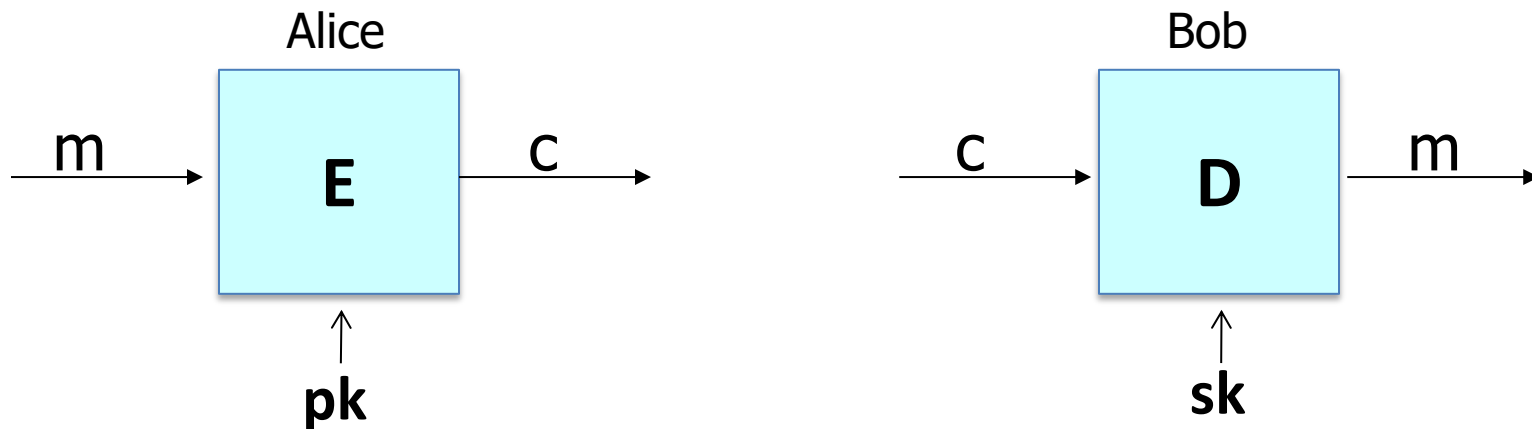# Public Key Encryption
from trapdoor permutations

This slide is made based the online course of Cryptography by Dan Boneh
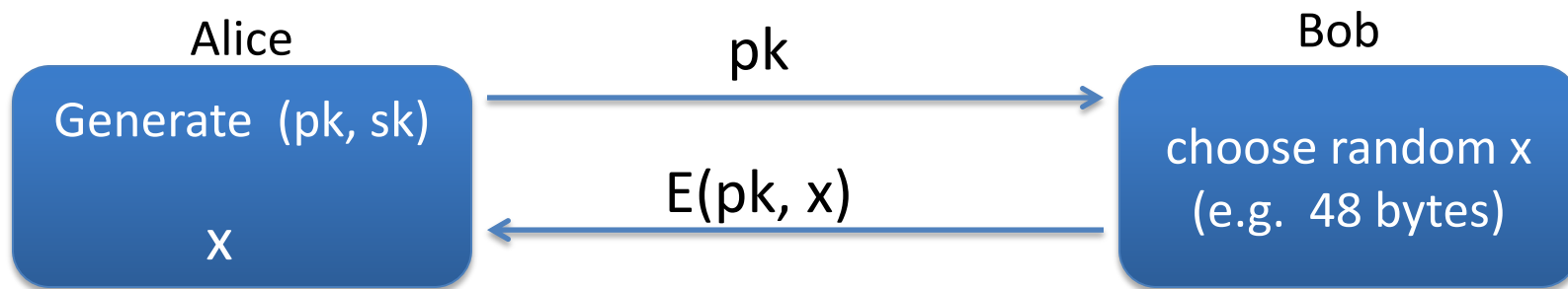
Public key encryption

Bob: generates   (PK, SK)   and gives PK to Alice

Alice

m → **E** → c

↑
pk

Bob

c → **D** → m

↑
sk

Dan Boneh

Applications

Session setup   (for now, only eavesdropping security)

Alice
                                  pk
Generate (pk, sk)   →   Bob

                              E(pk, x)   choose random x
        x   ←   (e.g. 48 bytes)

Non-interactive applications: (e.g. Email)

- Bob sends email to Alice encrypted using pk_{alice}

- Note: Bob needs pk_{alice} (public key management)

Public key encryption

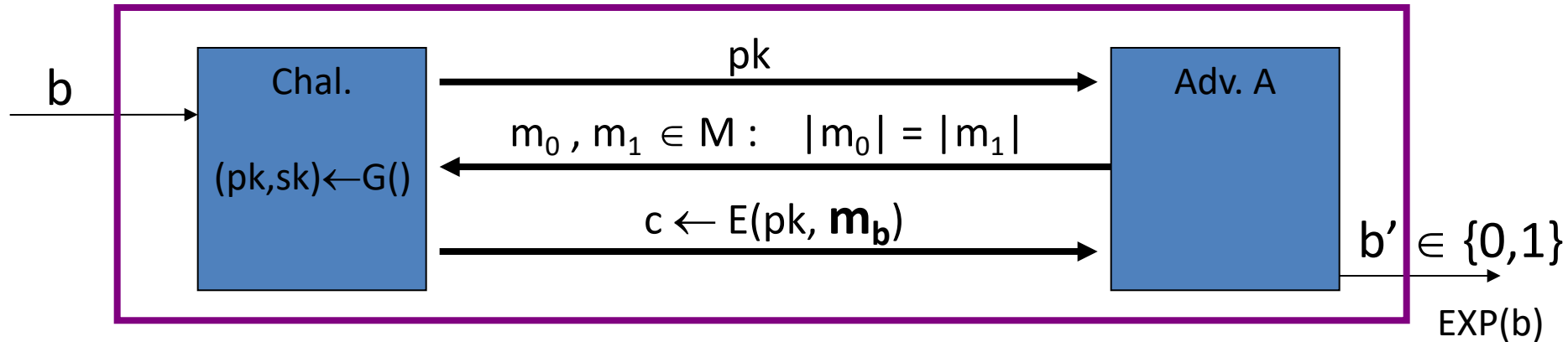**Def**: a public-key encryption system is a triple of algs.   (G, E, D)

- G(): randomized alg. outputs a key pair   (pk, sk)

- E(pk, m): randomized alg. that takes m∈M and outputs c ∈C

- D(sk,c): det. alg. that takes c∈C and outputs m∈M or ⊥

Consistency: ∀(pk, sk) output by G :

$$\forall m\in M: \quad D(sk, \ E(pk, m)) = m$$

Security: eavesdropping

For   b=0,1   define experiments EXP(0) and EXP(1) as:

Def: $\mathbb{E}$ =(G,E,D) is sem. secure (a.k.a IND-CPA) if for all efficient A:

$$\mathrm{Adv}_{SS} [A, \mathbb{E}] = \left| \Pr[\mathrm{EXP}(0)=1] - \Pr[\mathrm{EXP}(1)=1] \right| < \text{negligible}$$

Dan Boneh

Relation to symmetric cipher security

Recall: for symmetric ciphers we had two security notions:

- One-time security   and   many-time security (CPA)
- We showed that one-time security $\not\Rightarrow$ many-time security
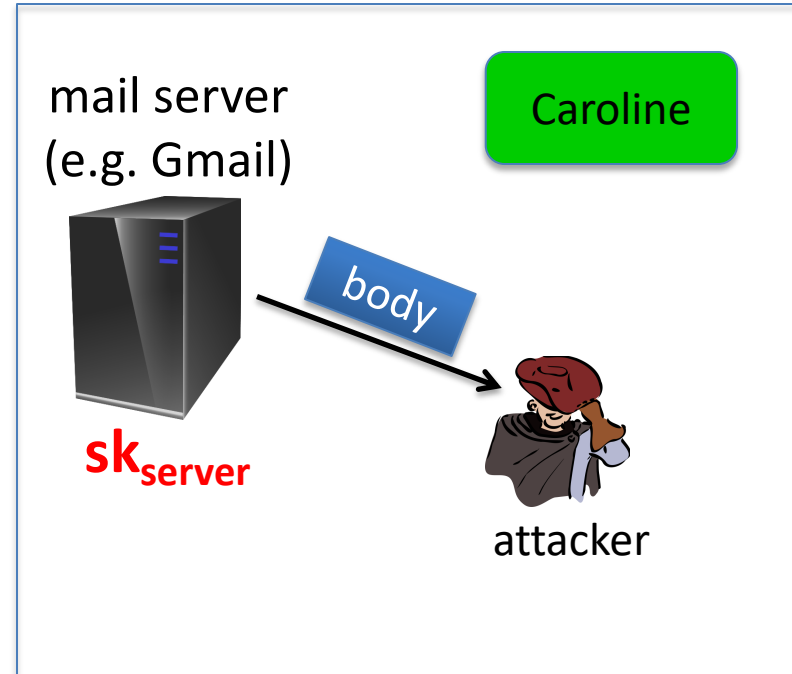
For public key encryption:

- One-time security $\Rightarrow$ many-time security (CPA)

   (follows from the fact that attacker can encrypt by himself)

- Public key encryption **must** be randomized

Security against active attacks
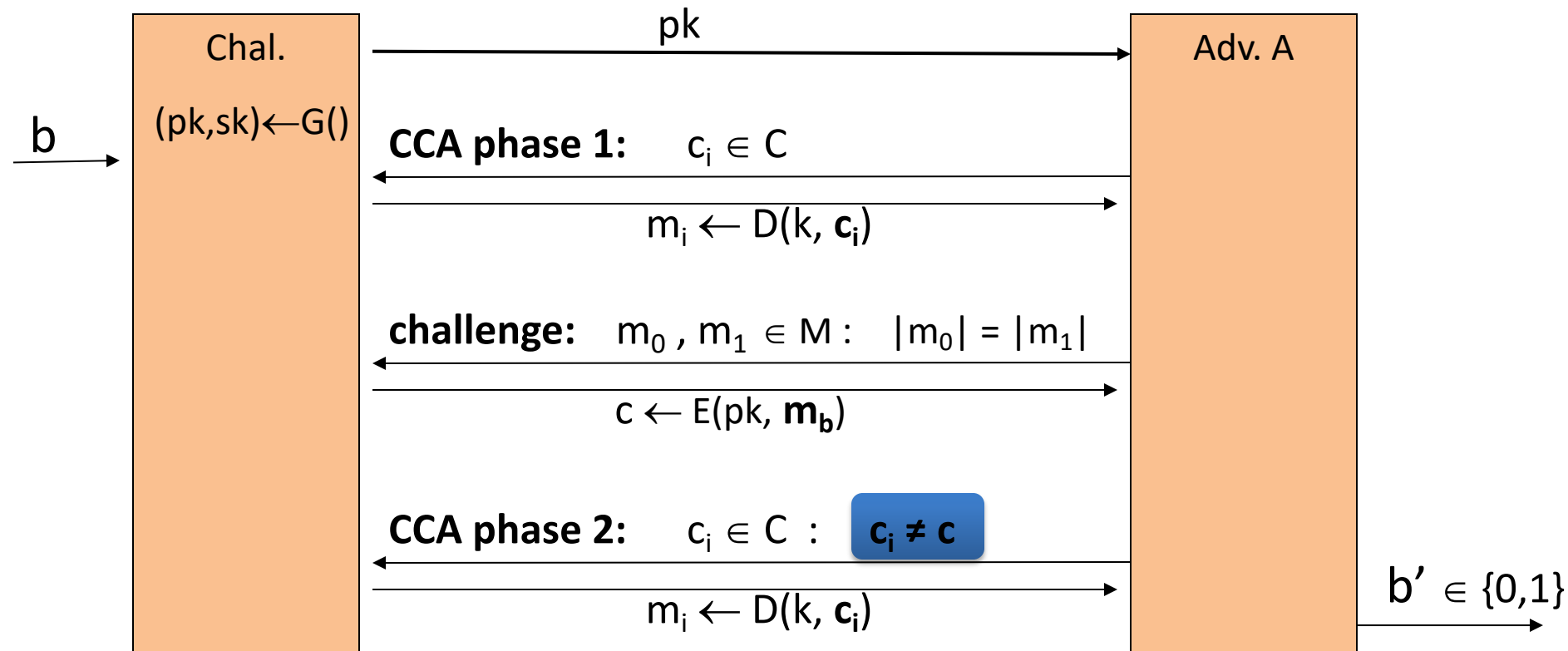
What if attacker can tamper with ciphertext?

Attacker is given decryption of msgs
that start with **"to: attacker"**

(pub-key) Chosen Ciphertext Security: definition

$\mathbb{E} = (G,E,D)$ public-key enc. over (M,C).   For   b=0,1   define EXP(b):

Chal.

(pk,sk)←G()

b →

pk →

Adv. A

CCA phase 1: $c_i \in C$

$m_i \leftarrow D(k, \mathbf{c_i})$

challenge: $m_0 , m_1 \in M$: $|m_0| = |m_1|$

$c \leftarrow E(pk, \mathbf{m_b})$

CCA phase 2: $c_i \in C$:   **$c_i \neq c$**

$m_i \leftarrow D(k, \mathbf{c_i})$

$b' \in \{0,1\}$ →

Chosen ciphertext security: definition

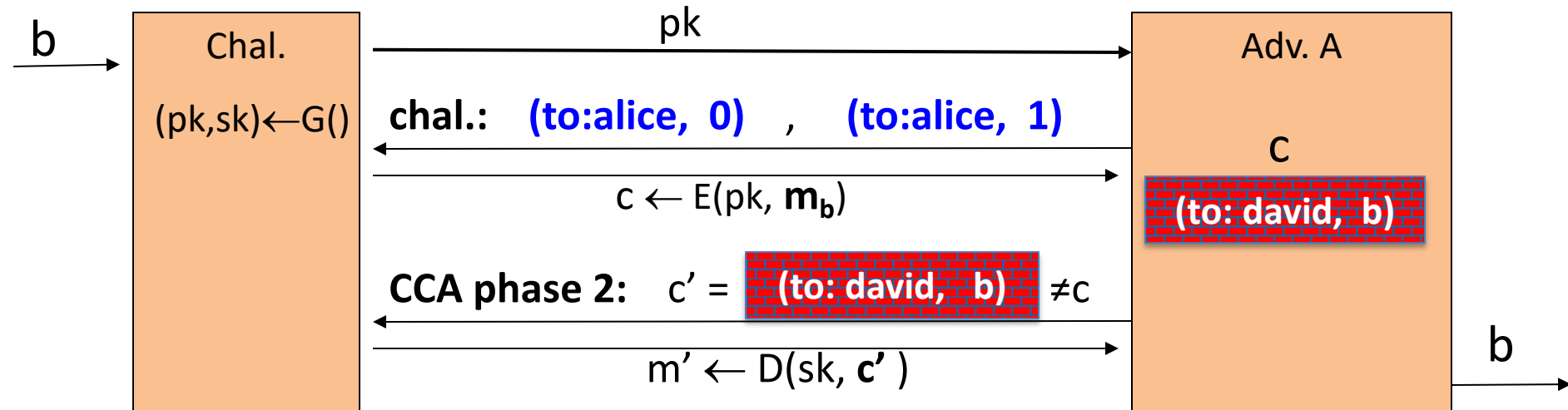**Def**: $\mathbb{E}$ is CCA secure (a.k.a IND-CCA) if for all efficient A:

$$\text{Adv}_{CCA} [A,\mathbb{E}] = \Big| \Pr[\text{EXP}(0)=1] - \Pr[\text{EXP}(1)=1] \Big| \text{ is negligible.}$$

Example: Suppose   (to: alice, body) $\longrightarrow$ (to: david, body)

b → Chal.

(pk,sk)←G()

pk →

chal.:   (to:alice, 0)   ,   (to:alice, 1)

c ← E(pk, **m_b**)

CCA phase 2:   c' = (to: david,   b) ≠c

m' ← D(sk, **c'**)

Adv. A

c

(to: david, b)

→ b

Active attacks: symmetric vs. pub-key

Recall: secure symmetric cipher provides **authenticated encryption**

[chosen plaintext security & ciphertext integrity **]**

- Roughly speaking: **attacker cannot create new ciphertexts**
- Implies security against chosen ciphertext attacks

In public-key settings:

- Attacker **can** create new ciphertexts using pk   !!
- So instead: we directly require chosen ciphertext security

This and next module:

constructing CCA secure pub-key systems

End of Segment

Public Key Encryption
from trapdoor permutations

Constructions

Goal: construct chosen-ciphertext secure public-key encryption

Trapdoor functions (TDF)

Def: a trapdoor func. $X \longrightarrow Y$ is a triple of efficient algs. (G, F, F^{-1})

• G():   randomized alg. outputs a key pair   (pk, sk)

• F(pk,·): det. alg. that defines a function $X \longrightarrow Y$

• F^{-1}(sk,·):   defines a function $Y \longrightarrow X$ that inverts   F(pk,·)
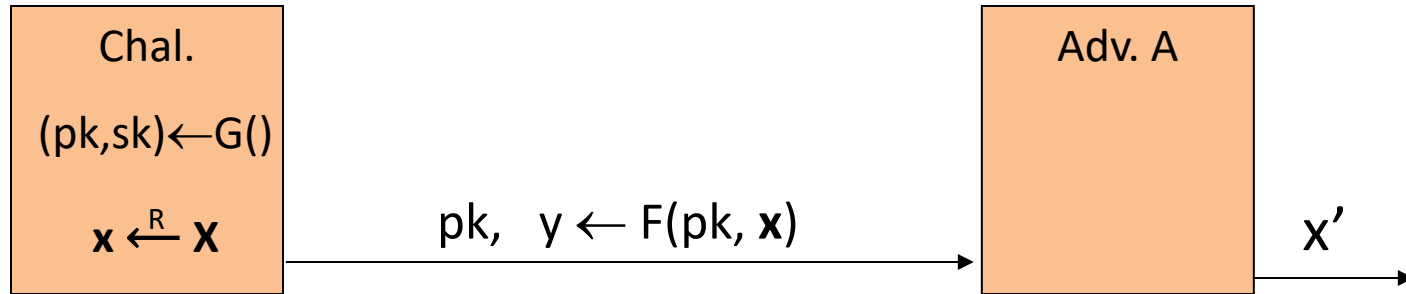
More precisely: $\forall$(pk, sk) output by G

$$\forall x \in X: \quad F^{-1}(sk,\ F(pk, x)\ ) = x$$

Secure Trapdoor Functions (TDFs)

(G, F, F^{-1}) is secure if $F(pk, \cdot)$ is a "one-way" function:

can be evaluated, but cannot be inverted without sk

| Chal. |
| $(pk,sk) \leftarrow G()$ |
| $x \xleftarrow{R} X$ |

pk, $\quad y \leftarrow F(pk, x)$

| Adv. A |

x'

<u>Def</u>: (G, F, F^{-1}) is a secure TDF if for all efficient A:

$$\text{Adv}_{OW}[A,F] = \textbf{\textcolor{red}{Pr[x = x']}} < \text{negligible}$$

Public-key encryption from TDFs

- (G, F, F^{-1}):   secure TDF $X \longrightarrow Y$

- (E_s, D_s) :   symmetric auth. encryption defined over (K,M,C)

- $H: X \longrightarrow K$ a hash function

We construct a pub-key enc. system (G, E, D):

      Key generation G: same as G for TDF

Public-key encryption from TDFs

- (G, F, F^{-1}):   secure TDF $X \longrightarrow Y$

- (E_s, D_s) :   symmetric auth. encryption defined over (K,M,C)

- $H: X \longrightarrow K$ a hash function

<u>E(pk, m)</u> :

$\quad x \xleftarrow{R} X, \qquad y \longleftarrow F(pk, x)$

$\quad k \longleftarrow H(x), \quad c \longleftarrow E_s(k, m)$

$\quad$ output (y, c)

<u>D(sk, (y,c))</u> :

$\quad x \longleftarrow F^{-1}(sk, y),$

$\quad k \longleftarrow H(x), \quad m \longleftarrow D_s(k, c)$

$\quad$ output   m

In pictures:

F(pk, x)	$E_s(H(x),\ m)$

header   body
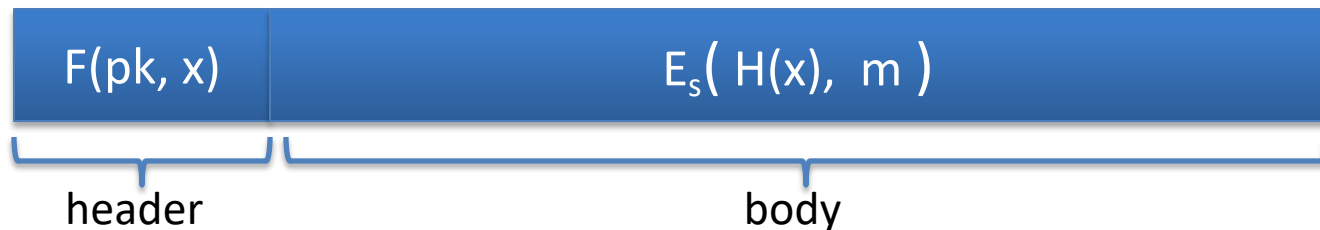
<u>Security Theorem</u>:

If **(G, F, F^{-1})** is a secure TDF,   **(E$_s$, D$_s$)** provides auth. enc.

and   **H:** $X \longrightarrow K$ is a   "random oracle"

then   **(G,E,D)**   is CCA^{ro} secure.

Incorrect use of a Trapdoor Function (TDF)

Never encrypt by applying F directly to plaintext:

E(pk, m) :
     output $c \longleftarrow F(pk, m)$

D(sk, c) :
     output $F^{-1}(sk, c)$

Problems:

- Deterministic: cannot be semantically secure !!
- Many attacks exist   (next segment)

Next step: construct a TDF

End of Segment

Public Key Encryption
from trapdoor permutations

The RSA trapdoor
permutation

Review: trapdoor permutations

Three algorithms: (G, F, F^{-1})

- G: outputs pk, sk.   pk defines a function $F(pk, \cdot): X \rightarrow X$

- F(pk, x):   evaluates the function at x

- F^{-1}(sk, y): inverts the function at y using sk

Secure trapdoor permutation:

   The function $F(pk, \cdot)$ is one-way without the trapdoor sk

Review: arithmetic mod composites

Let $N = p \cdot q$ where p, q are prime

$$Z_N = \{0, 1, 2, \ldots, N-1\} \quad ; \quad (Z_N)^* = \{\text{invertible elements in } Z_N\}$$

<u>Facts:</u> $x \in Z_N$ is invertible $\iff$ $\gcd(x, N) = 1$

    – Number of elements in $(Z_N)^*$ is $\varphi(N) = (p-1)(q-1) = N-p-q+1$

<u>Euler's thm:</u> $\boxed{\forall\, x \in (Z_N)^* \ : \ x^{\varphi(N)} = 1}$

The RSA trapdoor permutation

First published: Scientific American, Aug. 1977.

Very widely used:

— SSL/TLS: certificates and key-exchange

— Secure e-mail and file systems

    … many others

The RSA trapdoor permutation

G(): choose random primes   p,q $\approx$ 1024 bits.   Set **N=pq**.

   choose integers **e , d** s.t. **e·d = 1 (mod φ(N))**

   output   pk = (N, e)   ,   sk = (N, d)

F(pk, x): $\mathbb{Z}_N^* \to \mathbb{Z}_N^*$;   **RSA(x) = x^e**   (in Z$_N$)

F^{-1}(sk, y) = y^d ;   y^d = **RSA(x)^d** = x^{ed} = x$^{k\varphi(N)+1}$ = $\left(x^{\varphi(N)}\right)^k$ · **x** = x

The RSA assumption

RSA assumption: RSA is one-way permutation

For all efficient algs. A:

$$\Pr\Big[\ A(N,e,y) = y^{1/e}\ \Big] < \text{negligible}$$

where $p,q \xleftarrow{R}$ n-bit primes, $N \leftarrow pq$, $y \xleftarrow{R} Z_N^*$

Review: RSA pub-key encryption   (ISO std)

(E_s, D_s):   symmetric enc. scheme providing auth. encryption.

H: $Z_N \rightarrow K$ where K is key space of (E_s, D_s)

- **G**():   generate RSA params: pk = (N,e),   sk = (N,d)

- **E**(pk, m):   (1) choose random x in Z_N

    (2) y $\leftarrow$ RSA(x) = x^e ,   k $\leftarrow$ H(x)

    (3) output   (y , E_s(k,m))

- **D**(sk, (y, c)):   output D_s(H(RSA^{-1} (y)) , c)

Textbook RSA is insecure

Textbook RSA encryption:

- public key: **(N,e)**   Encrypt: $\mathbf{c} \longleftarrow \mathbf{m}^{\mathbf{e}}$ (in Z_N)
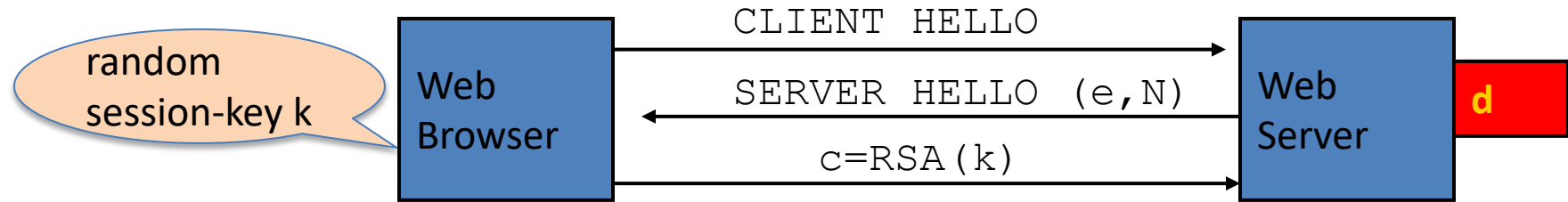- secret key: **(N,d)**   Decrypt: $\mathbf{c}^{\mathbf{d}} \longrightarrow \mathbf{m}$

Insecure cryptosystem !!

- Is not semantically secure and many attacks exist

⇒   The RSA trapdoor permutation is not an encryption scheme !

A simple attack on textbook RSA

Suppose k is 64 bits: $k \in \{0,\ldots,2^{64}\}$.   Eve sees: $c = k^e$ in Z_N

If $\mathbf{k = k_1 \cdot k_2}$ where $k_1, k_2 < 2^{34}$ (prob. $\approx 20\%$)   then $\mathbf{c/k_1^{\,e} = k_2^{\,e}}$ in Z_N

Step 1: build table: $c/1^e, c/2^e, c/3^e, \ldots, c/2^{34e}$. time: 2^{34}

Step 2: for $k_2 = 0,\ldots, 2^{34}$ test if $k_2^{\,e}$ is in table.   time: 2^{34}

Output matching (k_1, k_2).   Total attack time: $\approx 2^{40} \ll 2^{64}$
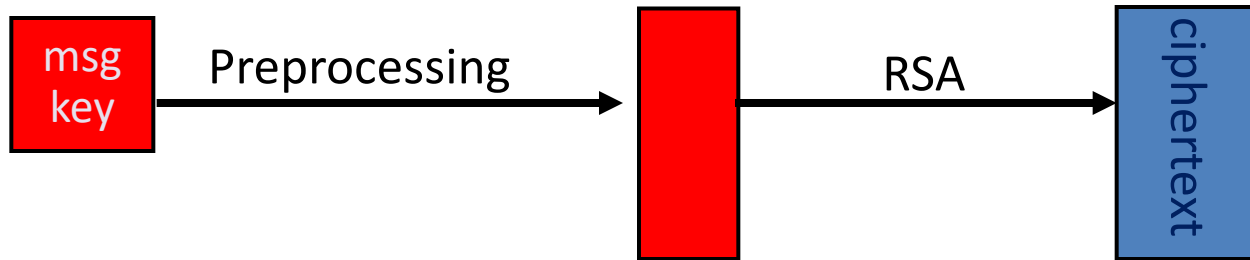
End of Segment

Public Key Encryption from trapdoor permutations

PKCS 1

RSA encryption in practice

Never use textbook RSA.

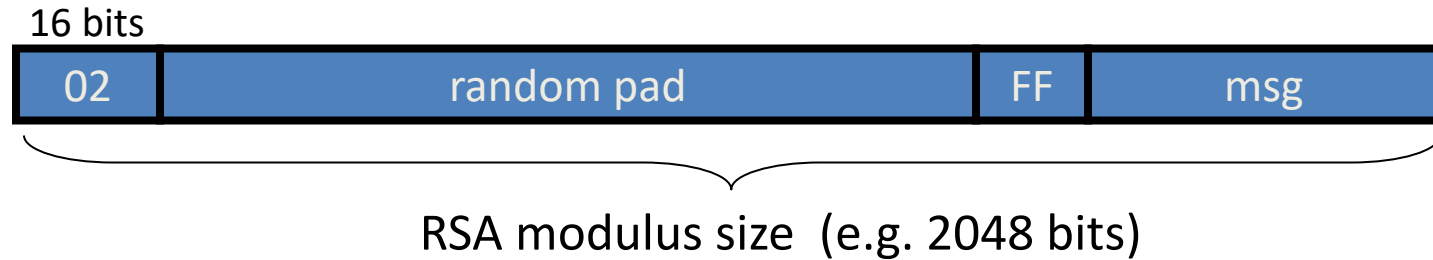RSA in practice   (since ISO standard is not often used) :

Main questions:
- How should the preprocessing be done?
- Can we argue about security of resulting system?
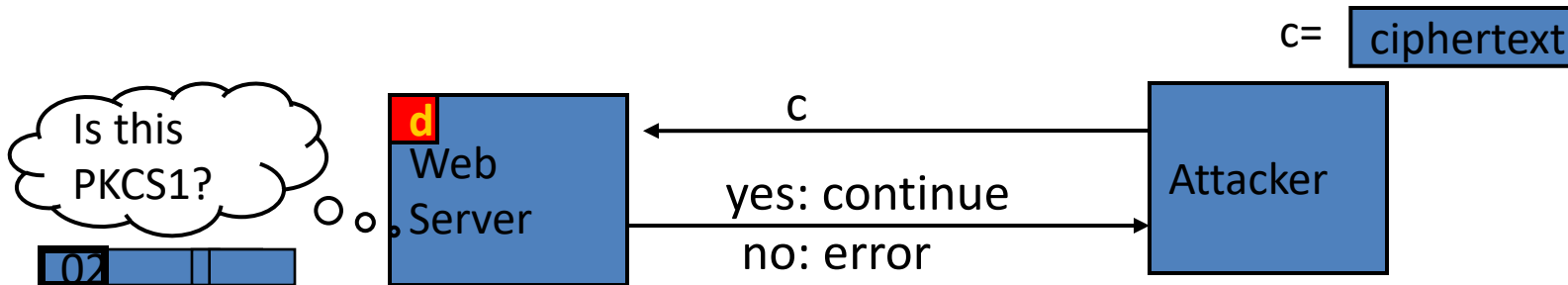
PKCS1 v1.5

PKCS1 mode 2:   (encryption)

16 bits

| 02 | random pad | FF | msg |

RSA modulus size (e.g. 2048 bits)

- Resulting value is RSA encrypted

- Widely deployed, e.g. in HTTPS

Attack on PKCS1 v1.5   (Bleichenbacher 1998)

PKCS1 used in HTTPS:

$\Rightarrow$ attacker can test if 16 MSBs of plaintext = '02'
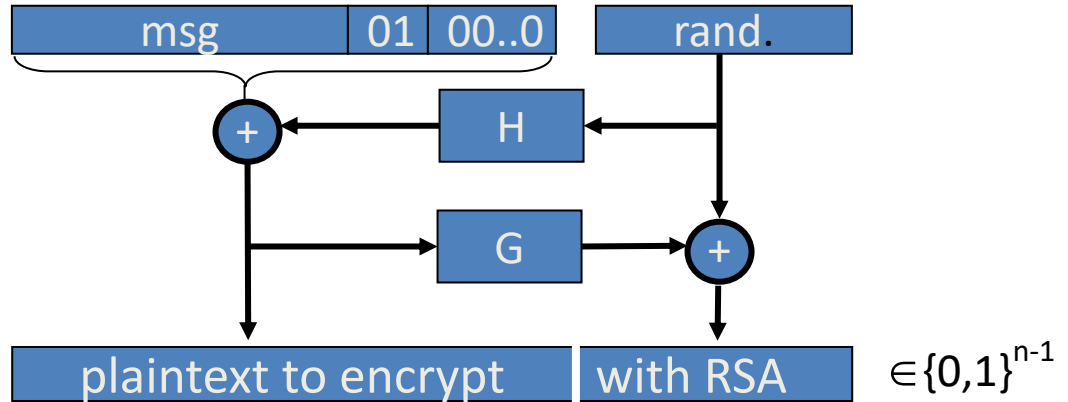
Chosen-ciphertext attack: to decrypt a given ciphertext c do:

– Choose $r \in Z_N$.   Compute $c' \longleftarrow r^e \cdot c \; = \left(r \cdot \text{PKCS1}(m)\right)^e$

– Send c' to web server and use response

Dan Boneh

PKCS1 v2.0: OAEP

New preprocessing function: OAEP   [BR94]

check pad
on decryption.
reject CT if invalid.

| msg | 01 | 00..0 | rand. |

H

+

G

+

| plaintext to encrypt | with RSA | $\in \{0,1\}^{n-1}$

Thm [FOPS'01] : RSA is a trap-door permutation $\Rightarrow$
        RSA-OAEP is CCA secure when H,G are *random oracles*

in practice: use SHA-256 for H and G

End of Segment

Public Key Encryption
from trapdoor permutations

RSA in practice

RSA With Low public exponent

To speed up RSA encryption use a small e: $\quad$ c = m^e (mod N)

- Minimum value: **e=3** $\quad$ (gcd(e, φ(N)) = 1)

- Recommended value: **e=65537=2^{16}+1**

$\qquad$ Encryption: 17 multiplications

Asymmetry of RSA: fast enc. / slow dec.

$\quad$ – ElGamal (next module): approx. same time for both.

Key lengths

Security of public key system should be comparable to security of symmetric cipher:

<table>
<tr><td>Cipher key-size</td><td>RSA
Modulus size</td></tr>
<tr><td>80 bits</td><td>1024 bits</td></tr>
<tr><td>128 bits</td><td>3072 bits</td></tr>
<tr><td>256 bits (AES)</td><td><u>**15360**</u> bits</td></tr>
</table>

Implementation attacks

Timing attack: [Kocher et al. 1997] ,   [BB'04]

The time it takes to compute $c^d \pmod{N}$ can expose d

Power attack: [Kocher et al. 1999)

The power consumption of a smartcard while
it is computing $c^d \pmod{N}$ can expose d.

Faults attack: [BDL'97]

A computer error during $c^d \pmod{N}$ can expose d.

A common defense: check output.   10% slowdown.

An Example Fault Attack on RSA (CRT)

A common implementation of RSA decryption: $x = c^d$ in Z_N

decrypt mod p: $x_p = c^d$ in Z_p

decrypt mod q: $x_q = c^d$ in Z_q

combine to get $x = c^d$ in Z_N

Suppose error occurs when computing x_q,   but no error in x_p

Then: output is x' where $x' = c^d$ in Z_p but $x' \neq c^d$ in Z_q

$\Rightarrow$ $(x')^e = c$ in Z_p but $(x')^e \neq c$ in Z_q $\Rightarrow$ $\gcd\big((x')^e - c , N \big) = p$

RSA Key Generation Trouble [Heninger et al./Lenstra et al.]

OpenSSL RSA key generation (abstract):

```
prng.seed(seed)
p = prng.generate_random_prime()
prng.add_randomness(bits)
q = prng.generate_random_prime()
N = p*q
```

Suppose poor entropy at startup:

- Same p will be generated by multiple devices, but different q

- N_1 , N_2 : RSA keys from different devices $\Rightarrow$ gcd(N_1,N_2) = p

RSA Key Generation Trouble [Heninger et al./Lenstra et al.]

Experiment: factors 0.4% of public HTTPS keys !!

Lesson:

- Make sure random number generator is properly
  seeded when generating keys

Further reading

- Why chosen ciphertext security matters, V. Shoup, 1998

- Twenty years of attacks on the RSA cryptosystem,
  D. Boneh, Notices of the AMS, 1999

- OAEP reconsidered, V. Shoup, Crypto 2001

- Key lengths, A. Lenstra, 2004

End of Segment