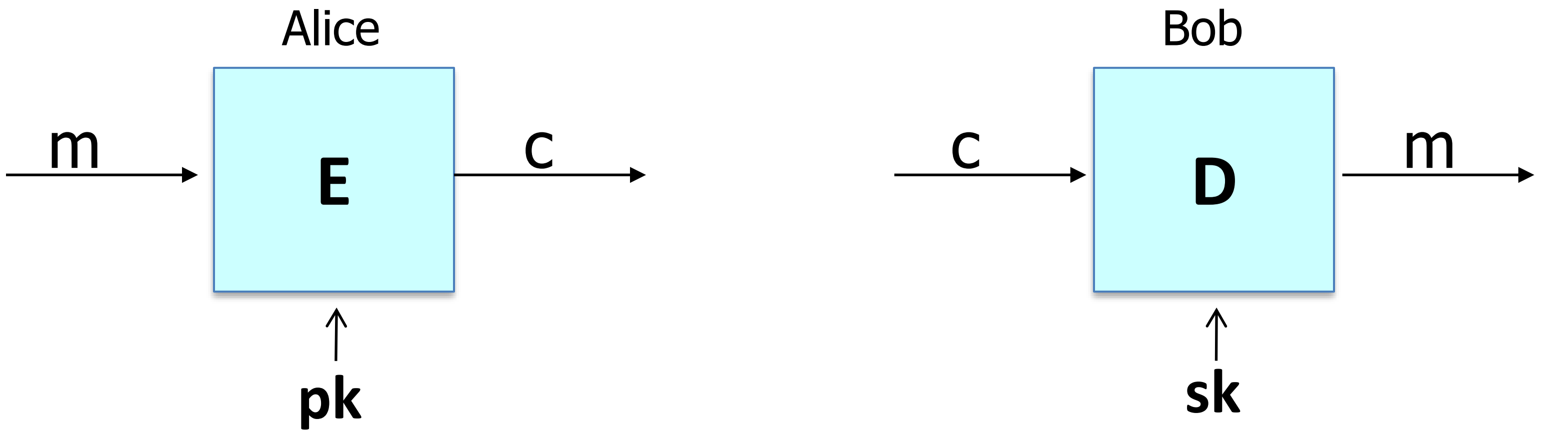
Public Key Encryption
from trapdoor permutations

# Public key encryption: definitions and security

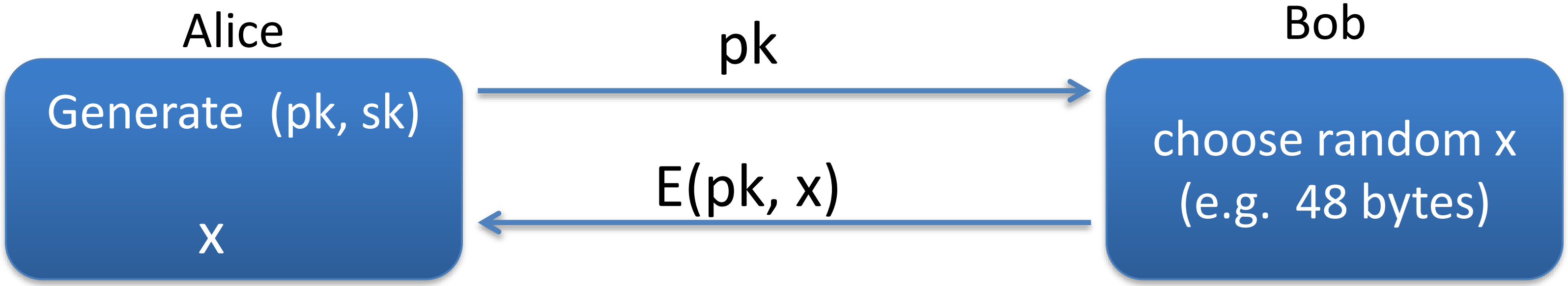
# Public key encryption

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



# Applications

**Session setup** (for now, only eavesdropping security)




**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)

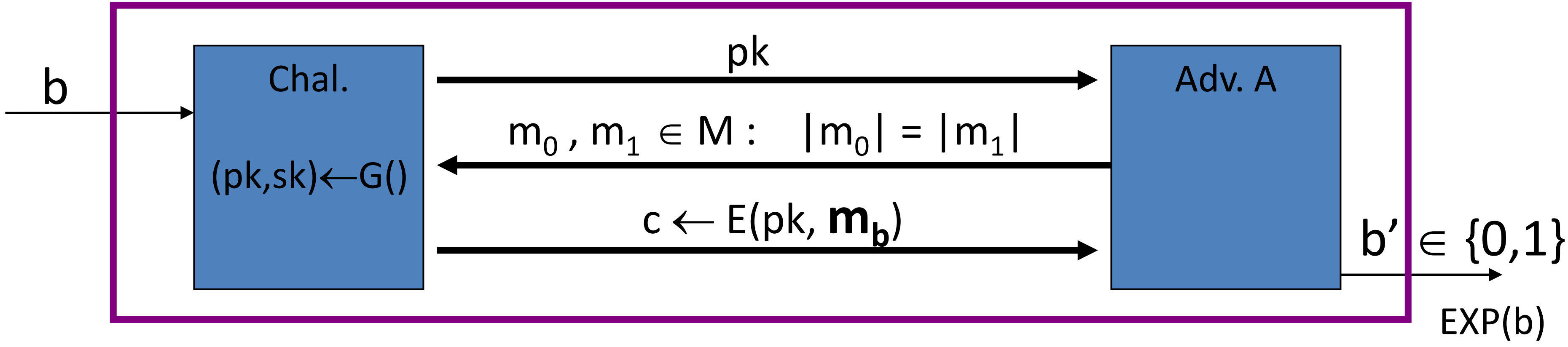
- G(): randomized alg. outputs a key pair (pk, sk)
- E(pk, m): randomized alg. that takes $m \in M$ and outputs $c \in C$
- D(sk,c): det. alg. that takes $c \in C$ and outputs $m \in M$ or $\perp$

Consistency: $\forall$(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:

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

# 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:

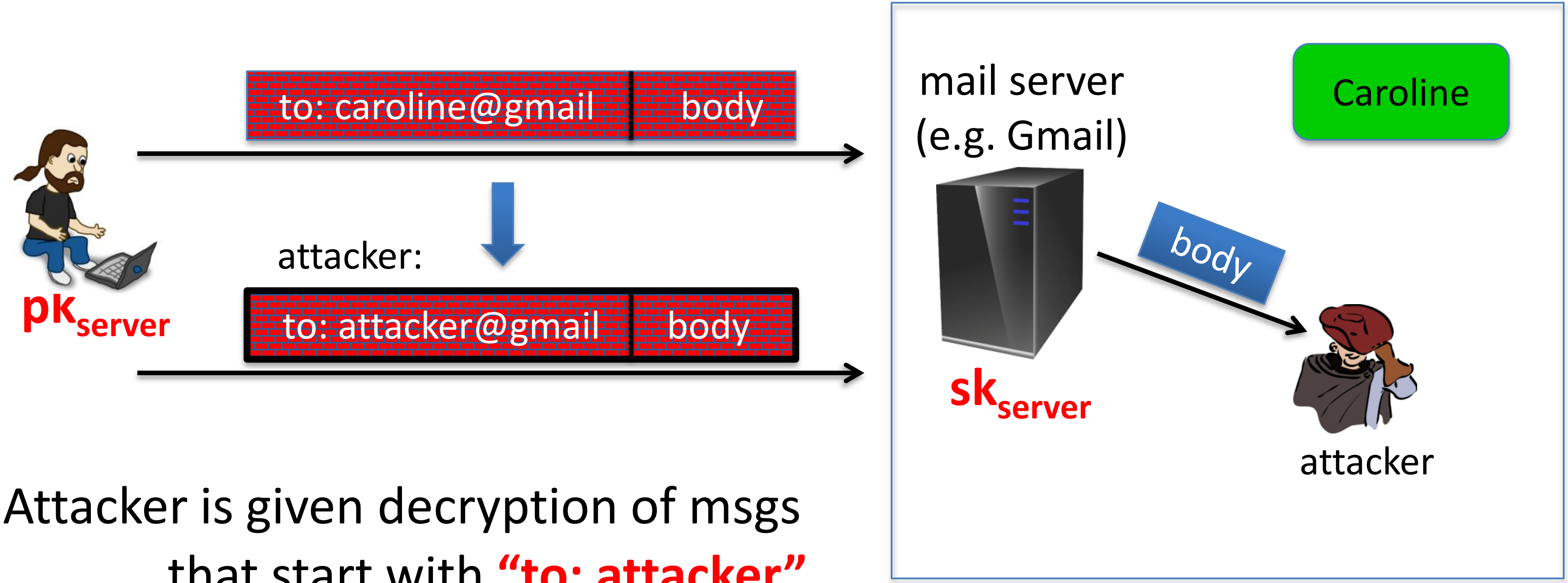
- One-time security $\Rightarrow$ many-time security (CPA)

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

- Public key encryption **must** be randomized

# Security against active attacks

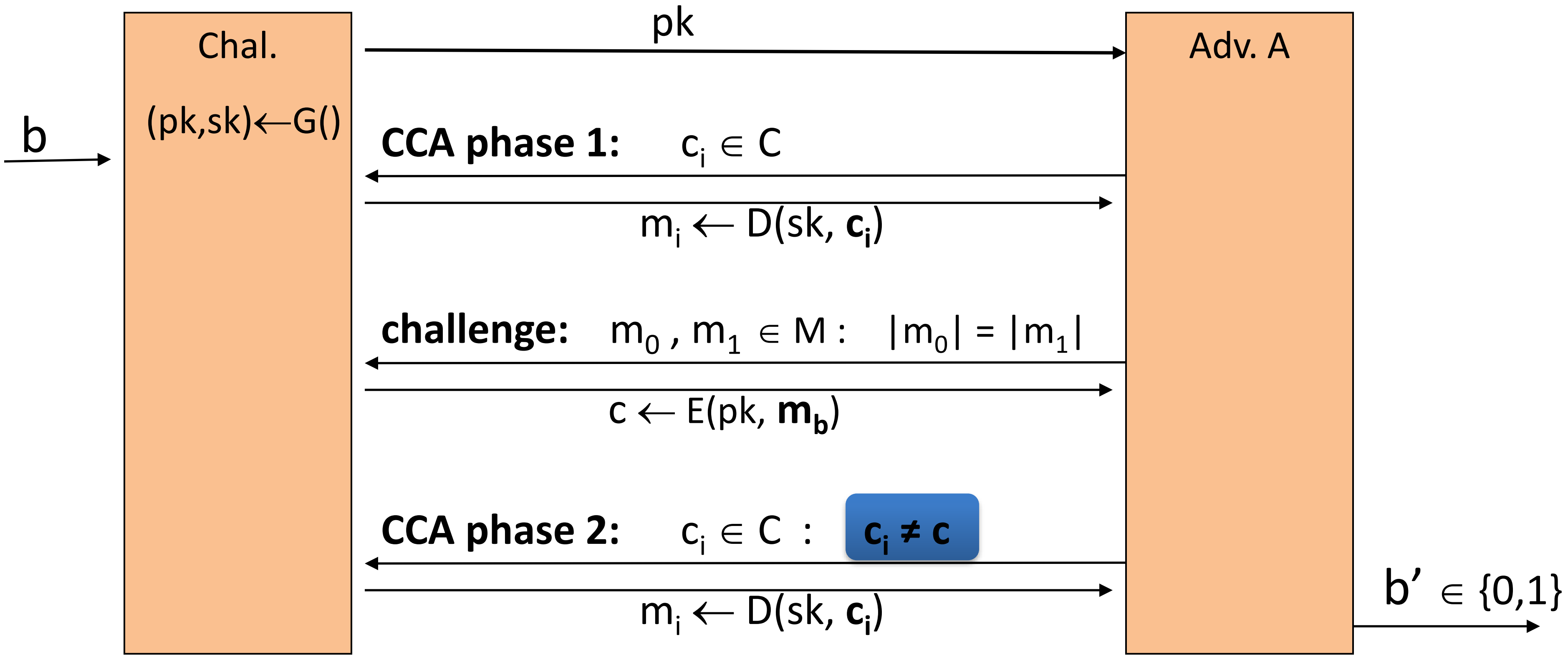
What if attacker can tamper with ciphertext?




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):

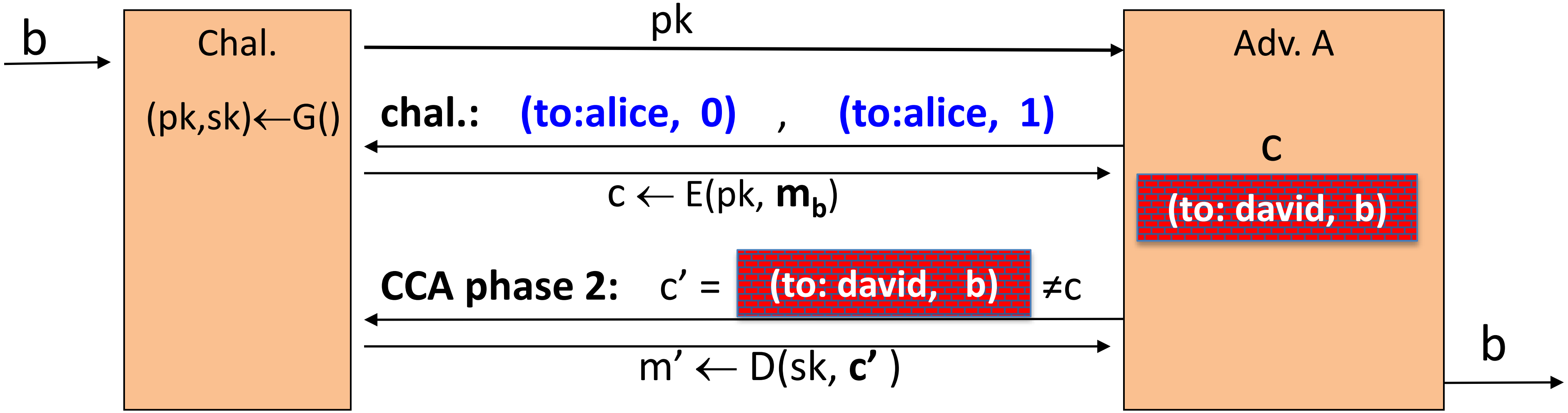


# Chosen ciphertext security: definition

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

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

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



# 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


# 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 $(pk, sk)$
- $F(pk,\cdot)$: det. alg. that defines a function $X \longrightarrow Y$
- $F^{-1}(sk,\cdot)$: defines a function $Y \longrightarrow X$ that inverts $F(pk,\cdot)$

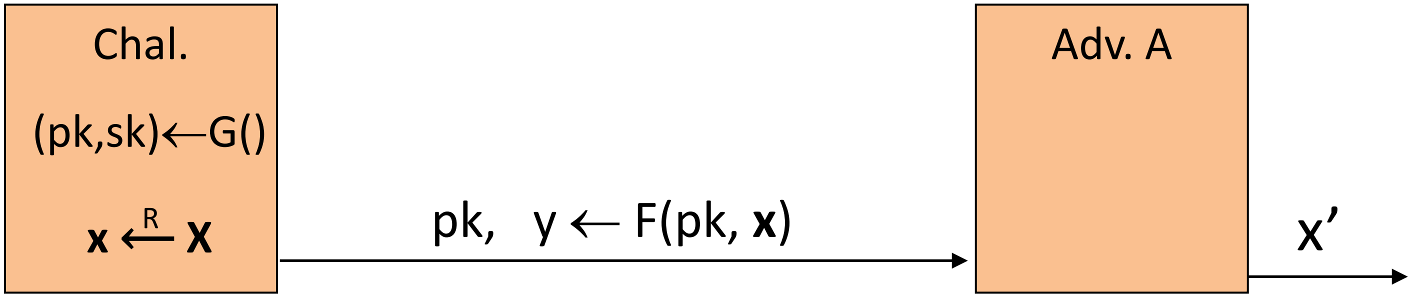
More precisely: $\forall (pk, sk)$ output by $G$

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

# 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




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

$$\mathrm{Adv}_{OW}[A,F] = \mathbf{Pr[\; x = x' \;]} < \text{negligible}$$

# 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> **:**

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

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

output $(y, c)$

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

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

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

output $m$

In pictures:

$$\underbrace{\boxed{F(pk,\ x)}}_{\text{header}}\ \underbrace{\boxed{E_s(\ H(x),\ \ m\ )}}_{\text{body}}$$

**<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:

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

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

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

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

Problems:

- Deterministic: cannot be semantically secure !!

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 $F(pk, \cdot): X \to X$
- 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)

# 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 ≈1024 bits (300 digits). 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)** = $y^d$ ; $y^d$ = **$\text{RSA}(x)^d$** = $x^{ed}$ = x

# RSA summarized

- Choose random primes $p$ and $q$ (keep secret, delete after key generation)
- Calculate $N = p.q$ (public)
- Calculate $\varphi(N) = (p-1) * (q-1)$ (keep secret, delete after key generation)
- Choose $e: 1 < e < \varphi(N)$ (public, integer and coprime to φ(N))
- Calculate $d = e^{-1}\, mod\, \varphi(N)$ (keep secret)

- Public key: $K_p = (N, e)$ Private key: $K_s = (N, d)$
- $E\left(K_p, x\right) = x^e\, mod\, N = c$ $D(K_s, c) = c^d\, mod\, N = x$

$$(x^e)^d\, mod\, N = x\, mod\, N$$

# The RSA assumption

RSA assumption:     RSA is one-way permutation

For all efficient algs. A:

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

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 $D_s(\ \overbrace{H(\underbrace{RSA^{-1}(y)}_{x})}^{k}\ ,\ c) = m$

# Textbook RSA is insecure

Textbook RSA encryption:

– public key: **(N,e)** Encrypt: $\mathbf{c \longleftarrow m^{e}}$ (in $Z_N$)

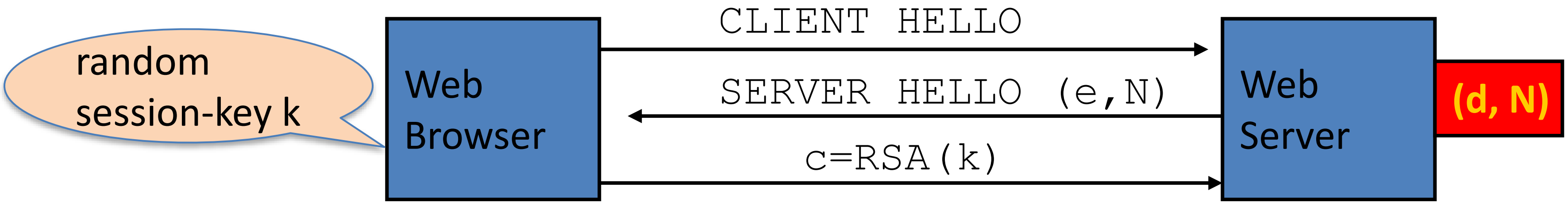
– secret key: **(N,d)** Decrypt: $\mathbf{c^{d} \longrightarrow m}$

Insecure cryptosystem (deterministic enc.)!!

– 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: $k \in \{0,\dots,2^{64}\}$. Eve sees: $c = k^e$ in $Z_N$

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

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

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

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

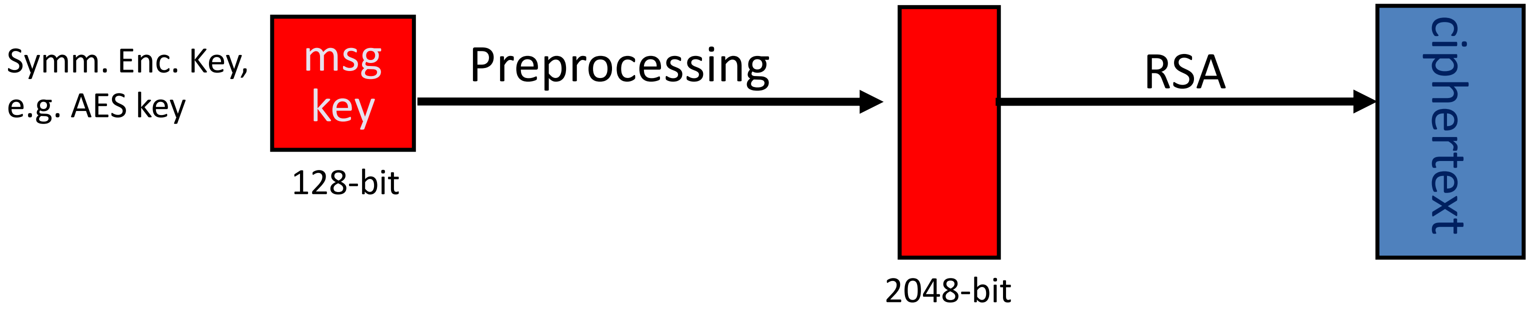

# Public Key Encryption from trapdoor permutations

## PKCS 1

# RSA encryption in practice

Never use textbook RSA.

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



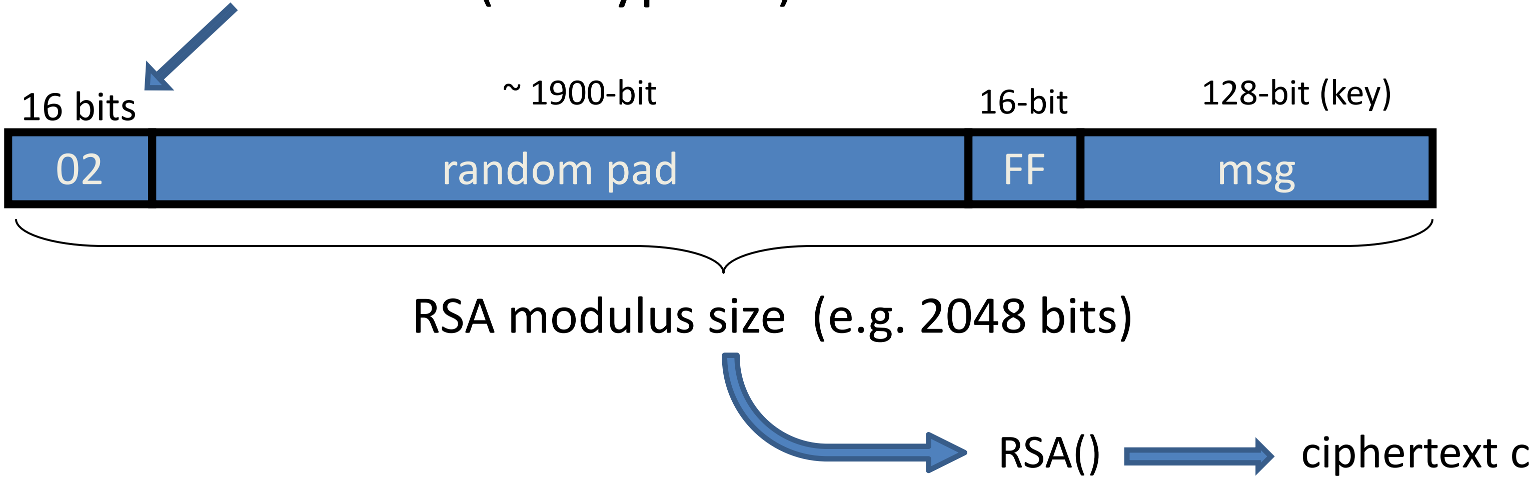

Main questions:

- How should the preprocessing be done?
- Can we argue about security of resulting system?

# PKCS1 v1.5

Public Key Cryptography Standards

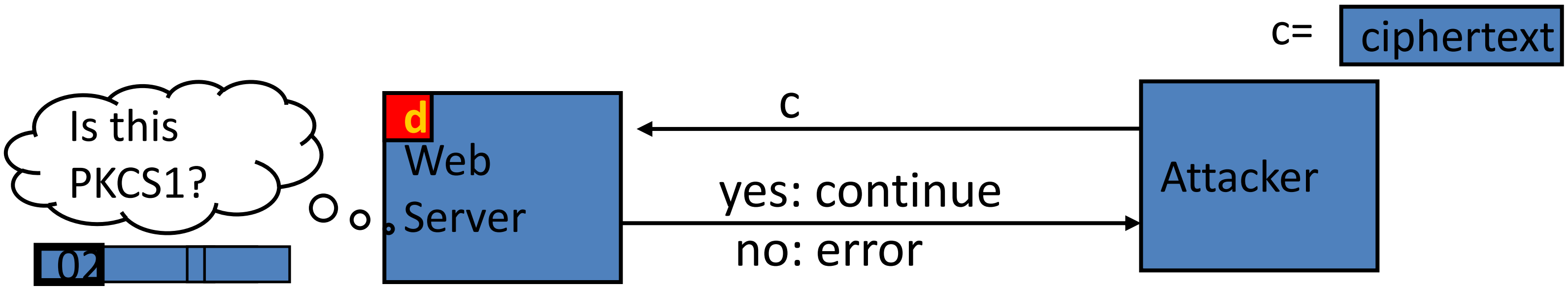
PKCS1 mode 2: (encryption)




- Resulting value is RSA encrypted
- Widely deployed, e.g. in HTTPS

# Attack on PKCS1 v1.5 (Bleichenbacher 1998)

PKCS1 used in HTTPS:

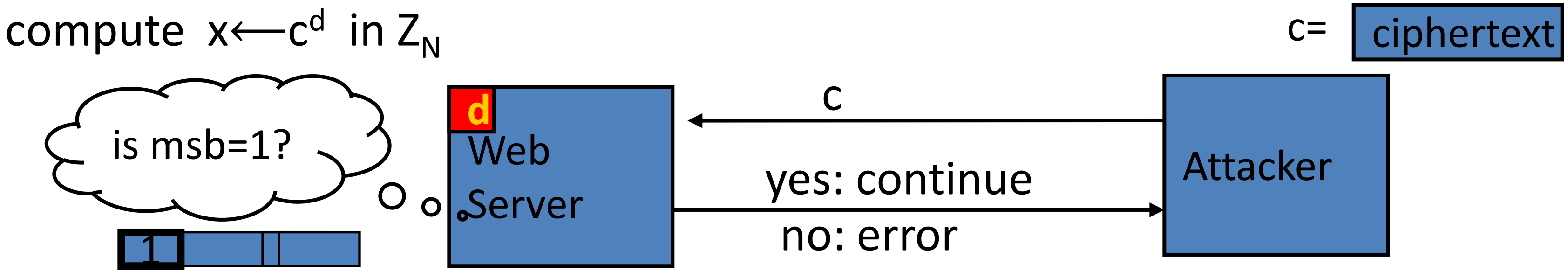



$\Rightarrow$ attacker can test if 16 MSBs of plaintext = '02'

Chosen-ciphertext attack: to decrypt a given ciphertext c do:

- Choose $r \in Z_N$. Compute $c' \leftarrow r^e \cdot c \ = (r \cdot \mathrm{PKCS1}(m))^e$
- Send c' to web server and use response
- Repeat by sending ciphertext queries as many times as needed to recover C

# Baby Bleichenbacher




Suppose N is $N = 2^n$ (an invalid RSA modulus). Then:

- Sending c reveals msb( x ) x=PKCS1(m)
- Sending $\mathbf{2^e \cdot c = (2x)^e}$ in $Z_N$ reveals msb(2x mod N) = $\mathbf{msb_2(x)}$
- Sending $\mathbf{4^e \cdot c = (4x)^e}$ in $Z_N$ reveals msb(4x mod N) = $\mathbf{msb_3(x)}$
- … and so on to reveal all of x

# HTTPS Defense (RFC 5246)

*Attacks discovered by Bleichenbacher and Klima et al. … can be avoided by treating incorrectly formatted message blocks … in a manner indistinguishable from correctly formatted RSA blocks. In other words:*

1. *Generate a string* ***R*** *of 46 random bytes*
2. *Decrypt the message to recover the plaintext M (session key)*
3. *If the PKCS#1 padding is not correct (*$\neq 02$*)*

   *pre_master_secret = **R***

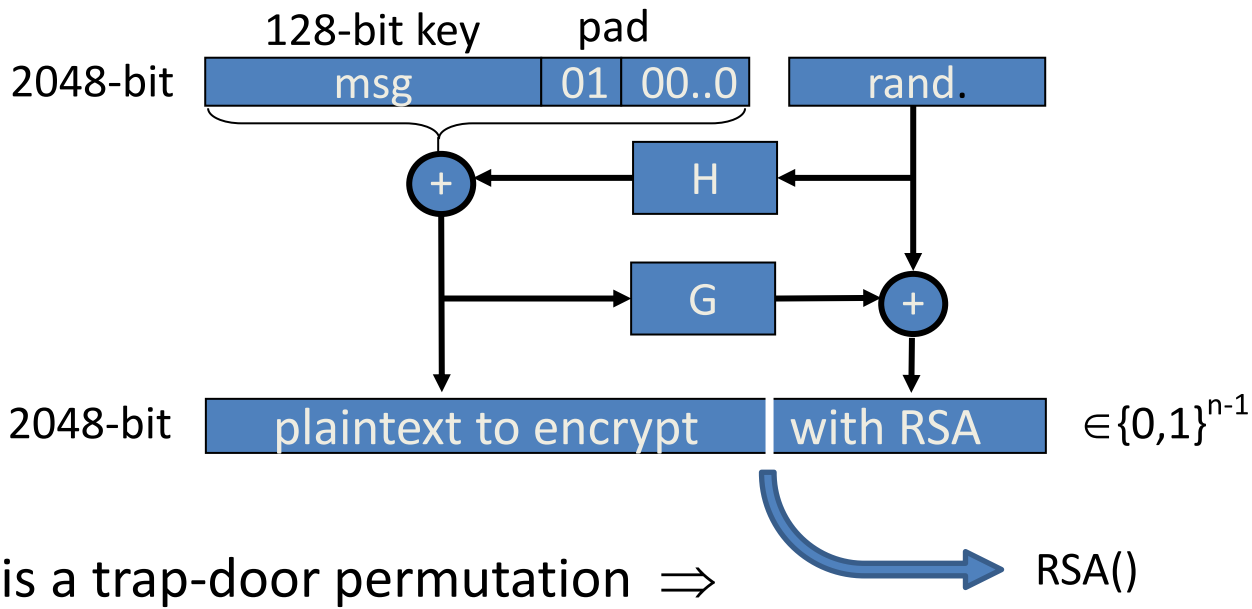
- *Session will terminate (since client and server ended up with different session keys)*

# PKCS1 v2.0: OAEP

Optimal Asymmetric Encryption Padding

New preprocessing function: OAEP [BR94]

check pad
on decryption.
reject CT if invalid.




**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


## Public Key Encryption from trapdoor permutations

# Is RSA a one-way function?

# Is RSA a one-way permutation?

To invert the RSA one-way func. (without d) attacker must compute:

$x$ from $c = x^e \pmod{N}$.

How hard is computing e'th roots modulo N ??

Best known algorithm:

- Step 1: factor N (hard)
- Step 2: compute e'th roots modulo p and q (easy)
  - Given both e'th roots, it's easy to combine them together, using the Chinese remainder theorem to recover the e'th root modulo N.

# Shortcuts?

Must one factor N in order to compute e'th roots?

To prove no shortcut exists show a reduction:

- Efficient algorithm for e'th roots mod N

  obtains $\Rightarrow$ efficient algorithm for factoring N.

- Oldest problem in public key cryptography (and still open).

Some (weak) evidence no reduction exists: (BV'98)

- "Algebraic" reduction $\Rightarrow$ factoring is easy.


# Public Key Encryption from trapdoor permutations

## RSA in practice

# RSA With Low public exponent

To speed up RSA encryption use a small $e$: $c = m^e \pmod{N}$

- Minimum value: **$e=3$** $(\gcd(e, \varphi(N)) = 1)$
- Recommended value: **$e=65537=2^{16}+1$**

Encryption: 17 multiplications (square 16 times, then multiply 1 time)

<u>Asymmetry of RSA:</u> fast enc. / slow dec.

# Key lengths

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

| Cipher key-size | RSA Modulus size |
|---|---|
| 80 bits | 1024 bits |
| 128 bits | 3072 bits |
| 256 bits (AES) | **15360** bits |

# 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.