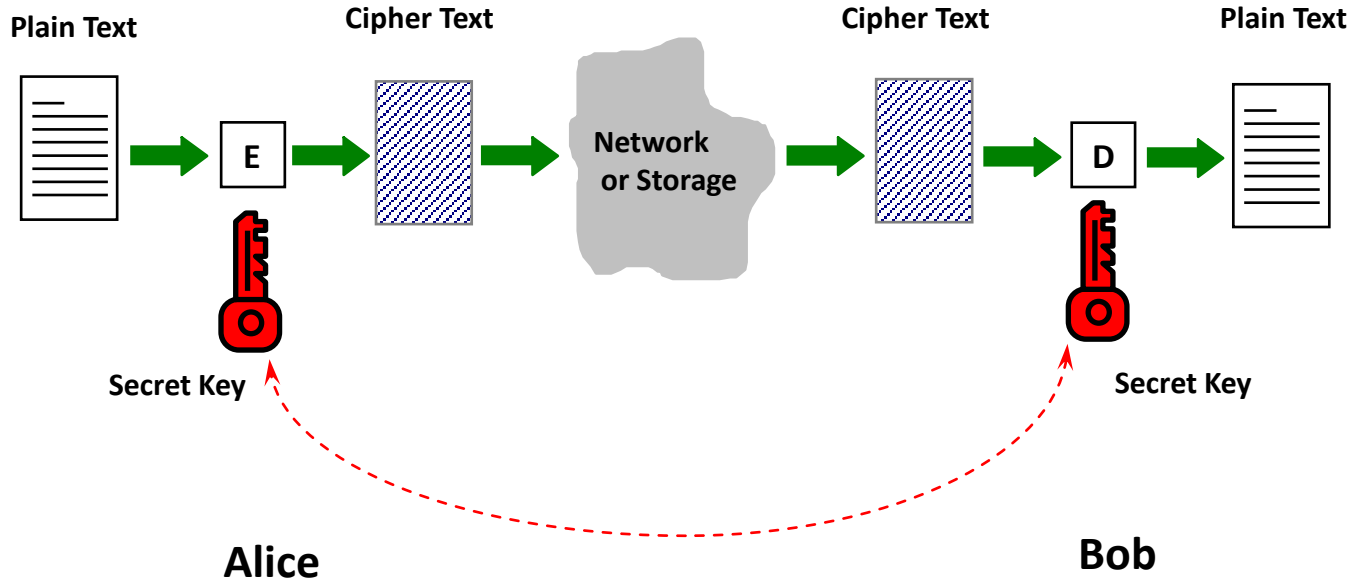


# **Asymmetric Cryptography**

Public key encryption:  
definitions and security

# Symmetric Cipher



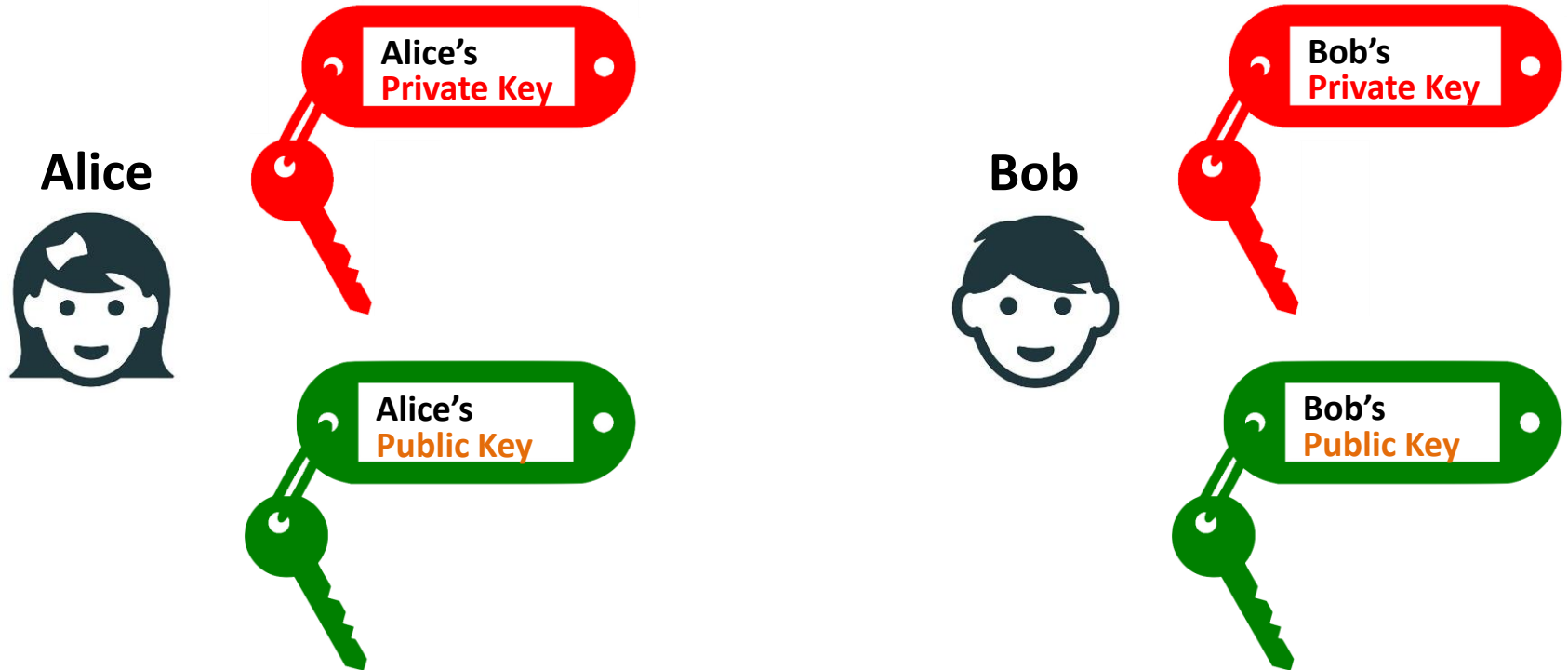
# Problems with Symmetric Ciphers

- In order for Alice & Bob to be able to communicate securely using a symmetric cipher, such as AES, they have to have a **shared key** in the first place.
  - What if they have never met before?
- Alice needs to keep 100 different keys if she wishes to communicate with 100 different people

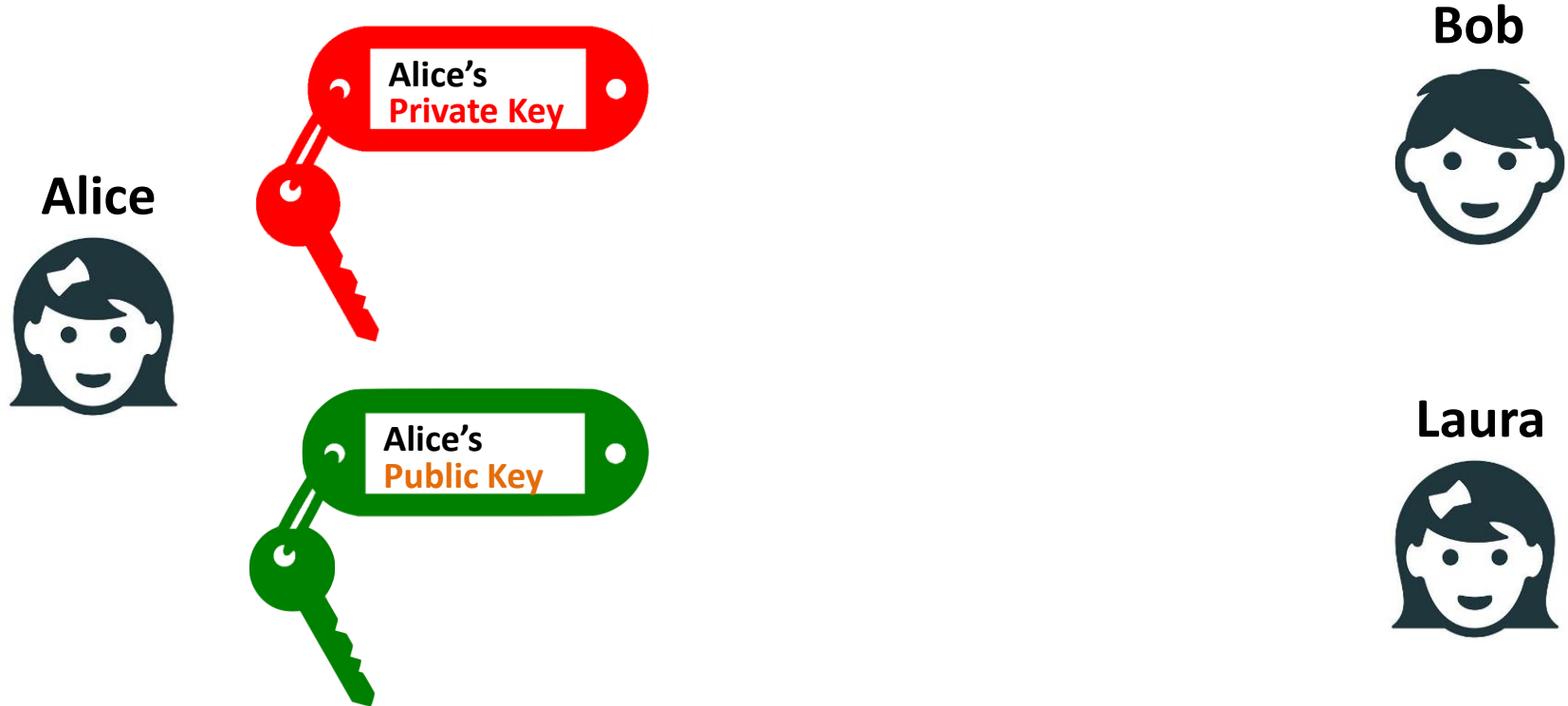
# Motivation of Asymmetric Cryptography

- Is it possible for Alice & Bob, who have no shared secret key, to communicate securely?
- This led to **Asymmetric Cryptography**

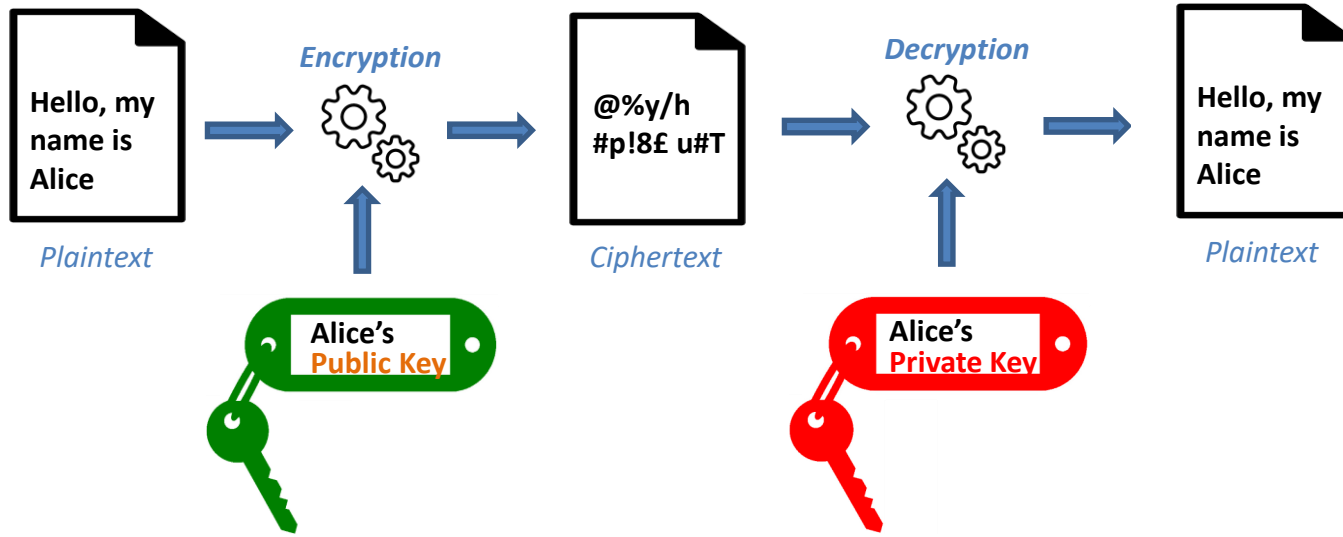
# Asymmetric Cryptography



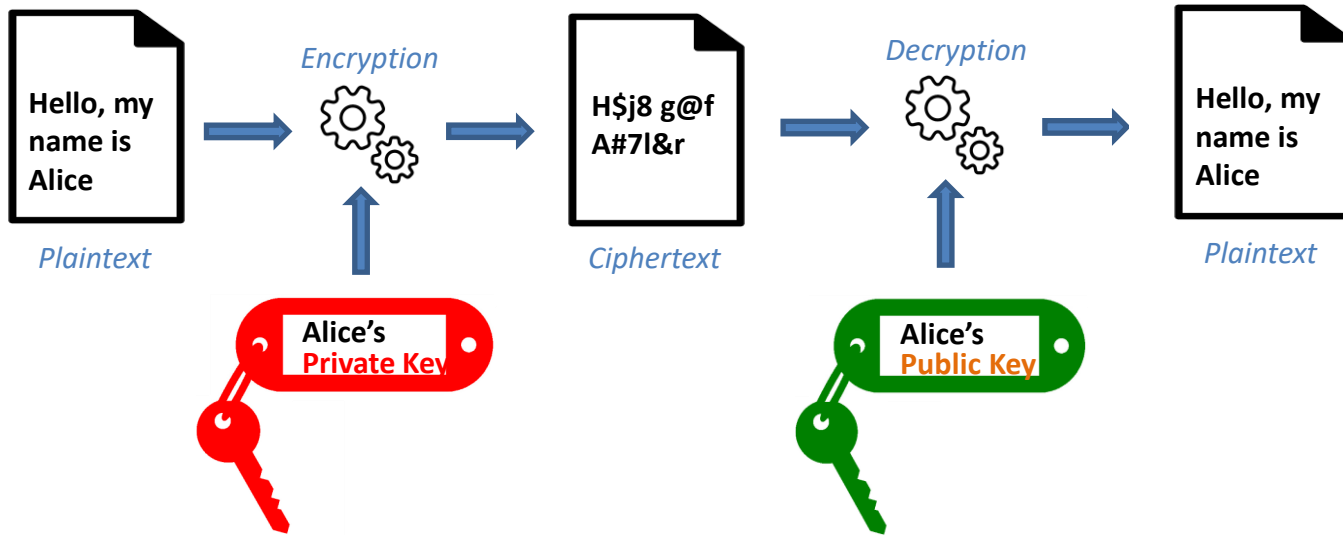
# Asymmetric Cryptography



# Public and private keys

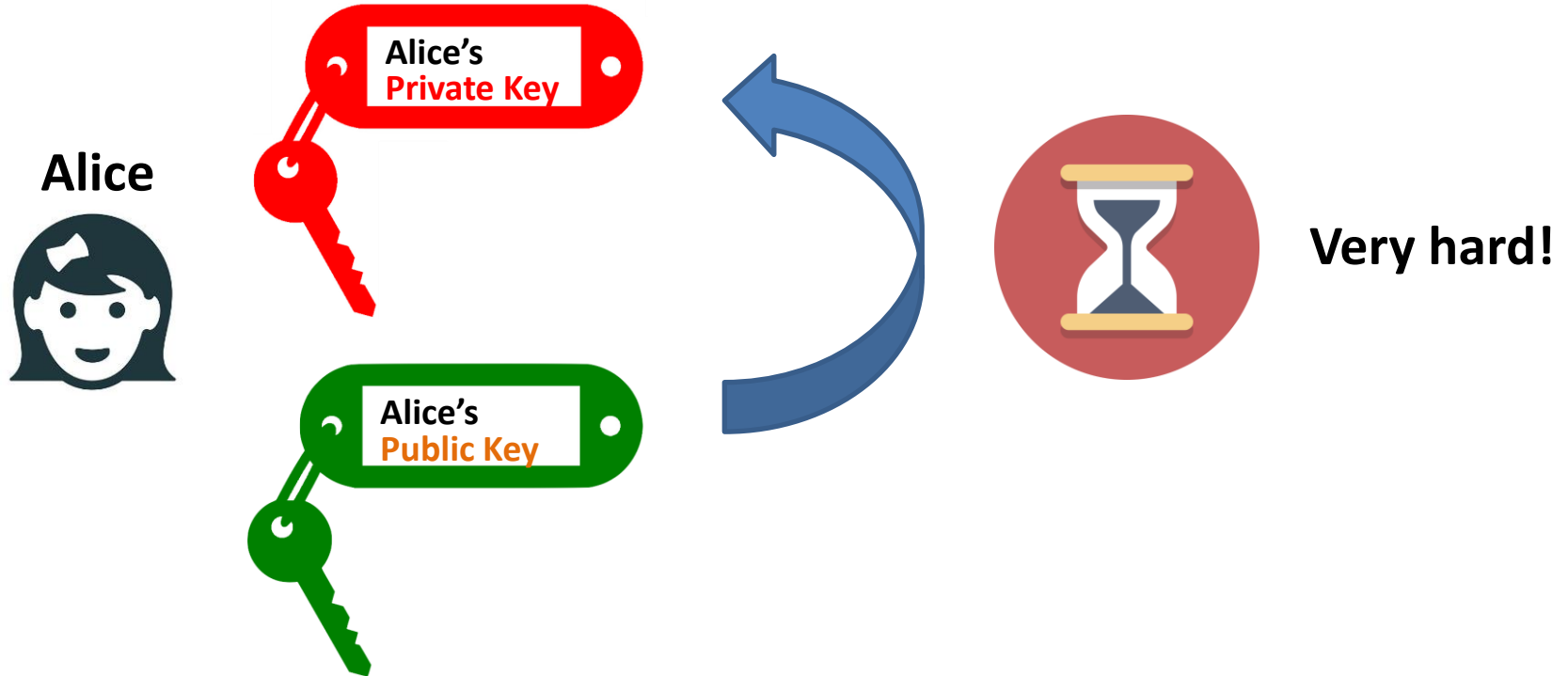


# Public and private keys





# Public and private keys



# Asymmetric Cryptography

- **Public** key
- **Private** key
- $E(\text{private-key}_{\text{Alice}}, m) = c$
- $D(\text{public-key}_{\text{Alice}}, c) = m$
- $E(\text{public-key}_{\text{Alice}}, m) = c$
- $D(\text{private-key}_{\text{Alice}}, c) = m$

# Main ideas

- Bob:
  - **publishes**, say in Yellow/White pages, his **public key**, and
  - **keeps** to himself the **matching private key**.

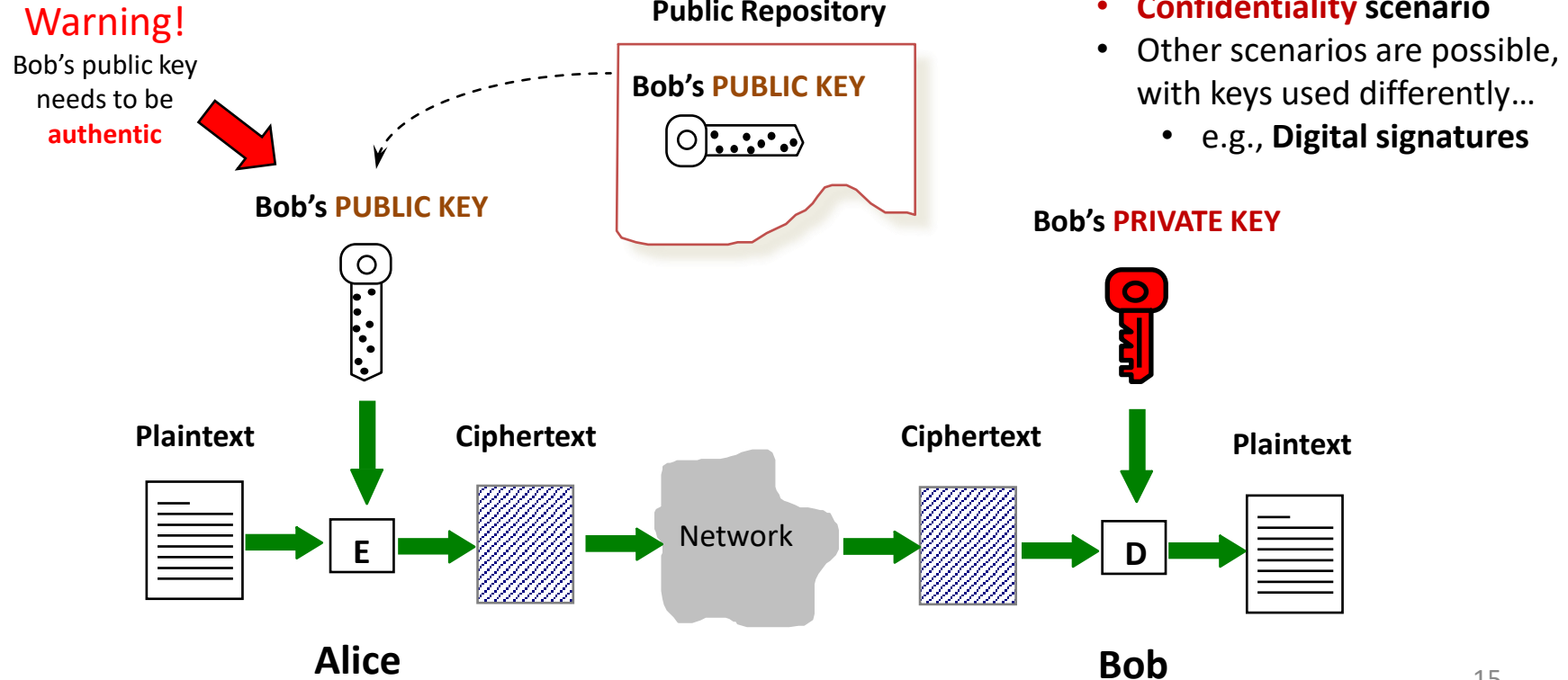
# Main ideas (Confidentiality)

- Alice:
  - Looks up the phone book, and **finds out Bob's public key**
  - **Encrypts** a message using **Bob's public key** and the encryption algorithm.
  - **Sends the ciphertext** to Bob.

# Main ideas (Confidentiality)

- Bob:
  - **Receives the ciphertext** from Alice.
  - **Decrypts** the ciphertext **using his private key**, together with the decryption algorithm

# Asymmetric Encryption



# Main differences with Symmetric Crypto

- The public key is different from the private key.
- Infeasible for an attacker to find out the private key from the public key.
- No need for Alice & Bob to distribute a shared secret key beforehand!
- Only one pair of public and private keys is required for each user!

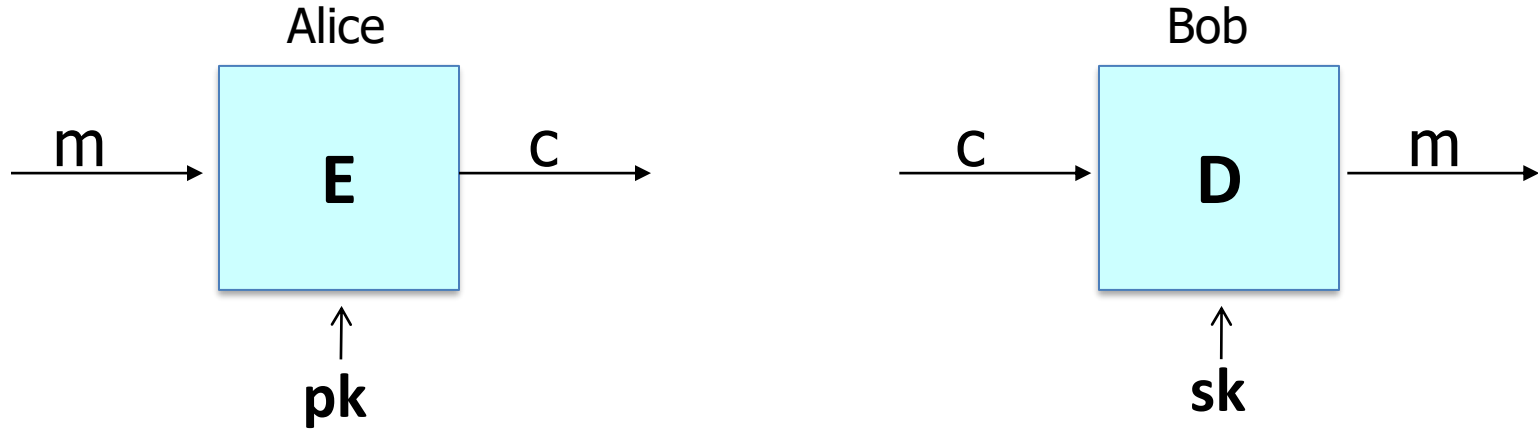
# Let's start seriously

- define what is public key encryption
- what it means for public key encryption to be secure



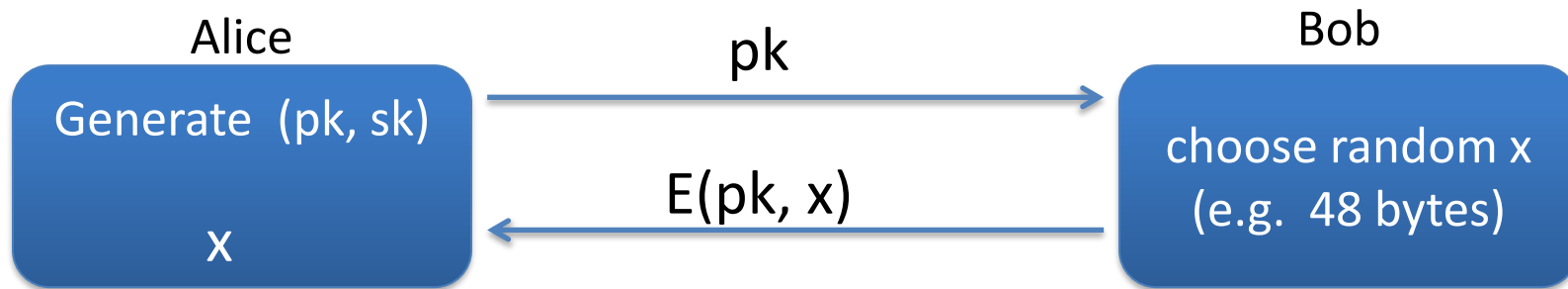
# Public key encryption

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.  $(G, E, D)$

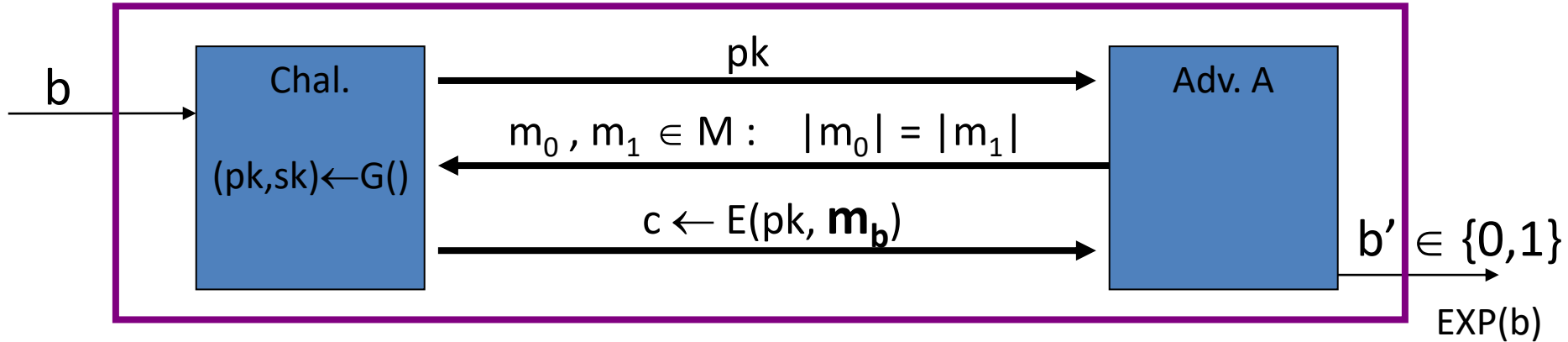
- $G()$ : randomized alg. outputs a key pair  $(pk, sk)$
- $E(pk, m)$ : randomized alg. that takes  $m \in M$  and outputs  $c \in C$
- $D(sk, c)$ : det. alg. that takes  $c \in C$  and outputs  $m \in M$  or  $\perp$

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

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

# 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  $A$ :

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

# 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

For public key encryption:

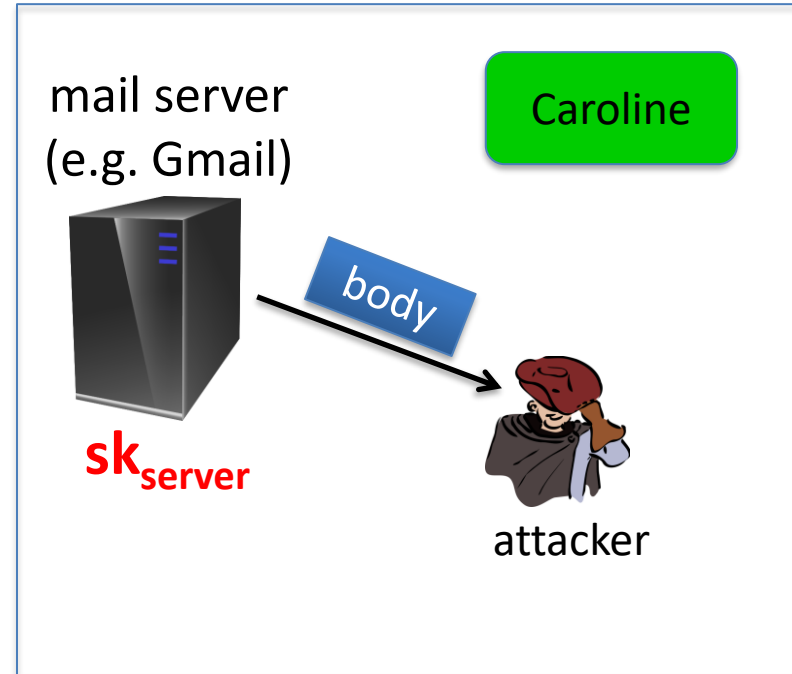
- One-time security  $\Rightarrow$  many-time security (CPA)  
(follows from the fact that attacker can encrypt by himself)
- Public key encryption **must** be randomized

# Security against active attacks

What if attacker can tamper with ciphertext?

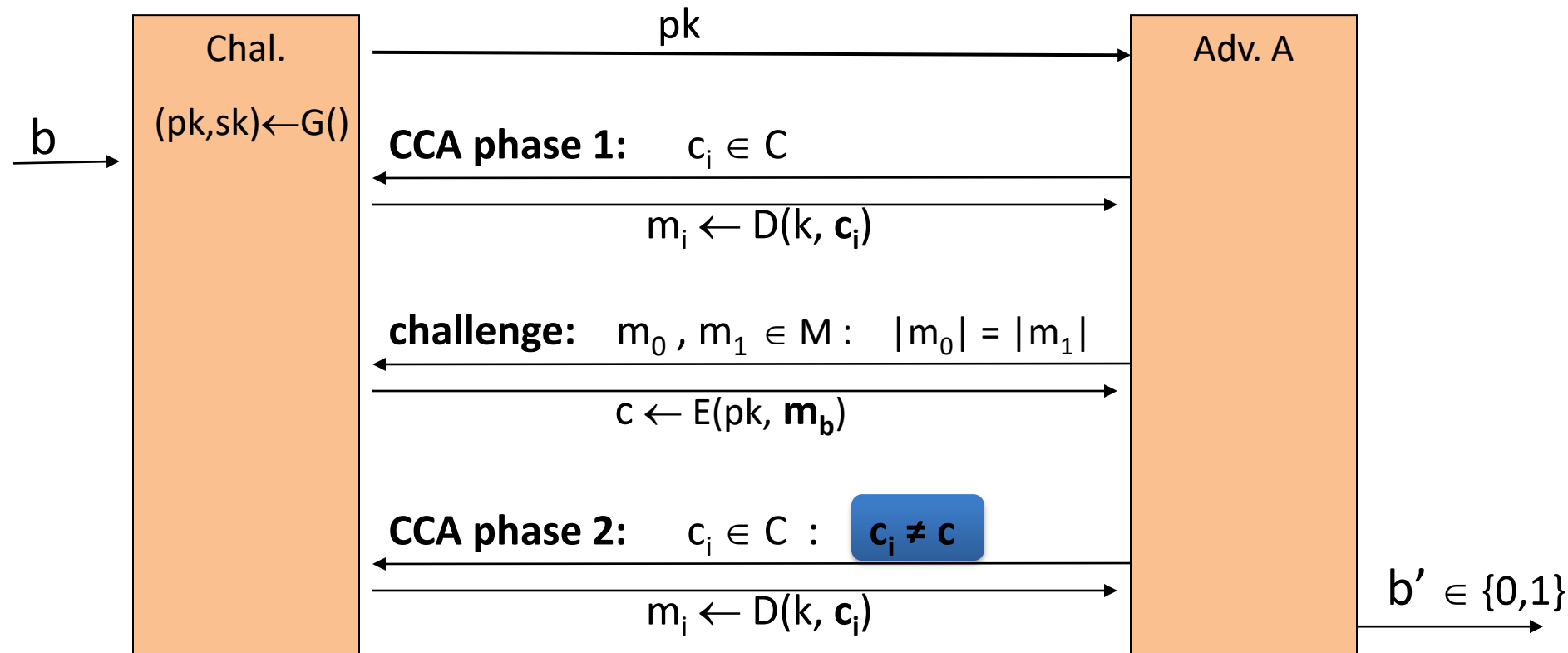


Attacker is given decryption of msgs  
that start with **“to: attacker”**



# (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  $A$ :

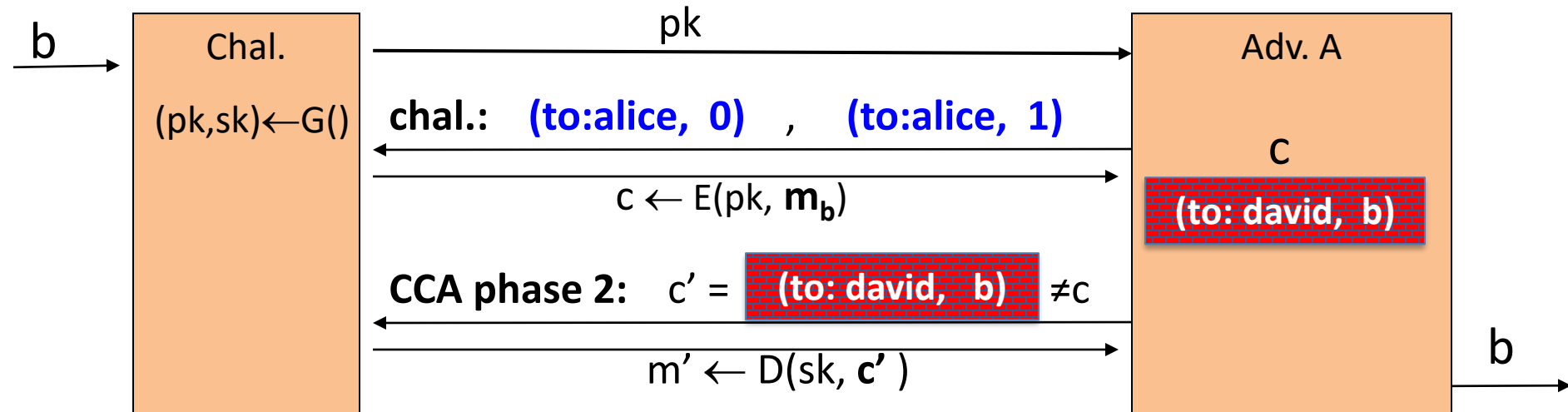
$$\text{Adv}_{\text{CCA}}[A, \mathbb{E}] = \left| \Pr[\text{EXP}(0)=1] - \Pr[\text{EXP}(1)=1] \right| \text{ is negligible.}$$

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

# Trapdoor Permutations

# 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  $F(pk, \cdot)$

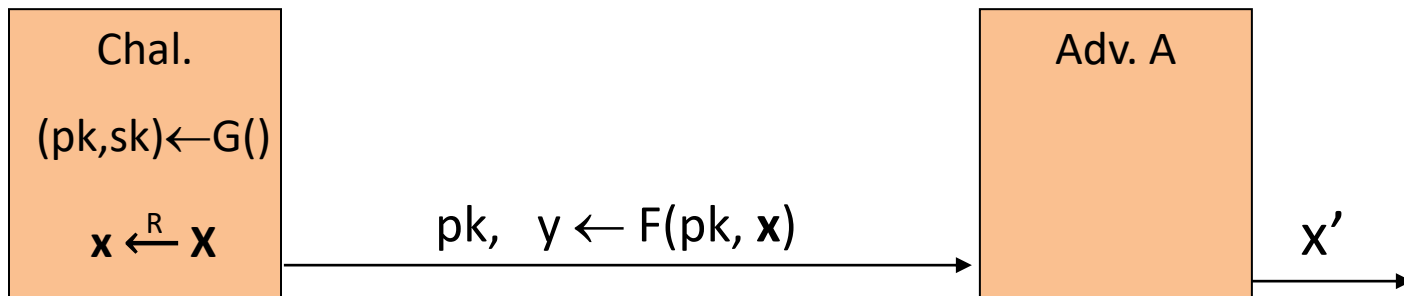
More precisely:  $\forall (pk, sk)$  output by  $G$

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

# 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

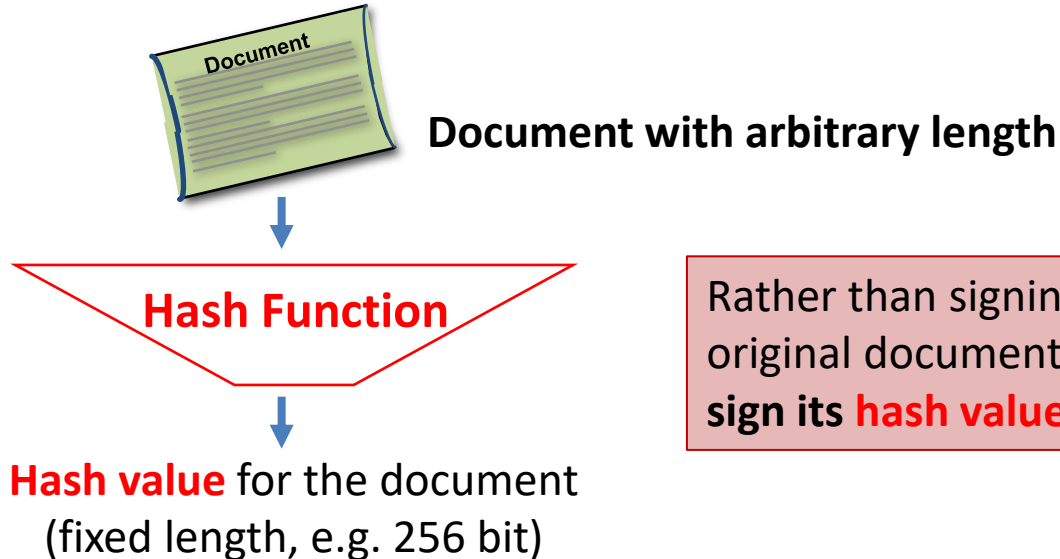
- $(G, F, F^{-1})$ : secure TDF  $X \rightarrow Y$
- $(E_s, D_s)$ : symmetric auth. encryption defined over  $(K, M, C)$
- $H: X \rightarrow K$  a hash function

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

Key generation  $G$ : same as  $G$  for TDF

# Hash Functions

- **Hash functions:**
  - **Input:** arbitrary length
  - **Output:** fixed length (generally much shorter than the input)



# One-Way Hash Algorithm

- A one-way hash algorithm hashes an input document into a condensed short output (say of 256 bits)
  - Denoting a one-way hash algorithm by  $H(.)$ , we have:
    - Input:  $m$  - a binary string of any length
    - Output:  $H(m)$  - a binary string of  $L$  bits, called the “hash of  $m$  under  $H$ ”.
    - The output length parameter  $L$  is fixed for a given one-way hash function  $H$ ,
    - Examples:
      - The one-way hash function “MD5” has  $L = 128$  bits
      - The one-way hash function “SHA-1” has  $L = 160$  bits

# Properties of One-Way Hash Algorithm

- A good one-way hash algorithm  $H$  needs to have these properties:
  - 1. **Easy to Evaluate:**
    - The hashing algorithm should be fast
  - 2. **Hard to Reverse:**
    - There is no feasible algorithm to “**reverse**” a hash value,
    - That is, given any hash value  $h$ , it is computationally infeasible to find any document  $m$  such that  $H(m) = h$ .
  - 3. **Hard to find Collisions:**
    - There is no feasible algorithm to find two or more input documents which are hashed into the same condensed output,
    - That is, it is computationally infeasible to find any two documents  $m_1$ ,  $m_2$  such that  $H(m_1) = H(m_2)$ .
  - 4. **A small change to a message should change the hash value so extensively** that the new hash value appears uncorrelated with the old hash value



# 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

$E(pk, m)$  :

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

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

output  $(y, c)$

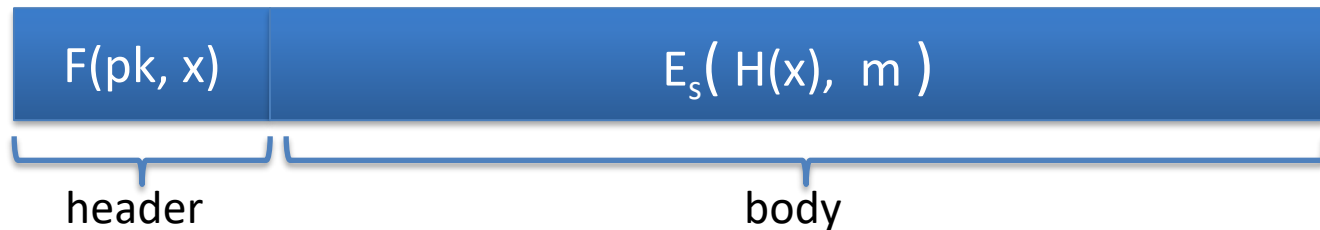
$D(sk, (y, c))$  :

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

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

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)

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

$E(pk, m)$  :

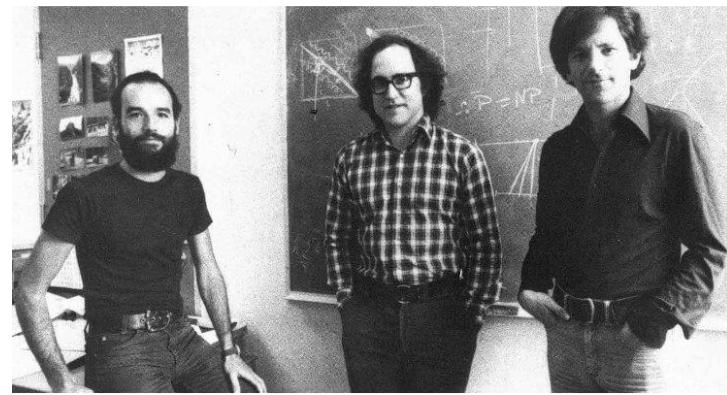
output  $c \leftarrow F(pk, m)$

$D(sk, c)$  :

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

Problems:

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



# The RSA trapdoor permutation

- One of the first practical responses to the challenge posed by Diffie-Hellman was developed by ***Ron Rivest, Adi Shamir, and Len Adleman*** of MIT in 1977
- Resulting algorithm is known as *RSA*
- Based on properties of *prime numbers* and results from *number theory*

# 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 where  $p, q \approx N^{1/2}$

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

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:

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

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

---

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

---

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



# The RSA assumption

RSA assumption: RSA is one-way permutation

For all efficient algs.  $A$ :

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

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

# 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  $(E_s, D_s)$

- **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  $(y, E_s(k, m))$
- **D**(sk, (y, c) ): output  $D_s(H(\text{RSA}^{-1}(y)), c) \rightarrow m$

# Textbook RSA is insecure

Textbook RSA encryption:

- public key:  $(N, e)$

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

- secret key:  $(N, d)$

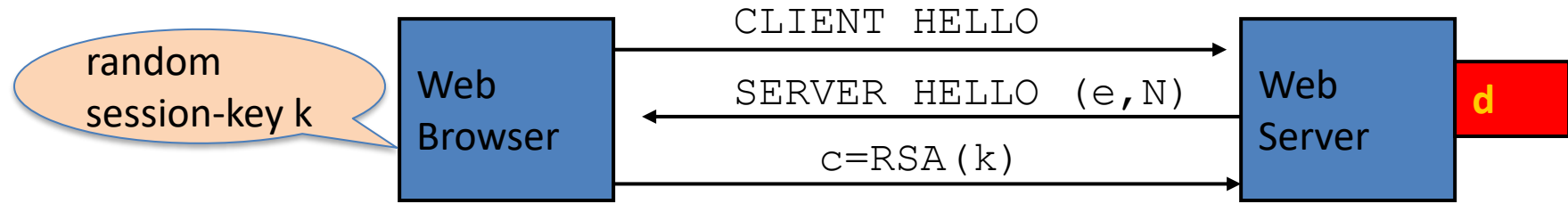
Decrypt:  $c^d \rightarrow m$

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:  $c = k^e$  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  $Z_N$

Meet-in-the-middle attack:

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:  $\approx 2^{40} \ll 2^{64}$

# Is RSA a one-way function?

Is it really hard to invert RSA without knowing the trapdoor?

# 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  $(N, e)$



# Wiener's attack

$$(N, e) \Rightarrow d \text{ and } d < N^{0.25}/3$$

$$\text{Recall: } e \cdot d = 1 \pmod{\varphi(N)} \Rightarrow \exists k \in \mathbb{Z}: e \cdot d = k \cdot \varphi(N) + 1$$

$$\left| \frac{e}{\varphi(N)} - \frac{k}{d} \right| = \frac{1}{d \cdot \varphi(N)} \leq \frac{1}{\sqrt{N}}$$

---

$$\varphi(N) = N - p - q + 1 \Rightarrow |N - \varphi(N)| \leq p + q \leq 3\sqrt{N}$$

$$d \leq N^{0.25}/3 \Rightarrow \frac{1}{2d^2} - \frac{1}{\sqrt{N}} \geq \frac{3}{\sqrt{N}} \quad \left| \frac{e}{N} - \frac{k}{d} \right| \leq \left| \frac{e}{N} - \frac{e}{\varphi(N)} \right| + \left| \frac{e}{\varphi(N)} - \frac{k}{d} \right| \leq \frac{1}{2d^2}$$

---

Continued fraction expansion of  $e/N$  gives  $k/d$ .

$$e \cdot d = 1 \pmod{k} \Rightarrow \gcd(d, k) = 1 \Rightarrow \text{can find } d \text{ from } k/d$$

# RSA in Practice

# RSA With Low public exponent

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

- Minimum value:  **$e=3$**  ( $\gcd(e, \phi(N)) = 1$ ) (Q: why not 2?)
- Recommended value:  **$e=65537=2^{16}+1$**

Encryption: 17 multiplications

Asymmetry of RSA: fast enc. / slow dec.

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

# Key lengths

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

<u>Cipher key-size</u>	RSA <u>Modulus size</u>
80 bits	1024 bits
128 bits	3072 bits
256 bits (AES)	<b><u>15360</u></b> 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  $Z_q$

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

# 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