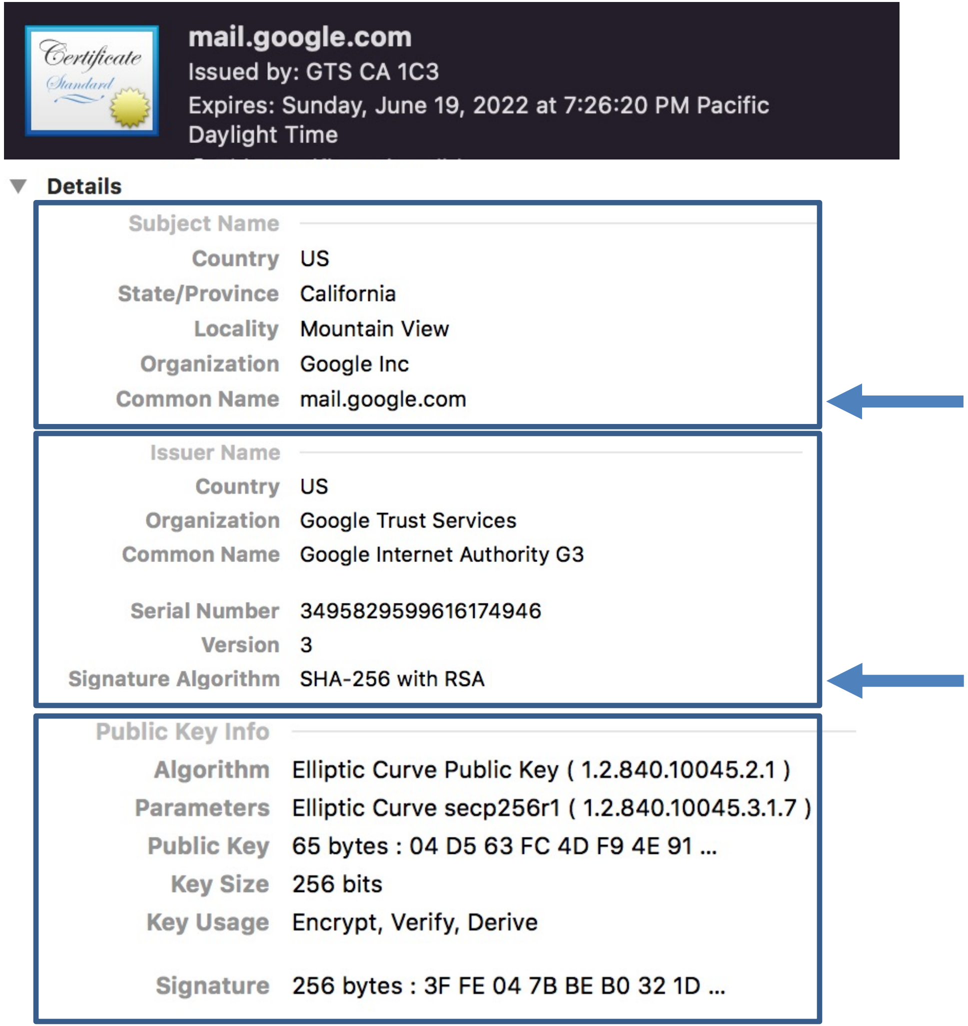
Certificates: bind Bob's ID to a PK

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



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

Sample certificate:



Signature schemes used in the real world

RSA signature scheme:

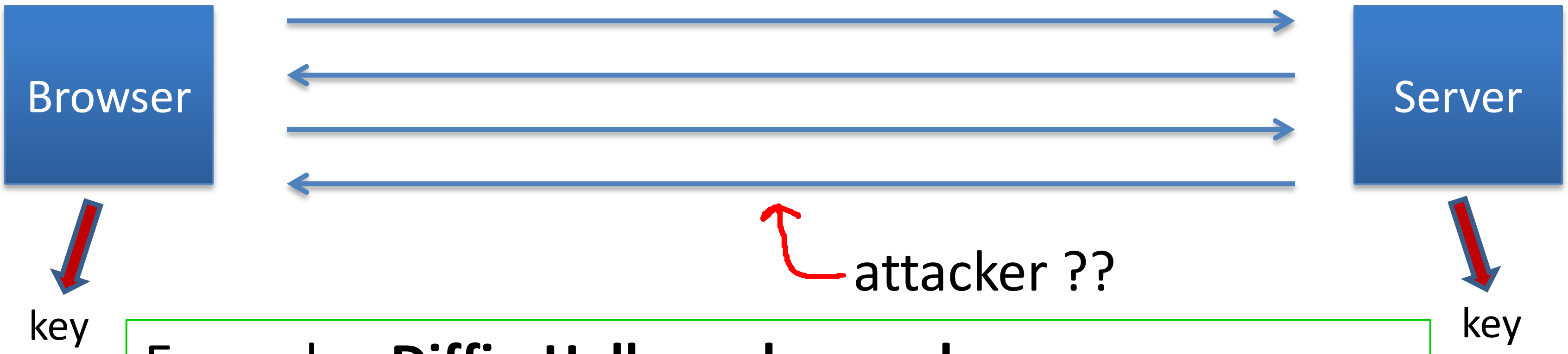
- Fast to verify, but signatures are long
- Often used in certificates

ECDSA, Schnorr, BLS signature schemes:

- Faster to generate signature and more compact than RSA
- Used everywhere, other than web certificates

(3) Key exchange

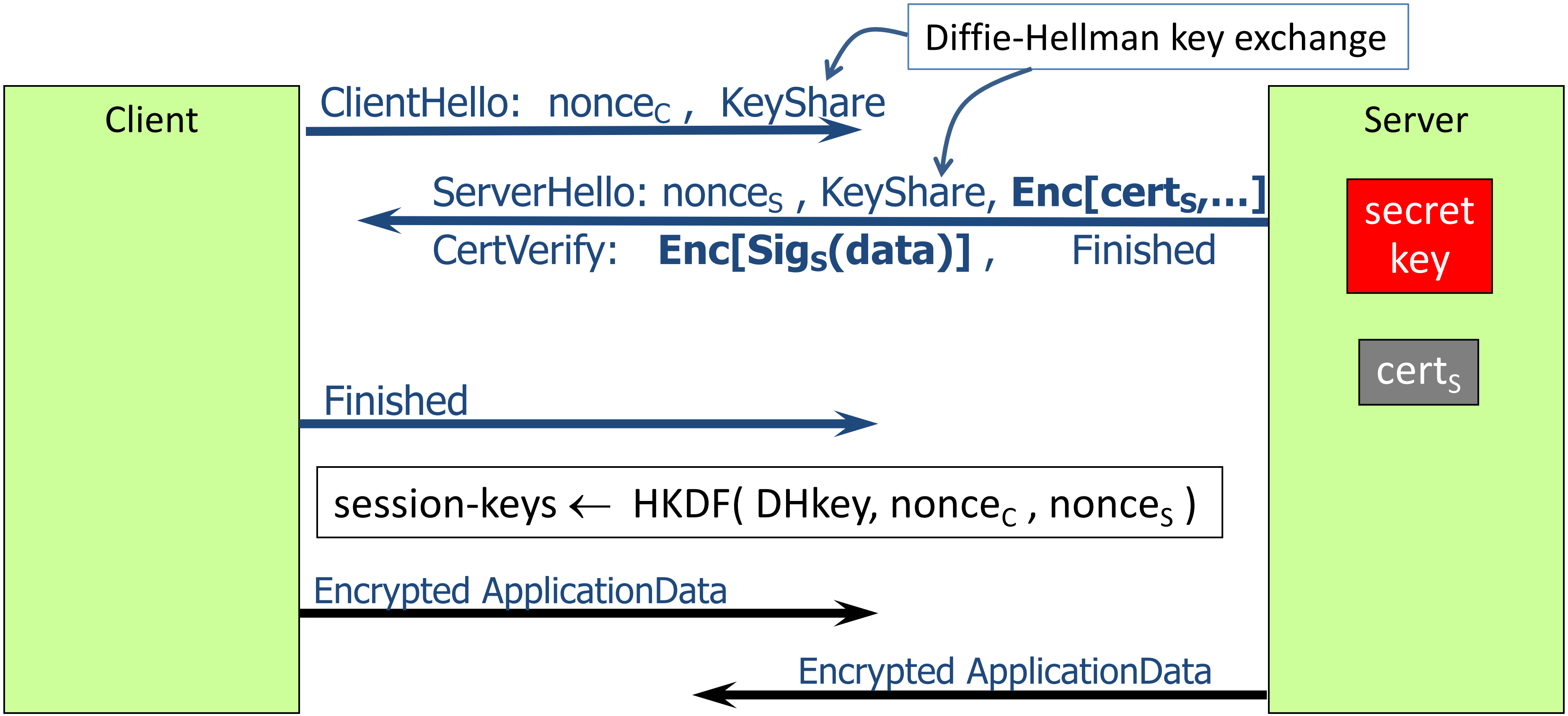
Goal: Browser and Server want a shared secret, unknown to attacker



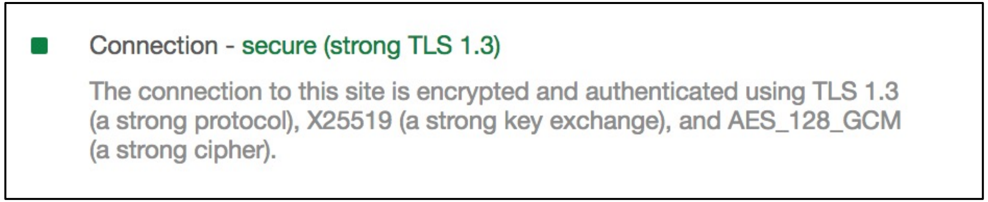
Example: **Diffie-Hellman key exchange**.

- Only secure against eavesdropping
- TLS 1.3: enhances Diffie-Hellman key exchange
  $\Longrightarrow$ security against an active attacker

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
  - requires a client pk/sk and client-cert

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