



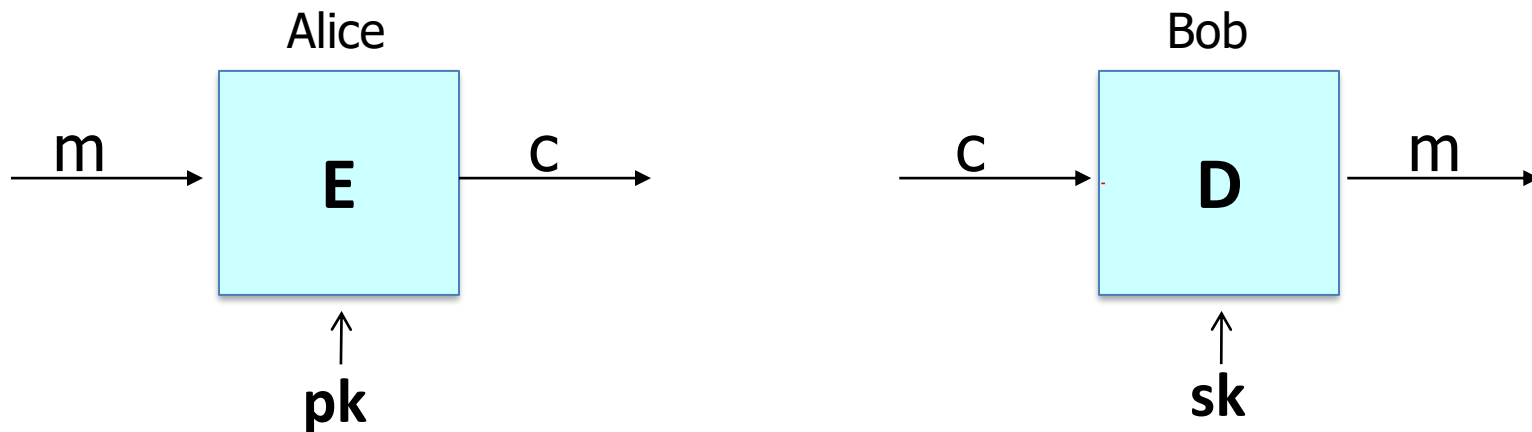
# Public Key Encryption from trapdoor permutations

## Public key encryption: definitions and security

对称	对称加密	MKE
公钥	PKE	sig

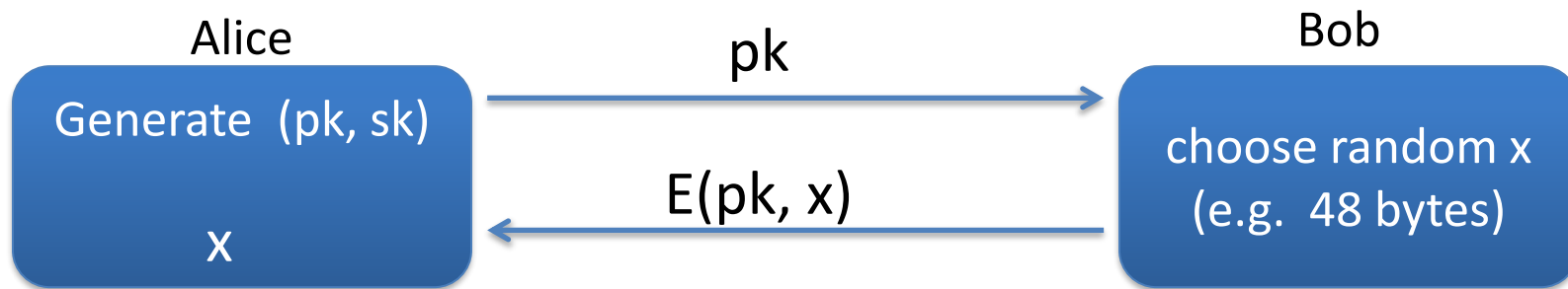
# Public key encryption

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

# Public key encryption

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

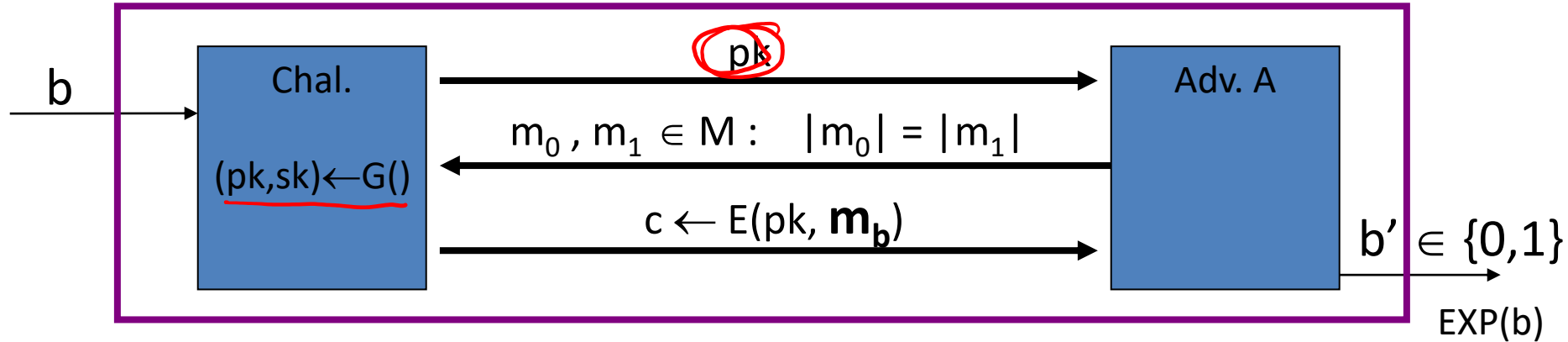
- $G()$ : randomized alg. outputs a key pair  $(\underline{pk}, \underline{sk})$
- $E(pk, m)$ : randomized alg. that takes  $m \in M$  and outputs  $c \in C$
- $D(\underline{sk}, c)$ : det. alg. that takes  $c \in C$  and outputs  $m \in M$  or  $\perp$

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

$$\underline{\forall m \in M}: D(\underline{sk}, \underline{E(pk, m)}) = m$$

# Security: eavesdropping

For  $b=0,1$  define experiments  $\text{EXP}(0)$  and  $\text{EXP}(1)$  as:

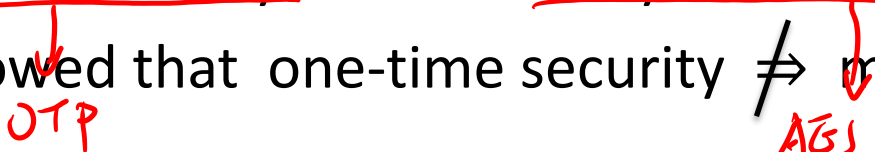


Def:  $\mathbb{E} = (G, E, D)$  is sem. secure (a.k.a IND-CPA) if for all efficient A:

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

# 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  
  
OTP AGS

For public key encryption:

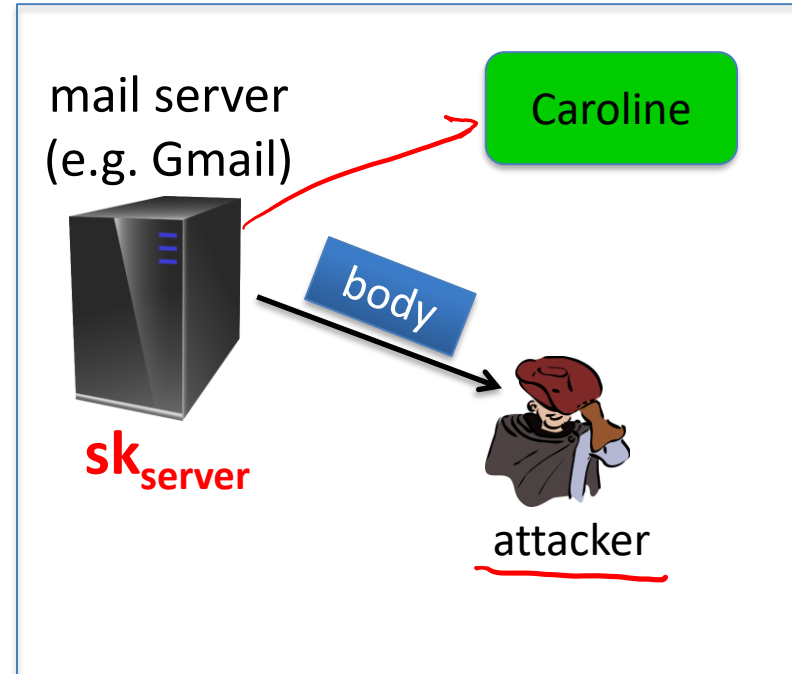
- One-time security  $\Rightarrow$  many-time security (CPA)  
(follows from the fact that attacker can encrypt by himself)
- Public key encryption **must** be randomized

# Security against active attacks

What if attacker can tamper with ciphertext?

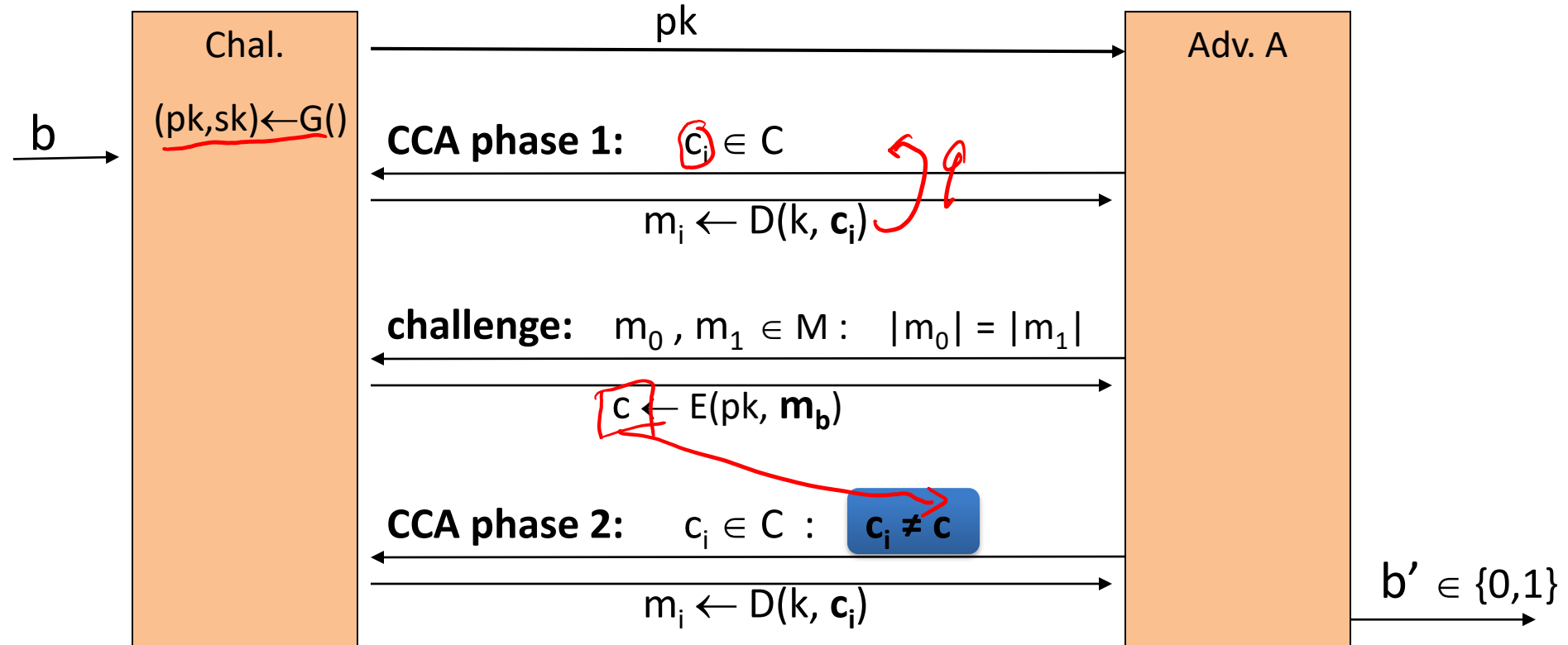


Attacker is given decryption of msgs  
that start with **“to: attacker”**



# (pub-key) Chosen Ciphertext Security: definition

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





# Chosen ciphertext security: definition

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

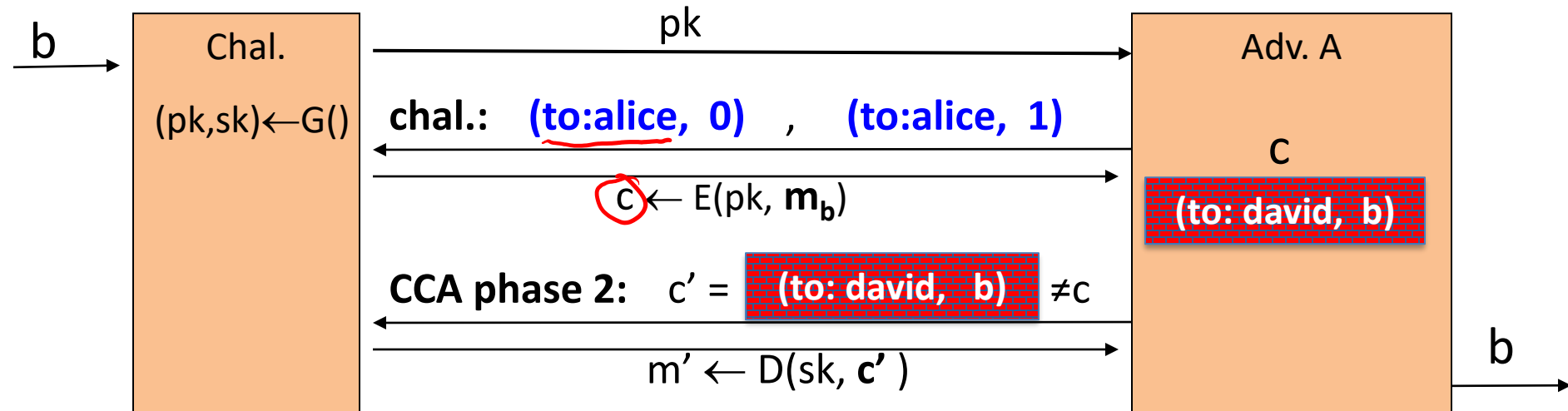
$$\text{Adv}_{\text{CCA}}[A, \mathbb{E}] = \left| \Pr[\text{EXP}(0)=1] - \Pr[\text{EXP}(1)=1] \right| \text{ is negligible.}$$

Example: Suppose

(to: alice, body)

→

(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



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

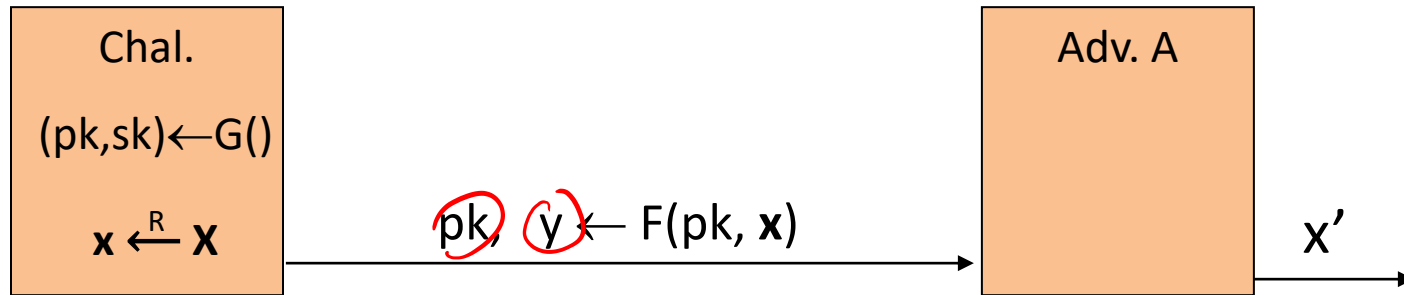
More precisely:  $\forall (pk, sk)$  output by  $G$

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

# 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  $sk$



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

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

# Public-key encryption from TDFs

- $(G, F, F^{-1})$ : secure TDF  $X \rightarrow Y$
- $(E_s, D_s)$ : symmetric auth. encryption defined over  $(K, M, C)$
- $H: X \rightarrow K$  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  $\overset{\text{LSD}}{\textcircled{X}} \rightarrow Y$
- $(E_s, D_s)$ : symmetric auth. encryption defined over  $(K, M, C)$
- $H: X \rightarrow K$  a hash function  ~~$\star$~~

$E(pk, m)$  :

$\textcircled{x} \xleftarrow{R} X, \quad \underline{y} \leftarrow F(pk, x)$

$\textcircled{k} \leftarrow H(x), \quad \underline{c \leftarrow E_s(\textcircled{k}, m)}$

output  $(y, c)$

$D(sk, (y, c))$  :

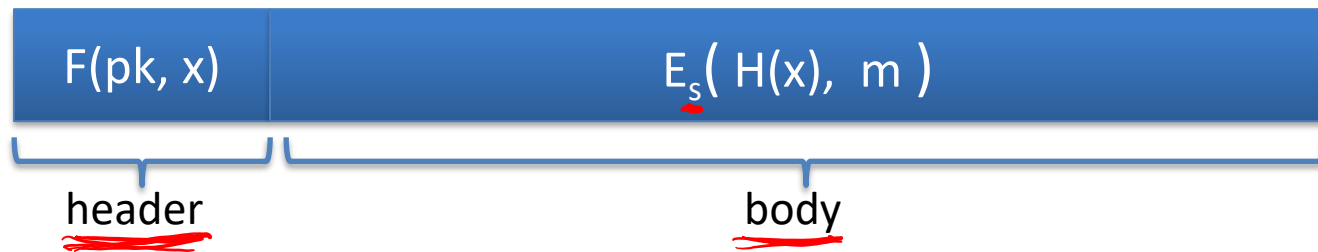
$x \leftarrow F^{-1}(sk, y),$

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

output  $m$



In pictures:



## Security Theorem:

If  $(G, F, F^{-1})$  is a secure TDF,  $(E_s, D_s)$  provides auth. enc.  
and  $H: X \rightarrow K$  is a “random oracle”  
then  $(G, E, D)$  is  $CCA^{ro}$  secure.

# Incorrect use of a Trapdoor Function (TDF)

$\hookrightarrow$  rsa

**Never** encrypt by applying  $F$  directly to plaintext:

$E(pk, m)$  :

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

$D(sk, c)$  :

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

Problems:

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

$$\mathbb{Z}_N = \{0, 1, 2, \dots, N-1\} \quad ; \quad (\mathbb{Z}_N)^* = \{\text{invertible elements in } \mathbb{Z}_N\}$$

Facts:  $x \in \mathbb{Z}_N$  is invertible  $\iff$   $\gcd(x, N) = 1$

– Number of elements in  $(\mathbb{Z}_N)^*$  is  $\phi(N) = (p-1)(q-1) = N - p - q + 1$

Euler's thm:

$$\forall x \in (\mathbb{Z}_N)^* : x^{\phi(N)} = 1 \text{ mod } N$$

# 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.  $e \cdot d = 1 \pmod{\phi(N)}$

output  $pk = (N, e)$  ,  $sk = (N, d)$   $e \cdot d = k \cdot \phi(N) + 1$   
 $x^{\phi(N)} = 1 \pmod N$

---

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

---

**F**<sup>-1</sup>(  $sk, y$  ) =  $y^d$  ;  $y^d = \mathbf{RSA}(x)^d = x^{\text{ed}} = x^{k\phi(N)+1} = \boxed{(x^{\phi(N)})^k} \cdot x = x$

# The RSA assumption

RSA assumption: RSA is one-way permutation

For all efficient algs.  $A$ :

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

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



# 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  $(E_s, D_s)$  *AES - 256.*

- **G()**: generate RSA params:  $pk = (N, e)$ ,  $sk = (N, d)$
- **E**(pk, m):
  - (1) choose random  $x$  in  $Z_N$
  - (2)  $y \leftarrow \text{RSA}(x) = x^e$ ,  $k \leftarrow H(x)$
  - (3) output  $(y, \text{E}_s(k, m))$
- **D**(sk, (y, c) ): output  $D_s( H(\text{RSA}^{-1}(y)) , c)$

# Textbook RSA is insecure

Textbook RSA encryption:

– public key:  $(N, e)$

Encrypt:  $c \leftarrow m^e \quad (\text{in } Z_N)$

– secret key:  $(N, d)$

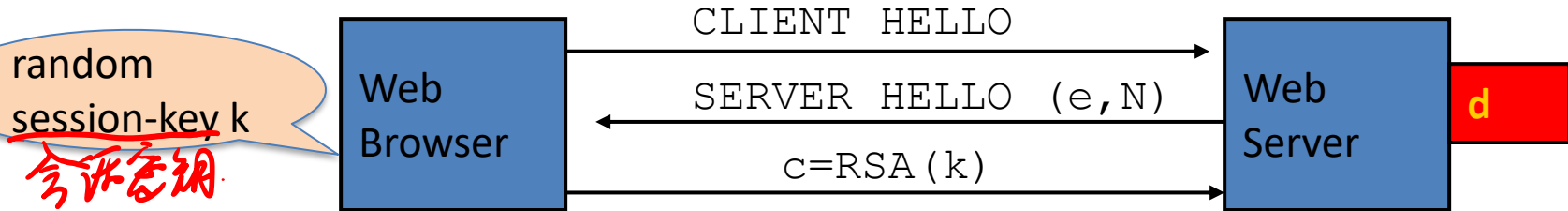
Decrypt:  $c^d \rightarrow m$

Insecure cryptosystem !!

– Is not semantically secure and many attacks exist

$\Rightarrow$  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:  $c = k^{e \cdot d}$  in  $Z_N$

If  $k = k_1 \cdot k_2$  where  $k_1, k_2 < 2^{34}$  (prob.  $\approx 20\%$ ) then  $c/k_1^e = k_2^e$  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:  $\approx 2^{40} \ll 2^{64}$



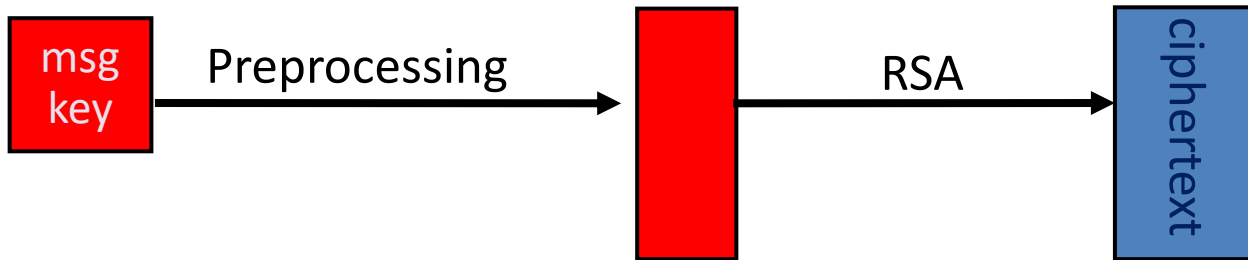
# Public Key Encryption from trapdoor permutations

## PKCS 1

# RSA encryption in practice

Never use textbook RSA.

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

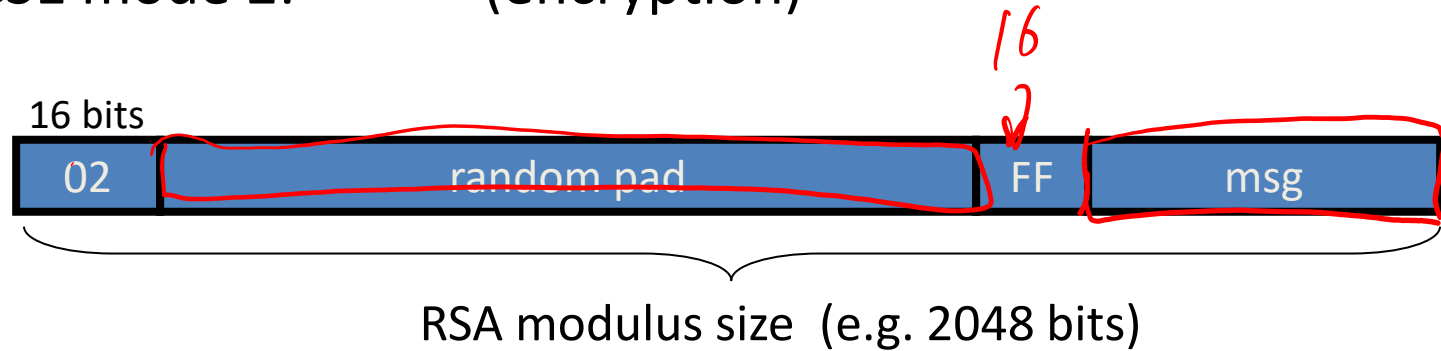


Main questions:

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

# PKCS1 v1.5

PKCS1 mode 2: (encryption)

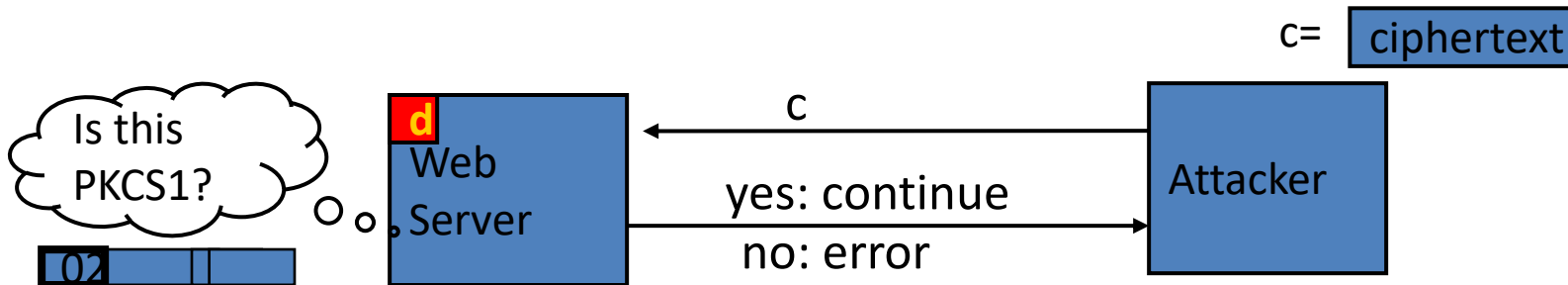


- Resulting value is RSA encrypted
- Widely deployed, e.g. in HTTPS

# Attack on PKCS1 v1.5

(Bleichenbacher 1998)

PKCS1 used in HTTPS:



⇒ attacker can test if 16 MSBs of plaintext = '02'

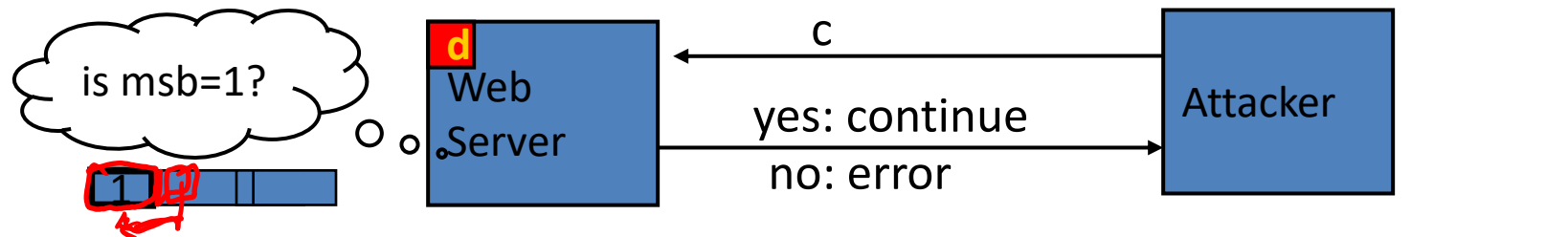
$$c = (\text{pkcs}(m))^e$$

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

- Choose  $r \in \mathbb{Z}_N$ . Compute  $c' \leftarrow \underline{r^e \cdot c} = (\underline{r \cdot \text{PKCS1}(m)})^e$
- Send  $c'$  to web server and use response  $\underline{r \cdot \text{pkcs1}(m)} \stackrel{?}{=} 02$

# Baby Bleichenbacher

compute  $x \leftarrow c^d$  in  $Z_N$



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

- Sending  $c$  reveals  $\text{msb}(x)$  ✗ & ✗
- Sending  $2^e \cdot c = (2x)^e$  in  $Z_N$  reveals  $\text{msb}(2x \bmod N) = \text{msb}_2(x)$
- Sending  $4^e \cdot c = (4x)^e$  in  $Z_N$  reveals  $\text{msb}(4x \bmod N) = \text{msb}_3(x)$
- ... and so on to reveal all of  $x$



# HTTPS Defense (RFC 5246)

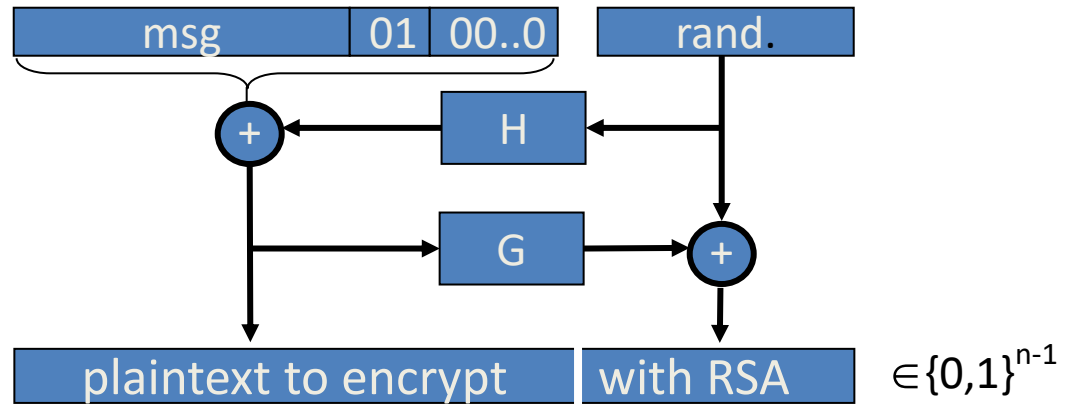
*Attacks discovered by Bleichenbacher and Klima et al. ... can be avoided by treating incorrectly formatted message blocks ... in a manner indistinguishable from correctly formatted RSA blocks. In other words:*

- 1. Generate a string **R** of 46 random bytes*
- 2. Decrypt the message to recover the plaintext  $M$*
- 3. If the PKCS#1 padding is not correct*  
$$\text{pre\_master\_secret} = \textbf{R}$$

# PKCS1 v2.0: OAEP

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*

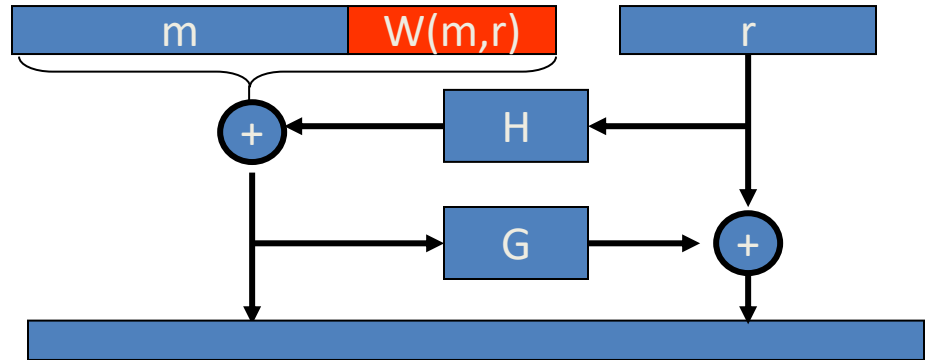
in practice: use SHA-256 for  $H$  and  $G$

# OAEP Improvements

**OAEP+:** [Shoup'01]

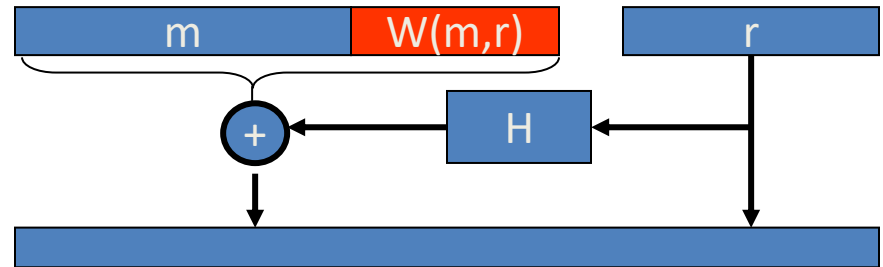
$\forall$  trap-door permutation  $F$   
F-OAEP+ is CCA secure when  
 $H, G, W$  are *random oracles*.

During decryption validate  $W(m,r)$  field.

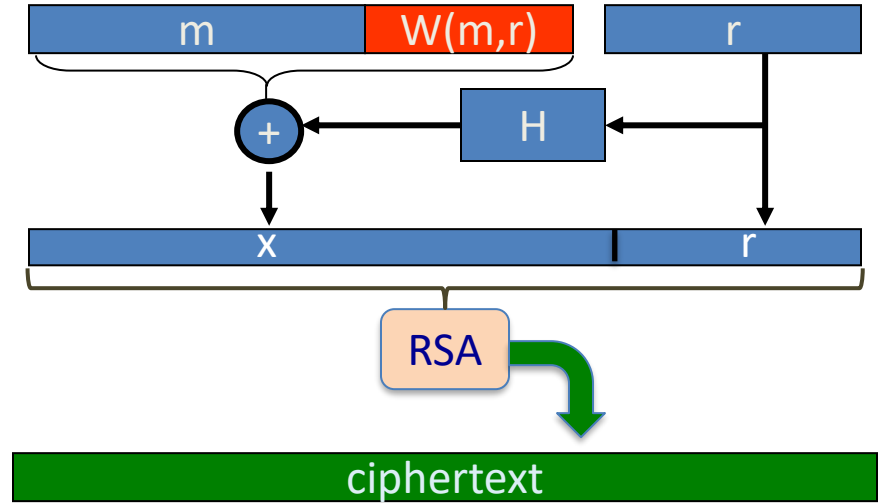


**SAEP+:** [B'01]

RSA ( $e=3$ ) is a trap-door perm  $\Rightarrow$   
RSA-SAEP+ is CCA secure when  
 $H, W$  are *random oracle*.



How would you decrypt  
an SAEP ciphertext **ct** ?



- ☐  $(x,r) \leftarrow \text{RSA}^{-1}(\text{sk}, \text{ct})$  ,  $(m,w) \leftarrow x \oplus H(r)$  , output  $m$  if  $w = W(m,r)$
- ☐  $(x,r) \leftarrow \text{RSA}^{-1}(\text{sk}, \text{ct})$  ,  $(m,w) \leftarrow r \oplus H(x)$  , output  $m$  if  $w = W(m,r)$
- ☐  $(x,r) \leftarrow \text{RSA}^{-1}(\text{sk}, \text{ct})$  ,  $(m,w) \leftarrow x \oplus H(r)$  , output  $m$  if  $r = W(m,x)$

# Subtleties in implementing OAEP

[M '00]

```
OAEP-decrypt(ct):  
    error = 0;  
    .....  
    if (  $\text{RSA}^{-1}(\text{ct}) > 2^{n-1}$  )  
        { error = 1; goto exit; }  
    .....  
    if (  $\text{pad}(\text{OAEP}^{-1}(\text{RSA}^{-1}(\text{ct}))) \neq \text{"01000"}$ )  
        { error = 1; goto exit; }
```

Problem: timing information leaks type of error

⇒ Attacker can decrypt any ciphertext

Lesson: Don't implement RSA-OAEP yourself !



## 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:

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

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)

# 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  $N$   
 $\Rightarrow$  efficient algorithm for factoring  $N$ .
- Oldest problem in public key cryptography.

Some evidence no reduction exists: (BV'98)

- “Algebraic” reduction  $\Rightarrow$  factoring is easy.



# How **not** to improve RSA's performance

To speed up RSA decryption use small private key  $d$  ( $d \approx 2^{128}$ )

$$c^d = m \pmod{N}$$

Wiener'87: if  $d < N^{0.25}$  then RSA is insecure.

BD'98: if  $d < N^{0.292}$  then RSA is insecure (open:  $d < N^{0.5}$ )

Insecure: priv. key  $d$  can be found from  $(N, e)$



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

Encryption: 17 multiplications

Asymmetry of RSA: fast enc. / slow dec.

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

# Key lengths

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

Cipher key-size

80 bits

128 bits

256 bits (AES)

RSA

Modulus size

1024 bits

3072 bits

**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.

# An Example Fault Attack on RSA (CRT)

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

$$\left. \begin{array}{l} \text{decrypt mod } p: \quad x_p = c^d \text{ in } Z_p \\ \text{decrypt mod } q: \quad x_q = c^d \text{ in } Z_q \end{array} \right\} \text{ combine to get } x = c^d \text{ in } Z_N$$

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  $Z_q$

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

# 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