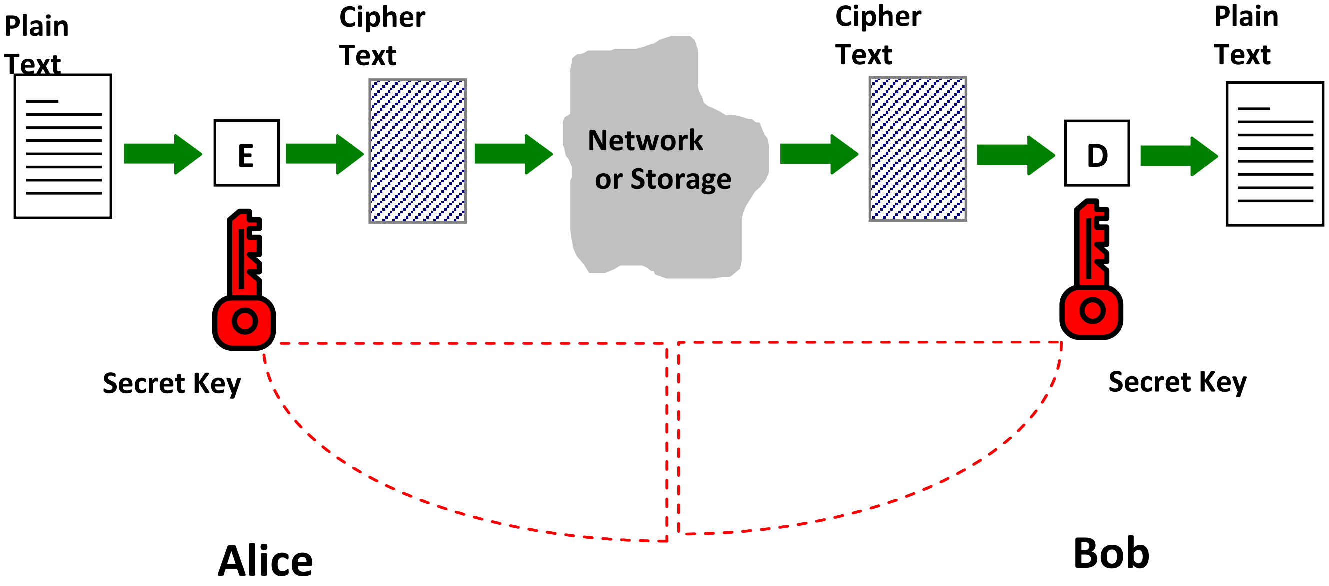
Asymmetric Cryptography

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 must have a **shared key** in the first place.
  - What if they have never met before?

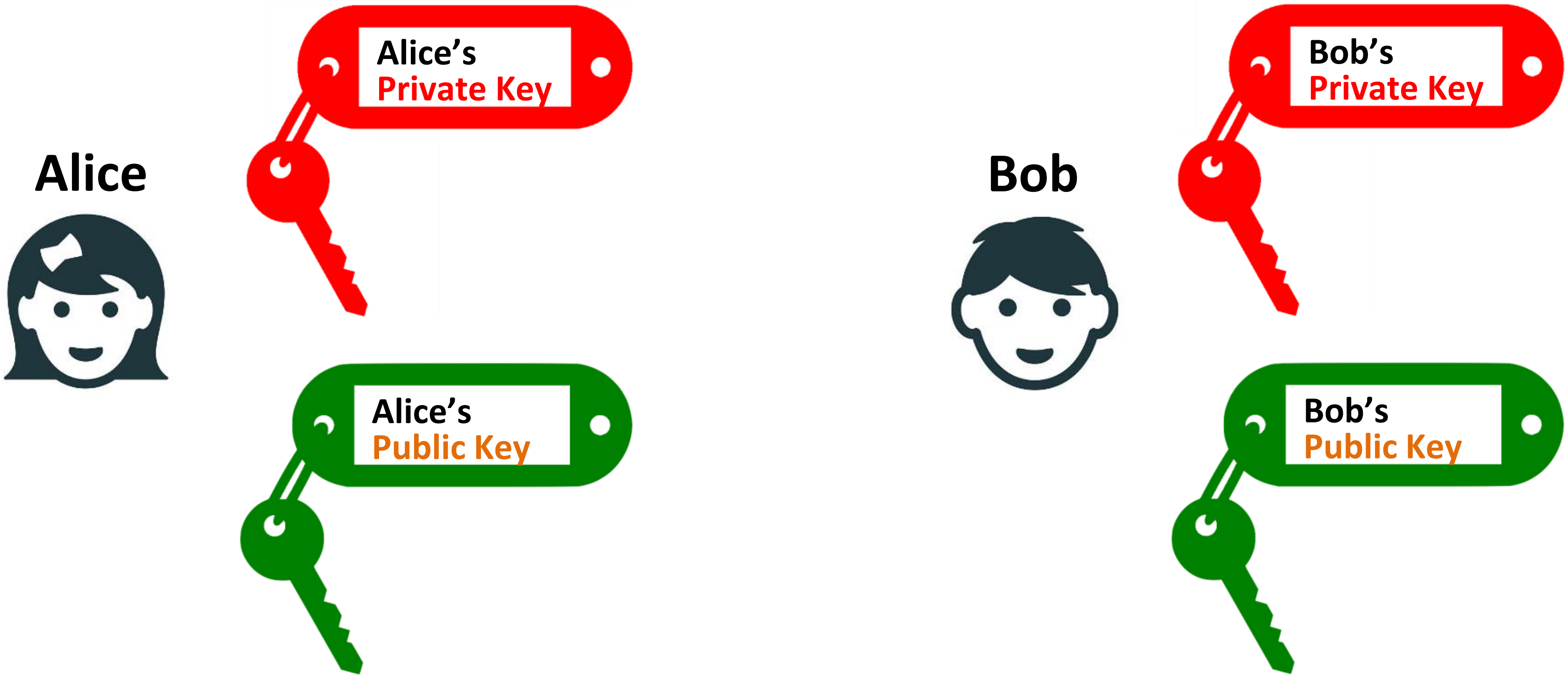
- Alice needs to keep 100 different keys if she wishes to communicate with 100 different people

Motivation of Asymmetric Cryptography

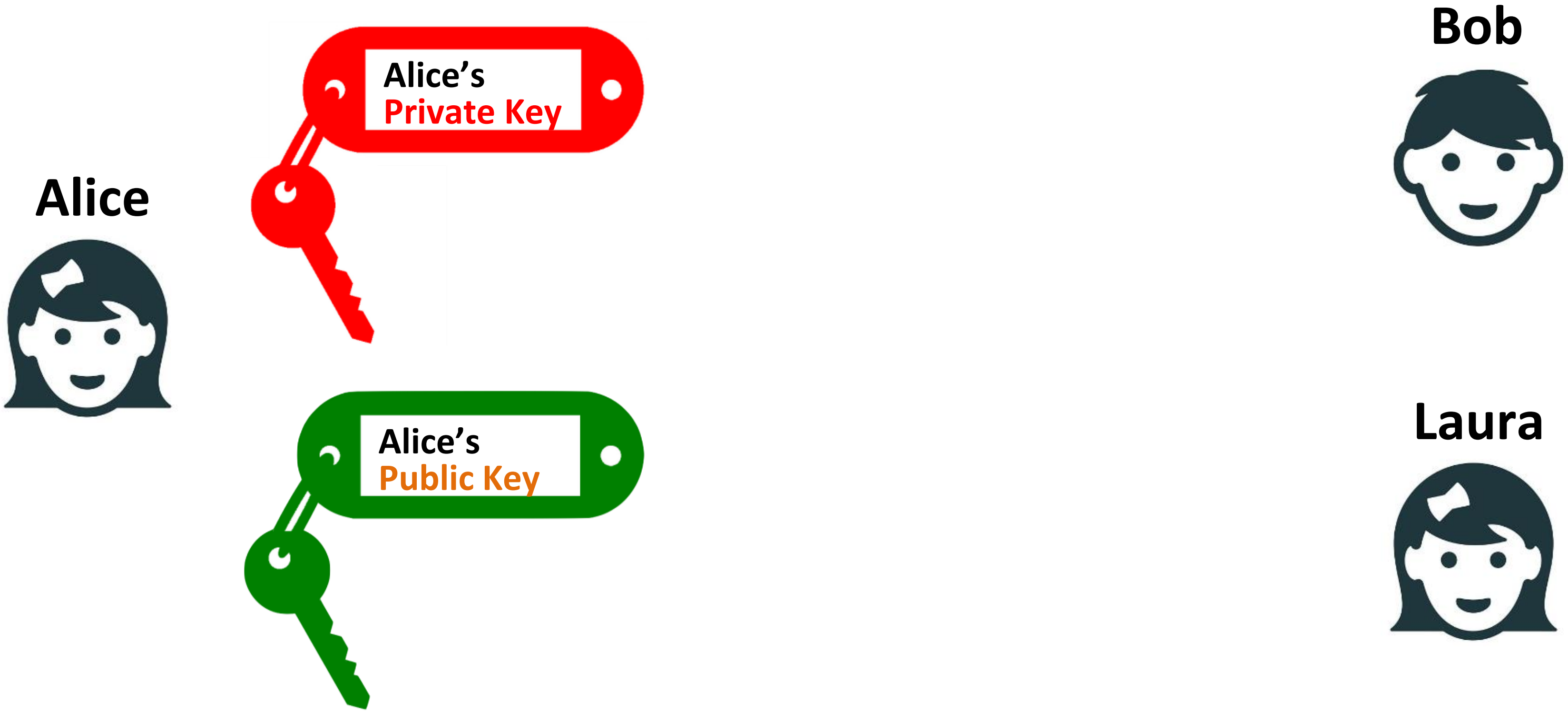
- Is it possible for Alice & Bob, who have no shared secret key, to communicate securely?

- This led to **Asymmetric Cryptography**

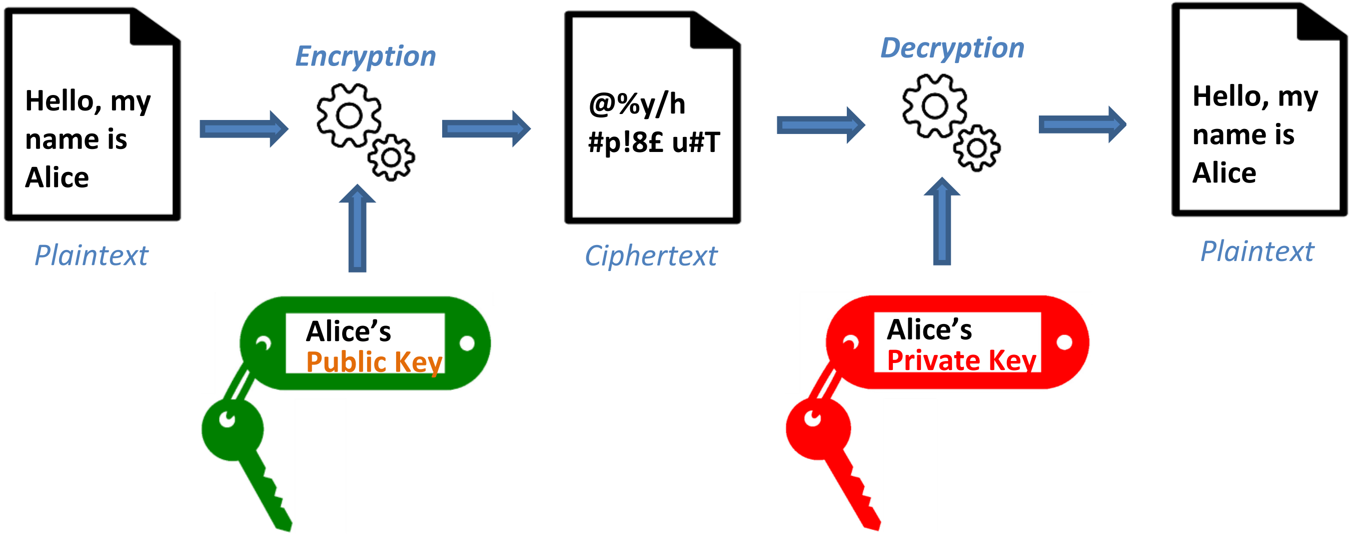
Asymmetric Cryptography



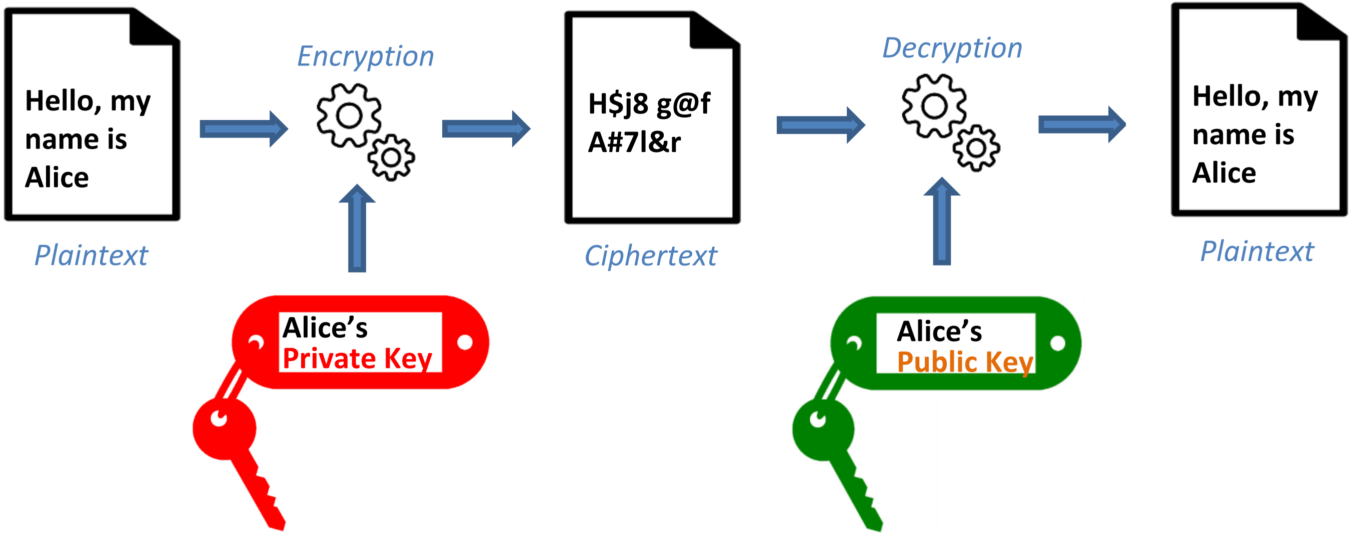
Asymmetric Cryptography



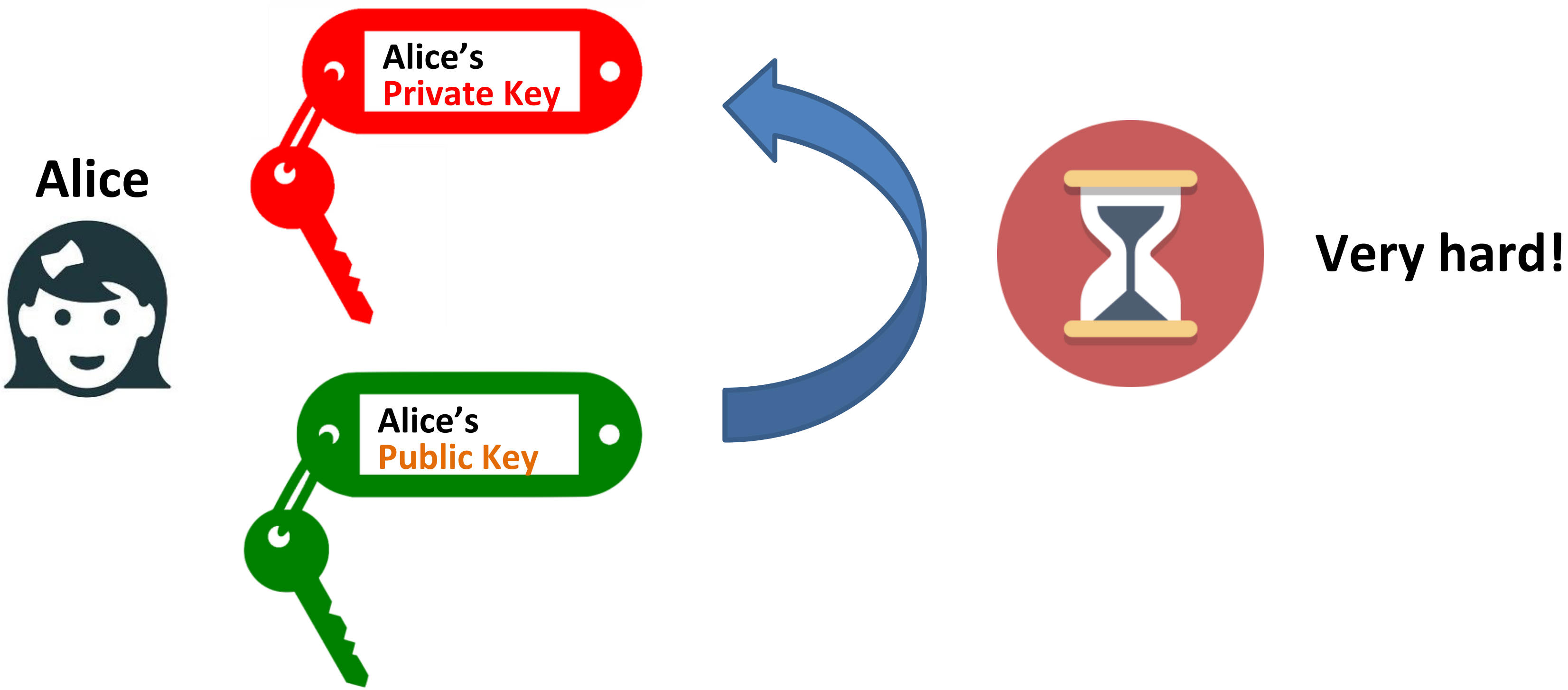
Public and private keys



Public and private keys



Public and private keys



Asymmetric Cryptography

- Public key
- Private key

- E(private-key$_{Alice,}$ m) = c
- D(public-key$_{Alice,}$ c) = m

- E(public-key$_{Alice,}$ m) = c
- D(private-key$_{Alice,}$ 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)

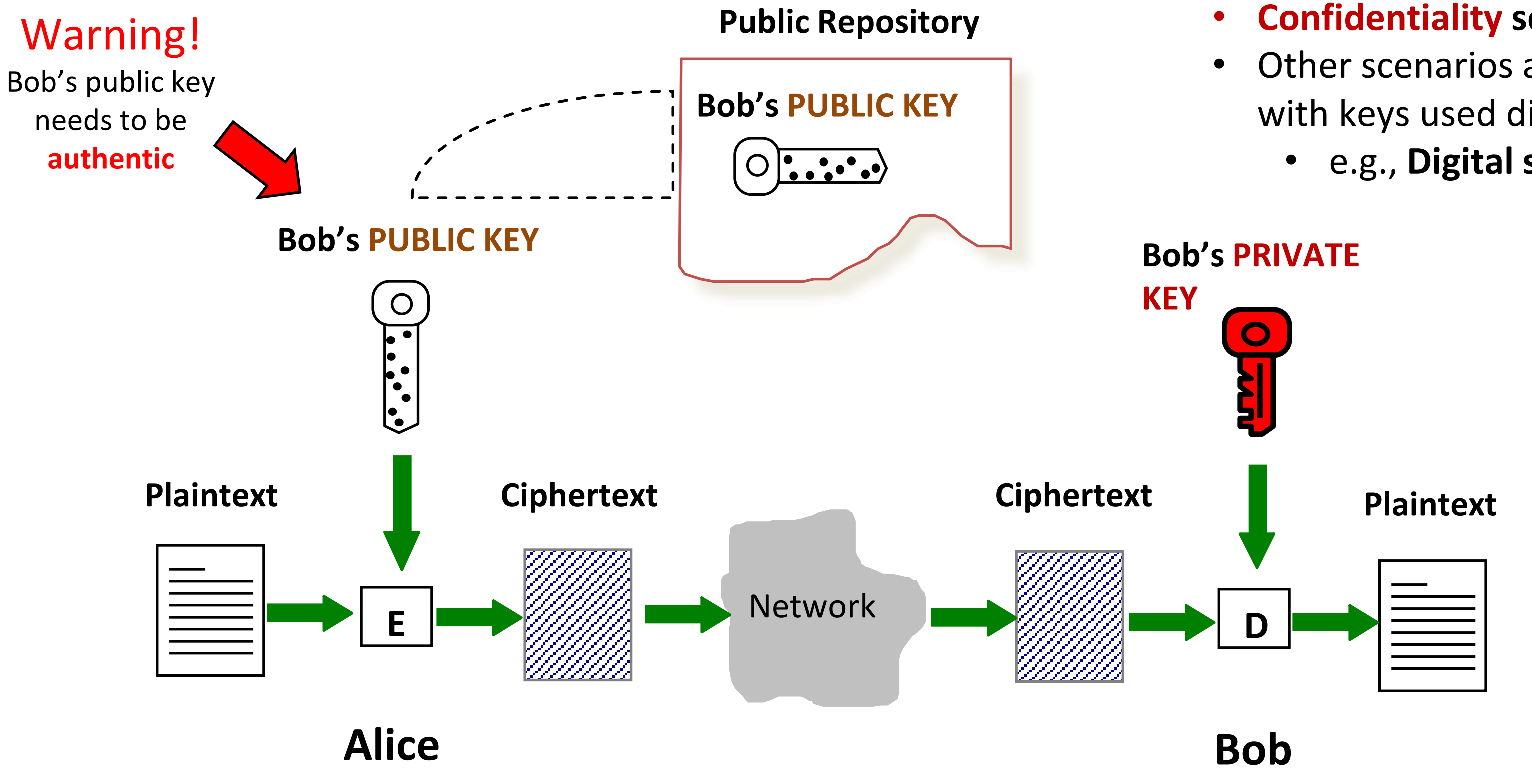
- Alice:
  - Looks up the phone book, and **finds out Bob's public key**
  - **Encrypts a message using Bob's public key** and the encryption algorithm.
  - **Sends the ciphertext** to Bob.

Main ideas (Confidentiality)

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

Asymmetric Encryption

- **Confidentiality scenario**
- Other scenarios are possible, with keys used differently…
  - e.g., **Digital signatures**



Main differences with Symmetric Crypto

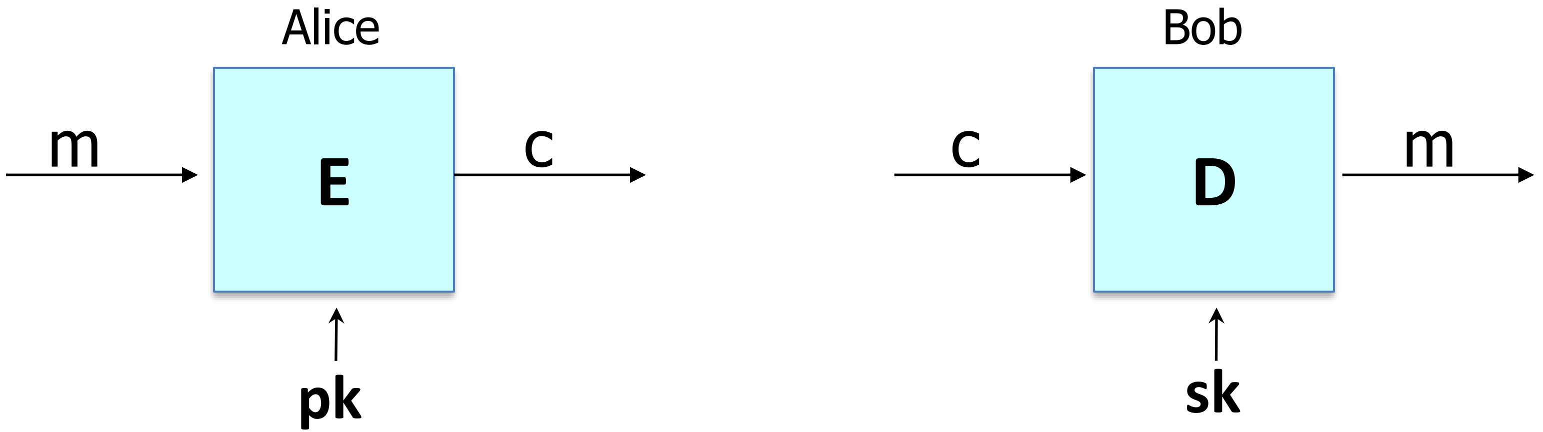
- The ***public key*** is different from the ***private key***.
- Infeasible for an attacker to find out the private key from the public key.
- No need for Alice and Bob to distribute a shared secret key beforehand!
- Only one pair of public and private keys is required for each user!

Let’s start seriously

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

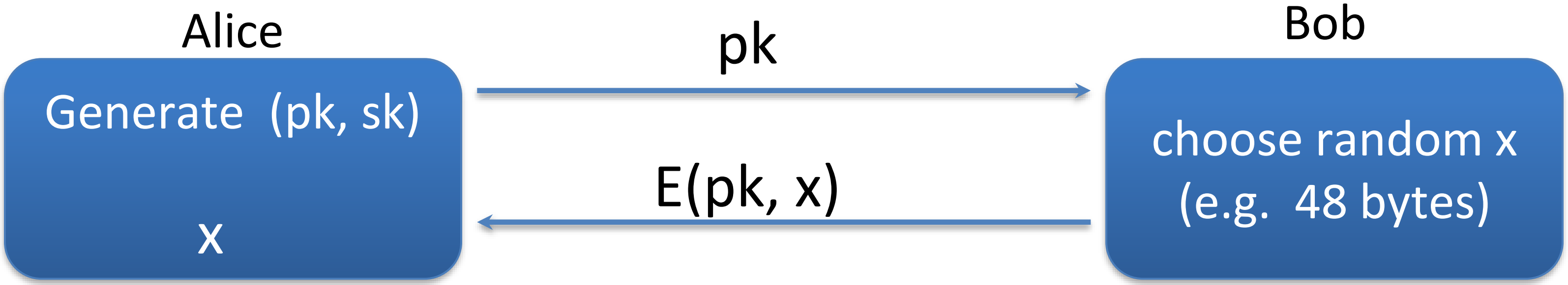
Public key encryption

Bob: generates (p_k, s_k) and gives p_k 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

<u>Def</u>: a public-key encryption system is a triple of algs. (G, E, D)

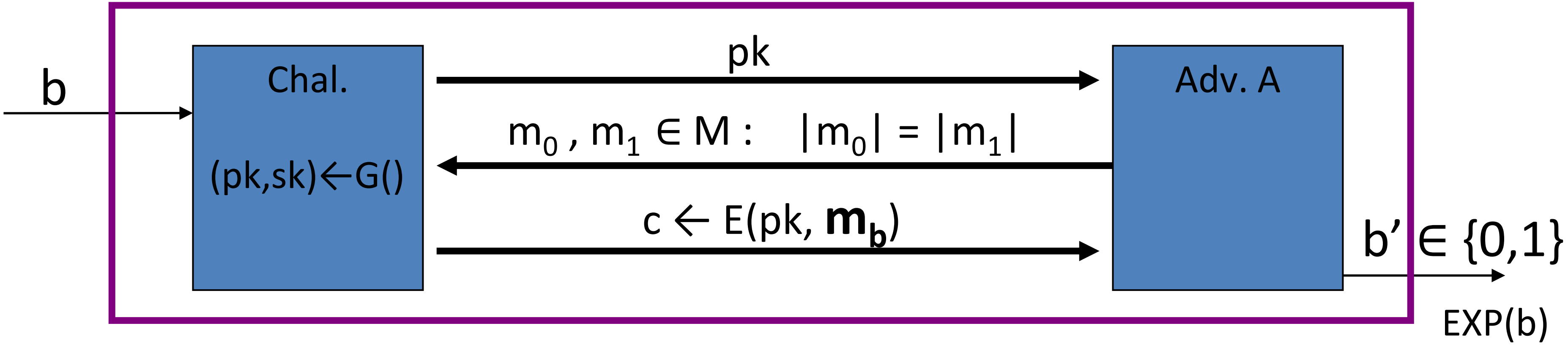
- **G**(): randomized alg. outputs a key pair (pk, sk)
- **E**(pk, m): randomized alg. that takes $m \in M$ and outputs $c \in C$
- **D**(sk,c): det. alg. that takes $c \in C$ and outputs $m \in M$ or $\perp$

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:

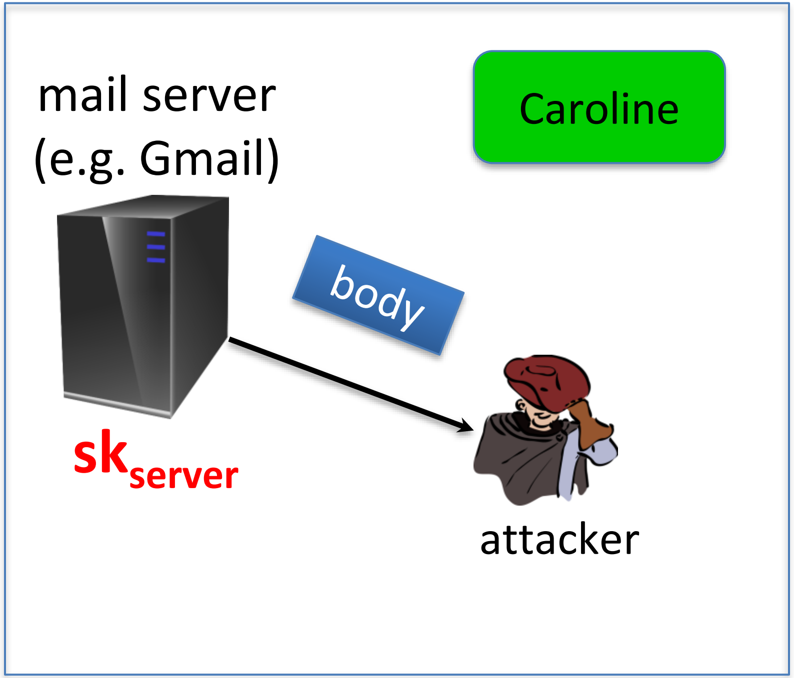
- One-time security $\Rightarrow$ many-time security (CPA)

  (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?



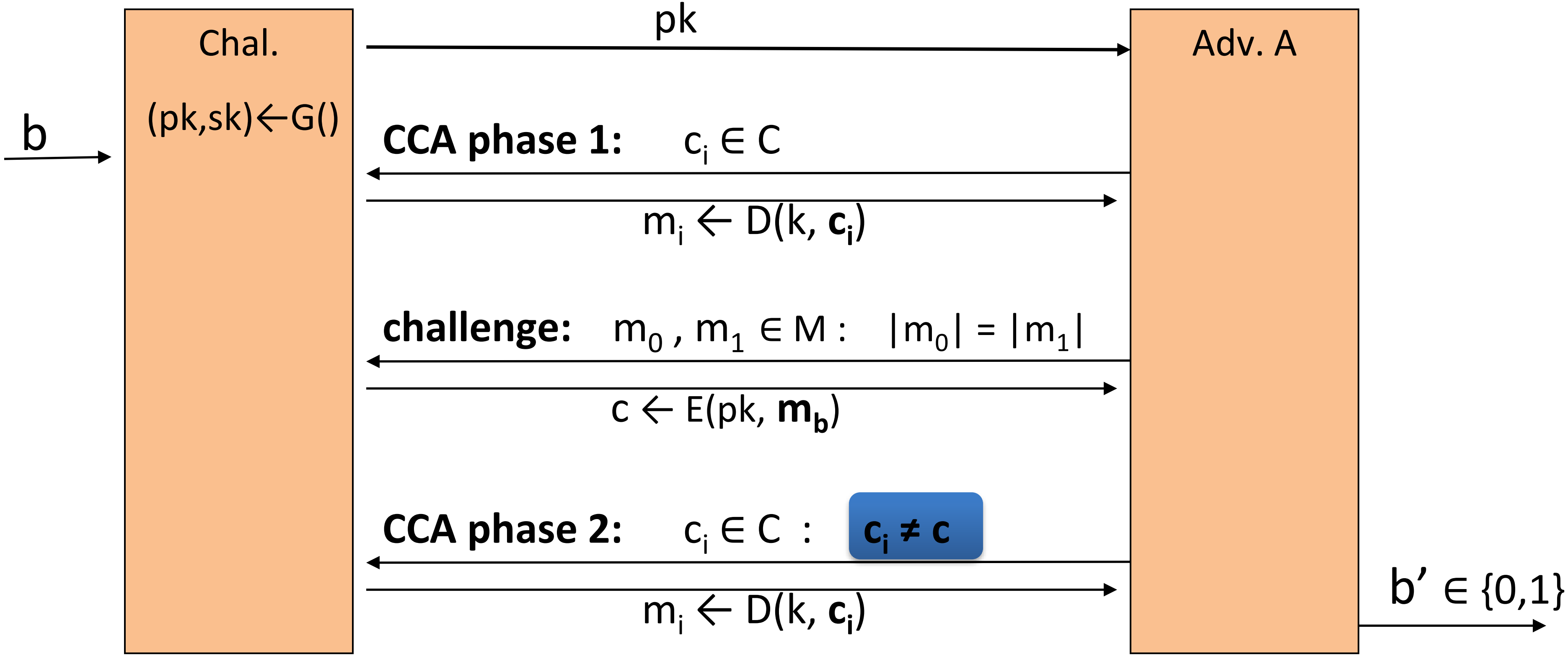
Attacker is given decryption of msgs

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):

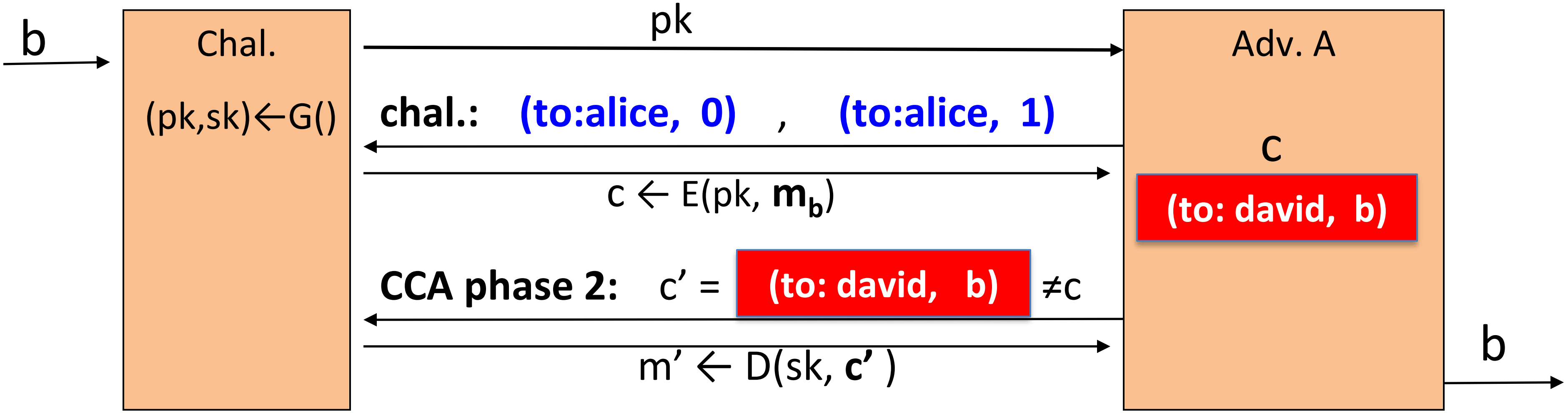


Chosen ciphertext security: definition

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

$$\mathrm{Adv}_{\mathrm{CCA}}[A,\mathbb{E}] = \left|\Pr[\mathrm{EXP}(0)=1] - \Pr[\mathrm{EXP}(1)=1]\right|$$ 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. $X \longrightarrow Y$ 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 $F(pk,\cdot)$

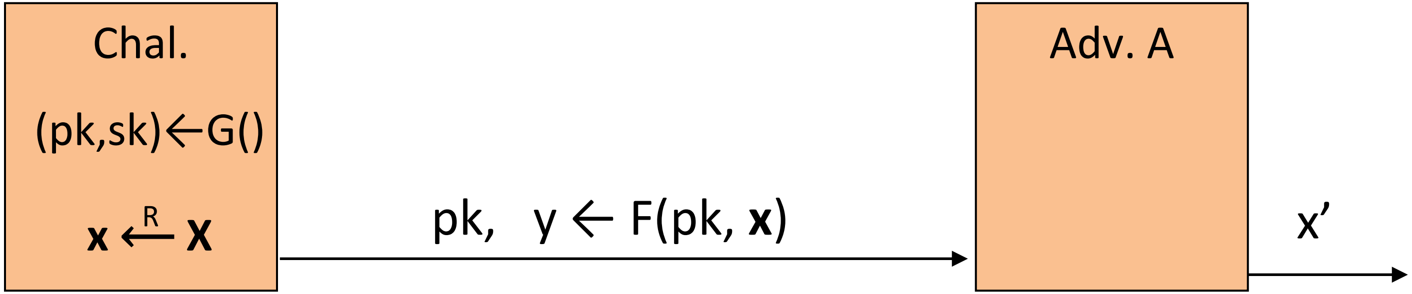
More precisely: $\forall (pk, sk)$ output by G

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

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 <u>without sk</u>

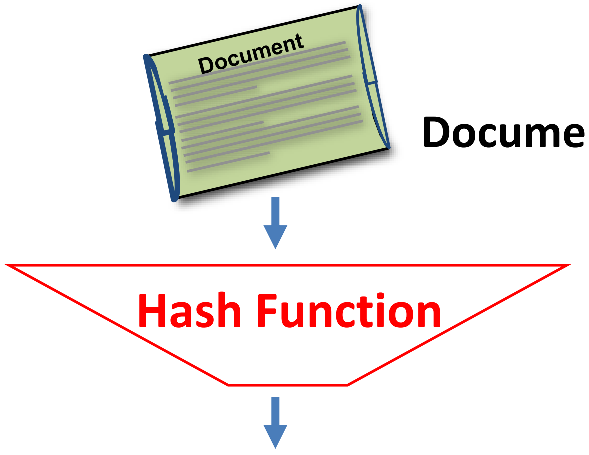


<u>Def</u>: (G, F, F^{-1}) is a secure TDF if for all efficient A:

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

Hash Functions

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



Document with arbitrary length

Hash value for the document
(fixed length, e.g. 256 bits)

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 the following 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 **m1**, **m2** such that **H(m1)= H(m2)**.

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 $X \longrightarrow Y$
- (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

E(pk, m) :

$$x \xleftarrow{R} X, \quad y \longleftarrow F(pk, x)$$
$$k \longleftarrow H(x), \; c \longleftarrow E_s(k, m)$$
output (y, c)

D(sk, (y,c)) :

$$x \longleftarrow F^{-1}(sk, y),$$
$$k \longleftarrow H(x), \; m \longleftarrow D_s(k, c)$$
output m

In pictures:

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

<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> :

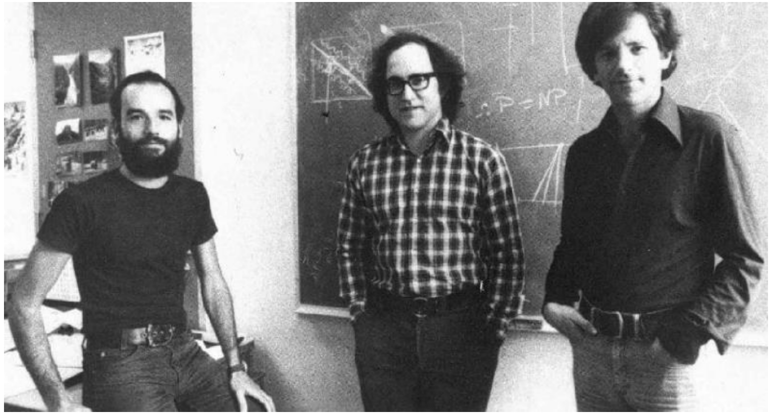
output $c \longleftarrow F(pk, m)$

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

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

Problems:

- Deterministic: cannot be semantically secure !!
- 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 ***R**on Rivest*, *Adi **S**hamir*, and *Len **A**dleman* 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 $F(pk, \cdot): X \to X$
- $F(pk, x)$: 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

Review: arithmetic mod composites

Let $N = p \cdot q$ where p,q are prime where $p,q \approx N^{1/2}$

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

Facts: $x \in Z_N$ is invertible $\Leftrightarrow$ $\gcd(x,N) = 1$

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

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

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** ≈1024 bits. Set **N=pq**.

choose integers **e , d** s.t. $\mathbf{e \cdot d = 1 \pmod{\varphi(N)}}$

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

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

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

RSA - small example

- Bob (**keys generation**):
  - chooses 2 primes: **$p=5$, $q=11$**
  - multiplies p and q: **$n = p \times q = 55$**
  - chooses a number **$e=3$** s.t. **$\gcd(e, 40) = 1$; $(40 = 55-5-11+1)$**
  - compute **$d=27$** that satisfy **$(3 \times d) \bmod 40 = 1$**

  - **Bob's public key: (3, 55)**
  - **Bob's private key: 27**

RSA - small example

- Alice (**encryption**):
  - has a message **m=13** to be sent to Bob
  - finds out **Bob's public encryption key (3, 55)**
  - calculates **c** as follows:

$$\begin{aligned} c &= m^e \bmod n \\ &= 13^3 \bmod 55 \\ &= 2197 \bmod 55 \\ &= 52 \end{aligned}$$

  - sends the ciphertext **c=52** to Bob

RSA - small example

- Bob (**decryption**):
  - receives the ciphertext **c=52** from Alice
  - uses his matching private decryption key **27** to calculate **m**:

$$m = 52^{27} \bmod 55$$
$$= 13 \text{ (Alice's message)}$$

The RSA assumption

RSA assumption: RSA is a 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 $D_s(\ H(RSA^{-1}(y))\ ,\ c) \rightarrow m$

Plain/Textbook RSA is insecure

Textbook RSA encryption:

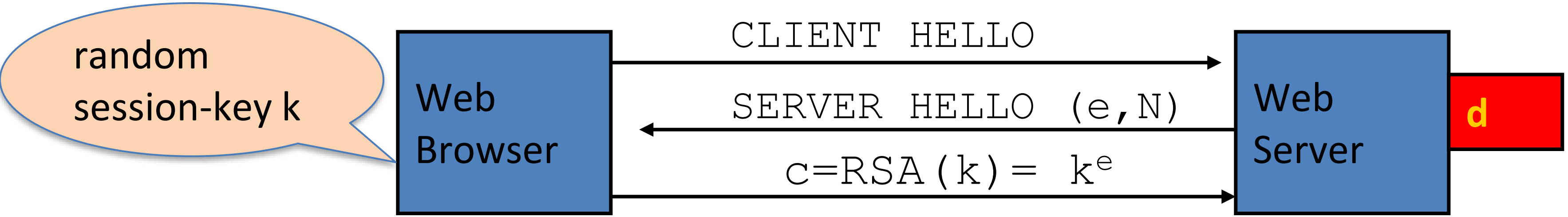
- public key: **(N,e)** Encrypt: $\mathbf{c \longleftarrow m^e}$ (in Z_N)
- secret key: **(N,d)** Decrypt: $\mathbf{c^d \longrightarrow m}$

Insecure cryptosystem !!

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

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

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

How hard is computing e'th roots modulo N ($c^{1/e}$ / $\sqrt[e]{c}$ 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** ($d \approx 2^{128}$)

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

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 (at home)

(N,e) => d and d < $N^{0.25}/3$

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

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

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

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

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

$e \cdot d = 1 \pmod{k} \Rightarrow \gcd(d,k)=1 \Rightarrow$ can find d 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: **$e=3$** ($\gcd(e, \varphi(N)) = 1$) (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:

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.

An Example Fault Attack on RSA (CRT)

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

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

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

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

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.]

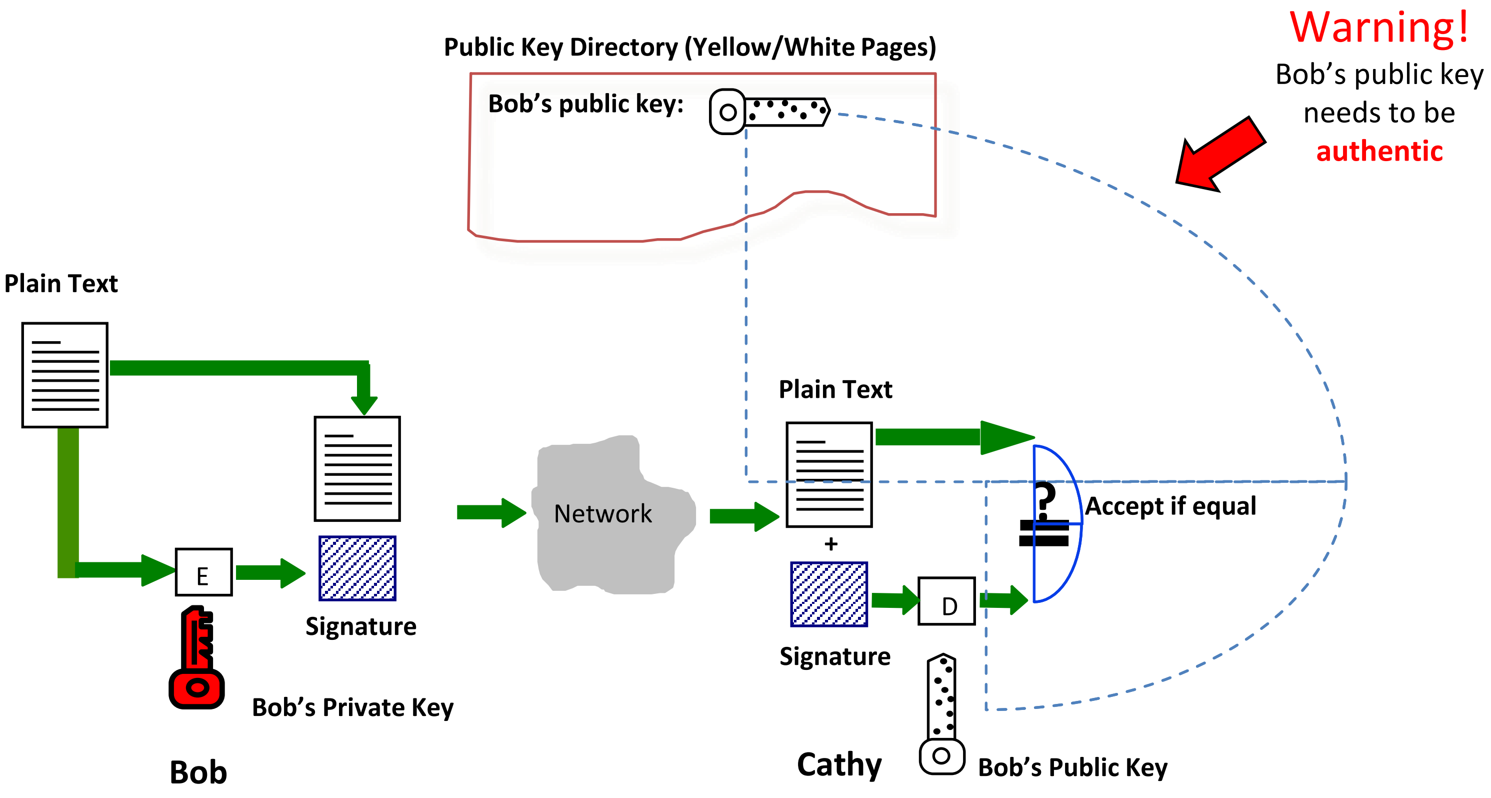
Experiment: factors 0.4% of public HTTPS keys !!

Lesson:

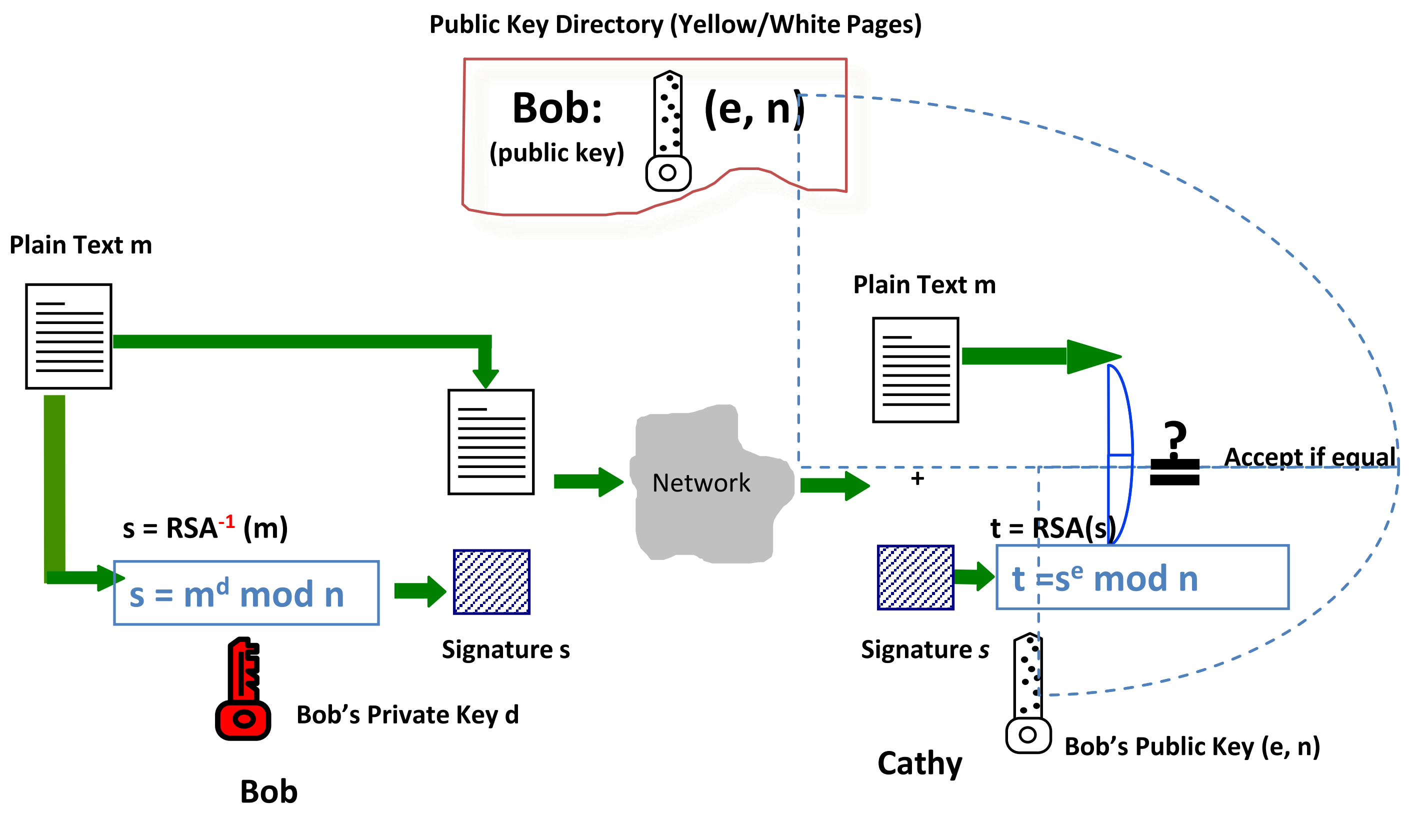
- Make sure random number generator is properly seeded when generating keys

Digital Signatures

Digital Signature



Digital Signature (based on RSA)



RSA Signature - small example

- Bob (**keys generation**):
  - chooses 2 primes: **$p=5$, $q=11$**
  - multiplies p and q: **$n = p \times q = 55$**
  - chooses a number **$e=3$** s.t. **$\gcd(e, 40) = 1$**
  - compute **$d=27$** that satisfy **$(3 \times d) \bmod 40 = 1$**

  - **Bob's public key: (3, 55)**
  - **Bob's private key: 27**

RSA Signature - small example

- Bob:
  - has a document **m=19** to sign:
  - uses **his private key d=27** to **calculate the digital signature** of **m=19**:

$$
\begin{aligned}
s &= m^d \bmod n \\
&= 19^{27} \bmod 55 \\
&= 24
\end{aligned}
$$

  - **appends 24 to 19**.

    Now **(m, s) = (19, 24**) indicates that the **doc is 19**, and **Bob's signature on the doc is 24**.

RSA Signature - small example

- Cathy, a verifier:
  - **receives** a pair **(m,s)=(19, 24)**
  - looks up the phone book and **finds out Bob's public key** (e, n)=(3, 55)
  - **calculates** $\mathbf{t} = s^e \bmod n$

    $= 24^3 \bmod 55$

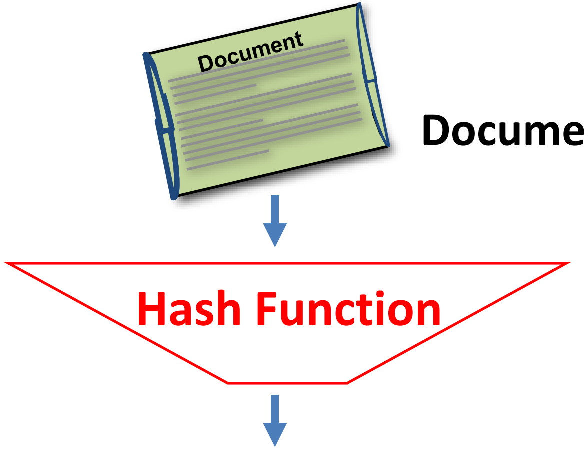
    $= 19$
  - **checks whether t=m**
  - **confirms that (19,24) is a genuinely signed document of Bob if t=m.**

How about Long Documents ?

- In the previous example, a document has to be an integer in [0,...,n)
- To sign a very long document, we need a so called one-way **hash algorithm**
- **Instead of signing directly on a doc,**
  - we **hash the doc first**,
  - and **sign the hashed data** which is normally short.

Hash Functions

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

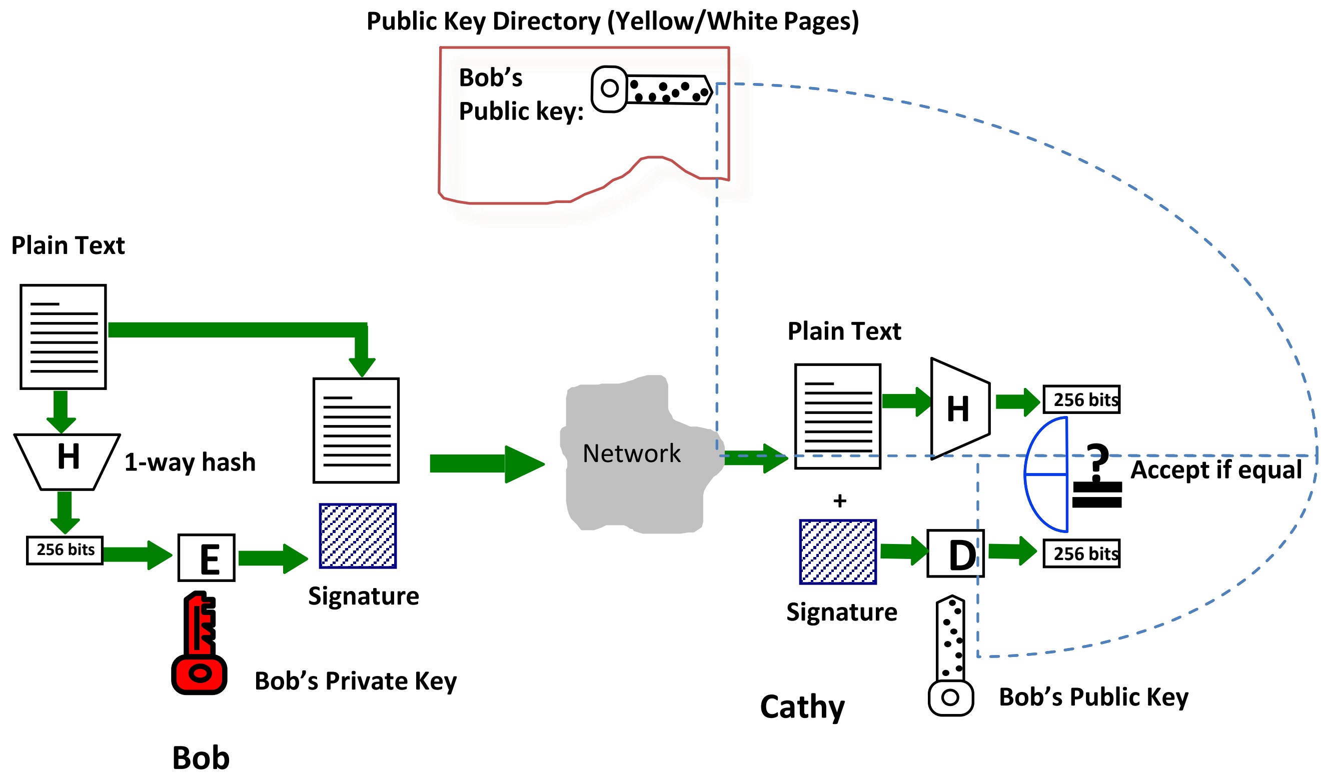


Document with arbitrary length

Hash value for the document
(fixed length, e.g. 256 bit)

Rather than signing the original document, we **sign its hash value**

Digital Signature (for long docs)



Why Digital Signature ?

- Unforgeable
  - takes 1 billion years to forge !
- Un-deniable by the signatory
- Universally verifiable
- Differs from doc to doc

Digital Signature - summary

- Three (3) steps are involved in digital signature
  - Setting up public and private keys
  - Signing a document
  - Verifying a signature

Setting up Public & Private Keys

- Bob does the following
  - prepares a pair of public and private keys
  - Publishes his public key in the public key file (such as an on-line phone book)
  - Keeps the private key to himself
- Note:
  - Setting up needs only to be done **once** !

Signing a Document

- Once setting up is completed, Bob can sign a document (such as a contract, a cheque, a certificate, ...) using the private key
- The pair of document & signature is a proof that Bob has signed the document.

Verifying a Signature

- Any party, say Cathy, can verify the pair of document and signature, by using Bob’s public key in the public key file.
- Important !
  - Cathy does NOT have to have public or private key !