Cryptography (and Information Security)
6CCS3CIS / 7CCSMCIS

Prof. Luca Viganò

Department of Informatics
King's College London, UK

First term 2019/20

Lecture 3

Outline

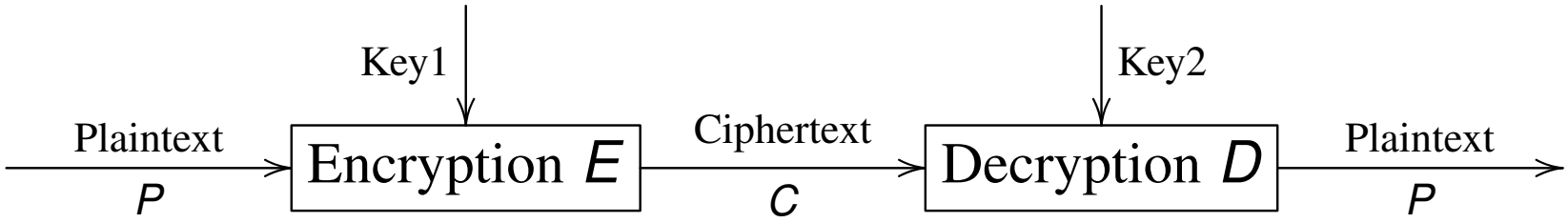
Roadmap

The section labeled with (*) contains additional material that will help understanding but will not be subject of exam questions.

Table of contents I

Table of contents II

Recall: General cryptographic schema



where $E(\text{Key1}, P) = C$ and $D(\text{Key2}, C) = P$.

- **Symmetric algorithms**:
  - Key1 = Key2, or are easily derived from each other.
- **Asymmetric (or public key) algorithms**:
  - Different keys, which cannot be derived from each other.
  - Public key can be published without compromising private key.
- Encryption and decryption should be easy, if keys are known.
- **Security depends only on secrecy of the key, not on the algorithm.**

Key distribution: problem and alternatives

Key distribution problem

- Symmetric schemes (e.g., DES and AES) require both principals to share a common secret key.
- The issue is how to securely distribute this key.
- Often secure system failure is due to a break in the key distribution scheme.

A and B have various key distribution alternatives

- A can select key and physically deliver to B.
- If A and B have communicated previously, they can use previous key to encrypt a new key.
- Third party can select and deliver key to A and B.
- If A and B have secure communications with a third party C, C can relay key between A and B.

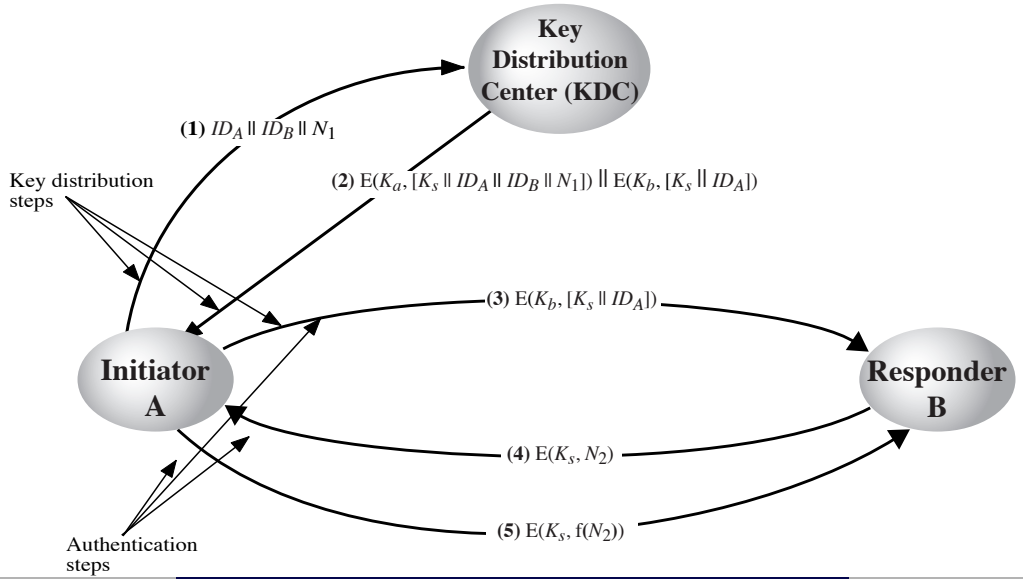
Typically a **hierarchy of keys** is used

Session key

- Used to encrypt data between users for 1 session, then discarded.

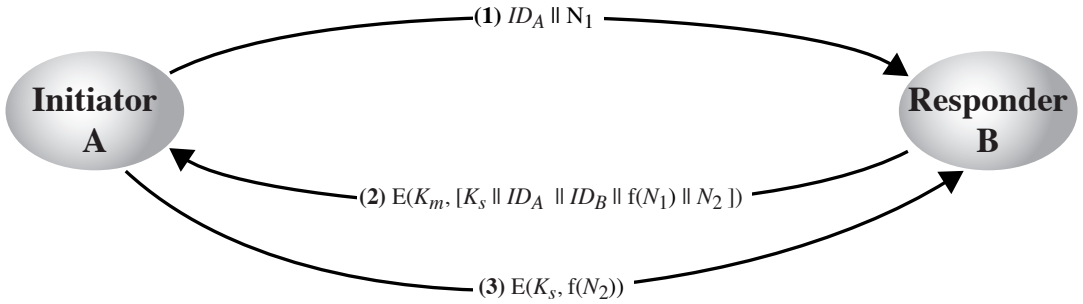
Master key

- Used to encrypt session keys.
- Shared by user(s) and **key distribution center KDC** (and, e.g., distributed as in figure).



Key distribution issues

- Hierarchies of KDC's required for large networks, but requires mutual trust between agents involved.
- Session key lifetimes should be limited for greater security.
- Use of automatic key distribution on behalf of users, but then must trust system.
- Use of **decentralized key distribution**.



- Controlling key usage.

Table of contents I

Table of contents II

Motivation

Public key cryptography

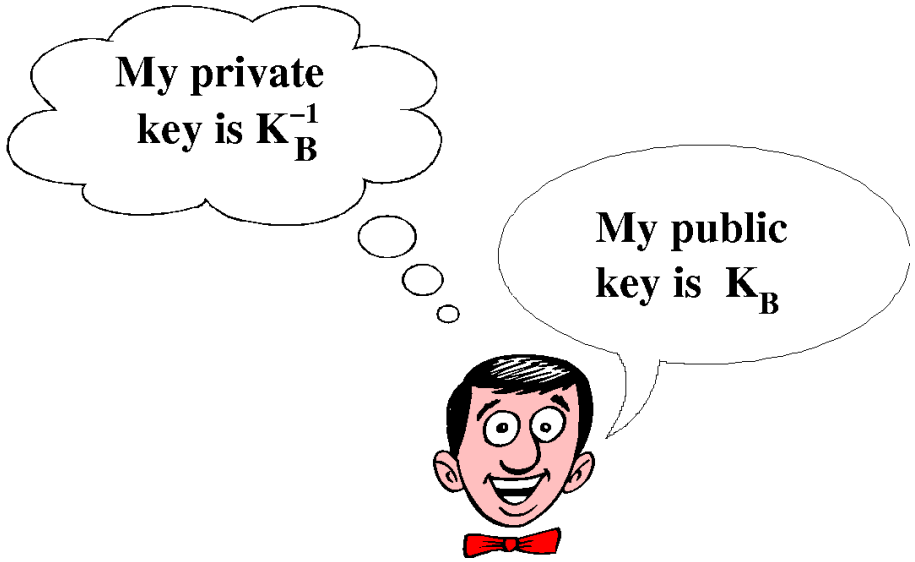
Born in May 1975, the child of two problems: the key distribution problem and the problem of signatures. The discovery consisted not of a solution, but of the recognition that the two problems, each of which seemed unsolvable by definition, could be solved at all and that the solutions to both came in one package.

Whitfield Diffie, *The first-ten years of public key cryptography*, 1988

Let's consider to what extent these problems are "solved".

Public-key cryptography

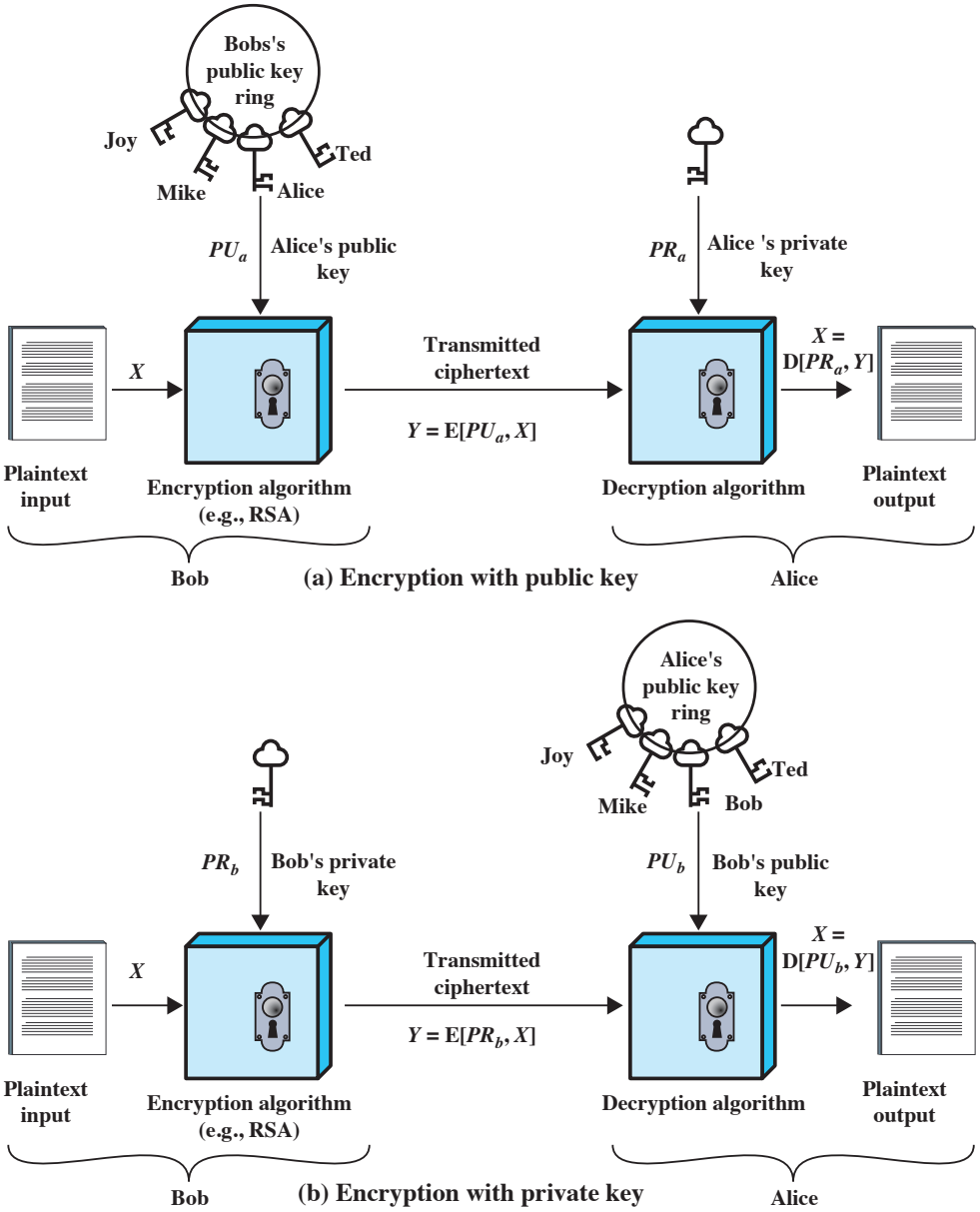
- Let $\{E_e \mid e \in \mathcal{K}\}$ and $\{D_d \mid d \in \mathcal{K}\}$ form an encryption scheme.
- Consider transformation pairs (E_e, D_d) where knowing E_e it is infeasible, given $c \in \mathcal{C}$, to find an $m \in \mathcal{M}$ such that $E_e(m) = c$.
- This implies it is **infeasible to determine** d **from** e.
- Hence, E_e constitutes a trap-door one-way function with trapdoor d (as explained in more detail later).
- Called **public key** as e can be public information:



Public-key cryptography: encryption/authentication

Alice's public key written equivalently as K_A, K_a, KA, Ka, PU_A, PU_a, $PU(A)$, $PU(a)$,...

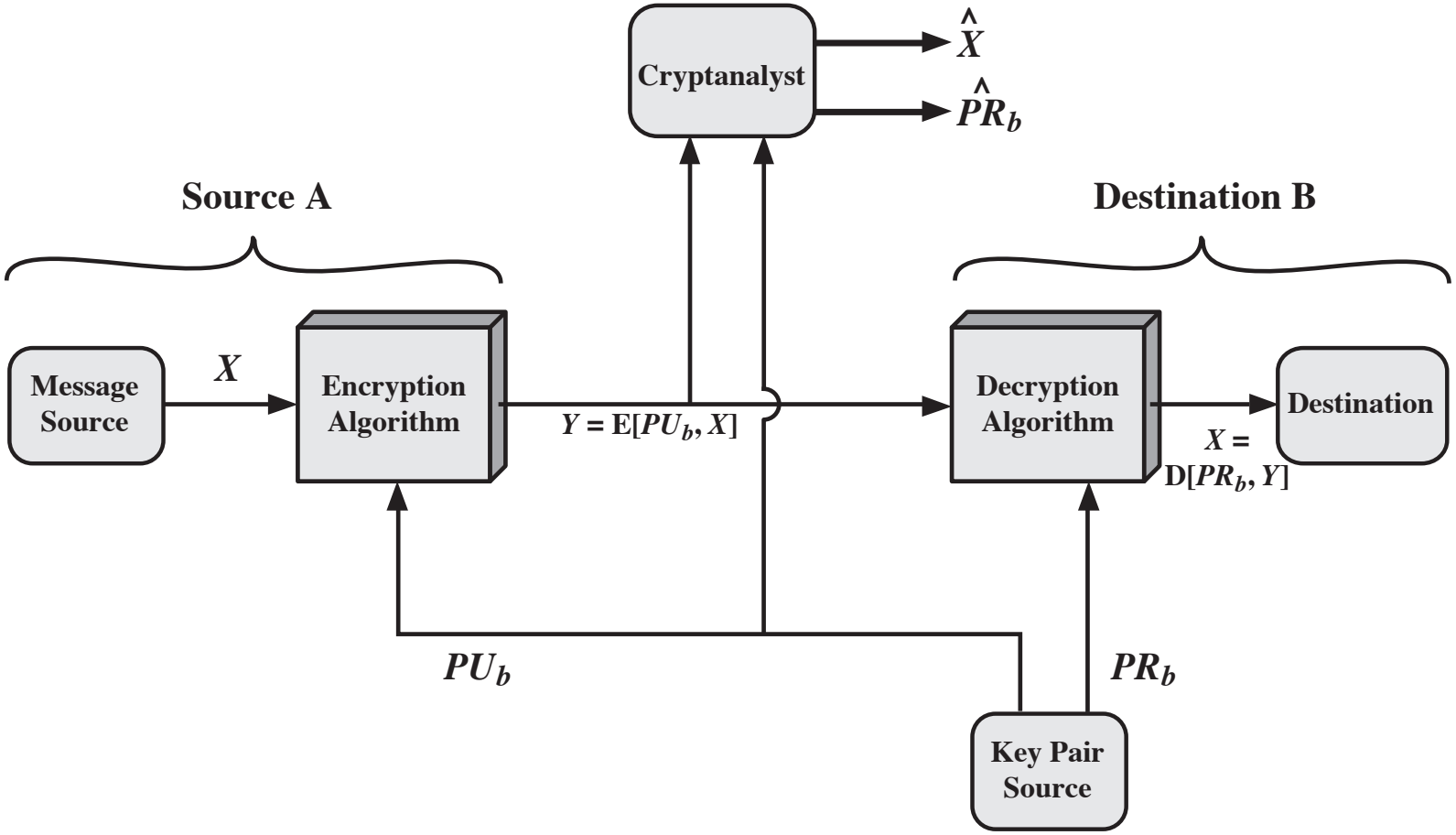
Alice's private key written equivalently as K_A^{-1}, K_a^{-1}, KA^{-1}, Ka^{-1}, PR_A, PR_a, $PR(A)$, $PR(a)$, ...



Conventional (symmetric) encryption vs. public-key (asymmetric) encryption

Conventional Encryption	Public-Key Encryption
Needed to Work: 1. The same algorithm with the same key is used for encryption and decryption. 2. The sender and receiver must share the algorithm and the key. *Needed for Security:* 1. The key must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	*Needed to Work:* 1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption. 2. The sender and receiver must each have one of the matched pair of keys (not the same one). *Needed for Security:* 1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

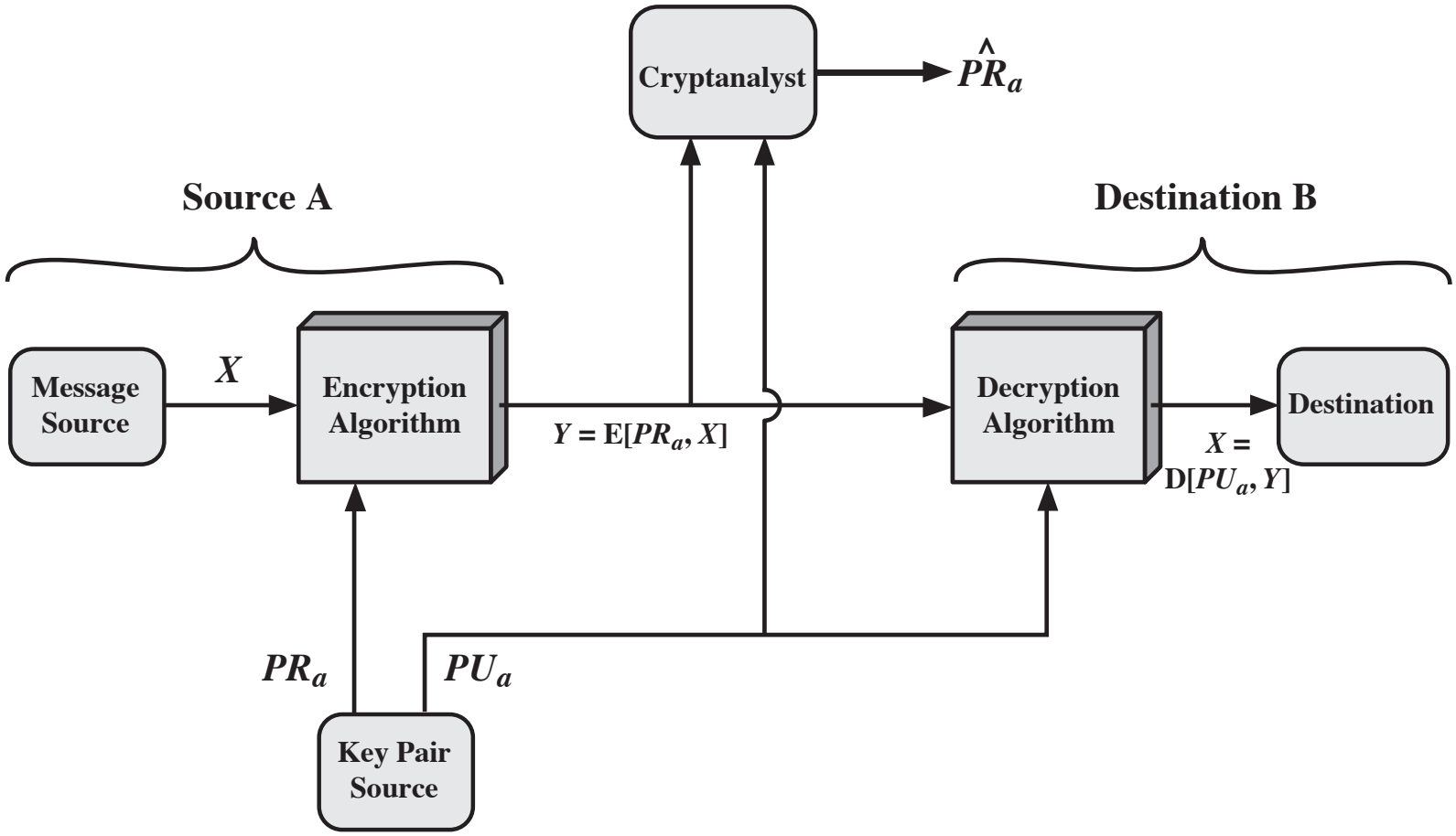
Public-key cryptosystem: secrecy (confidentiality)



Secrecy (confidentiality)

- X is a secret intended for B.
- Only B, who possesses PR_b, can decrypt $Y = E(PU_B, X)$.

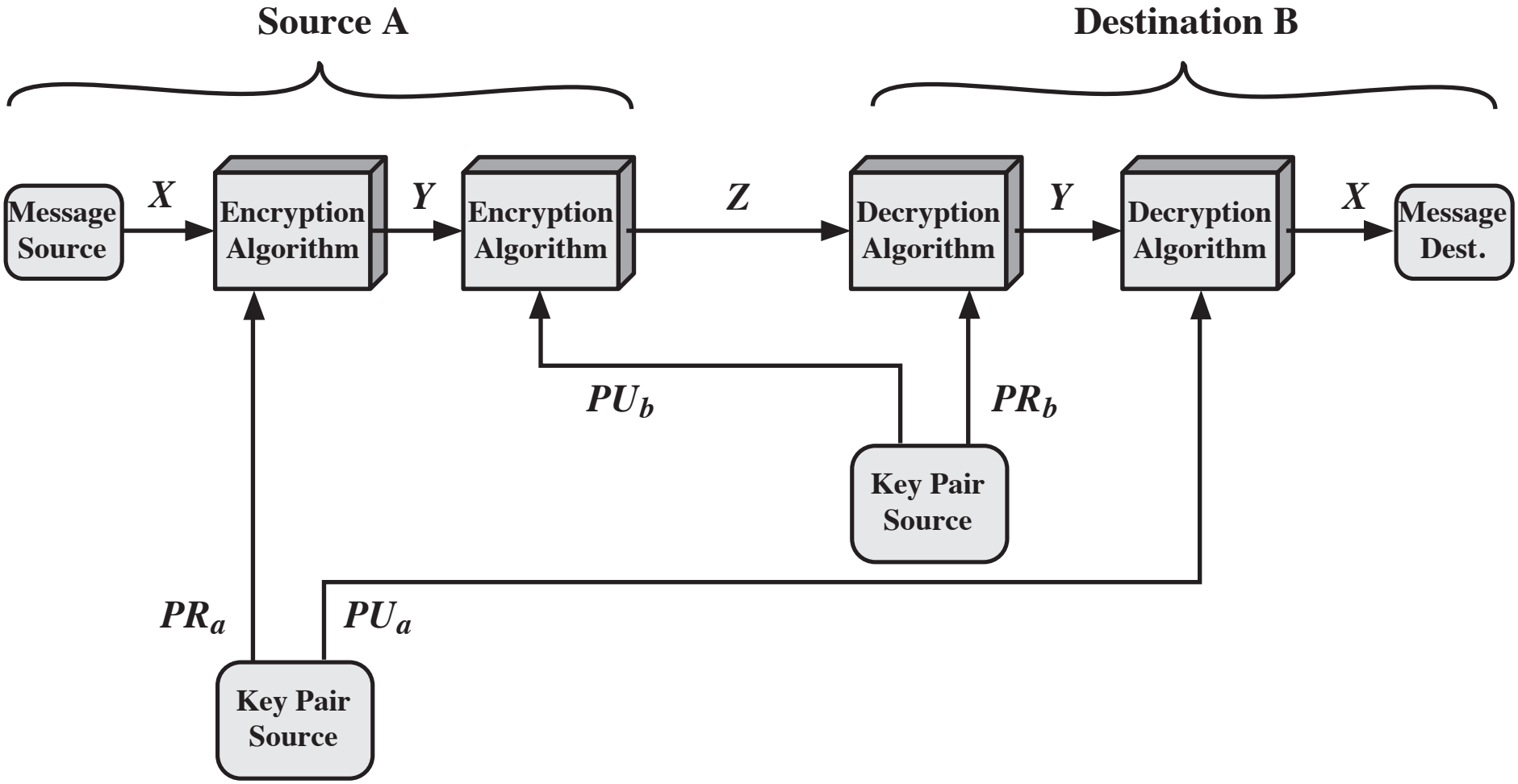
Public-key cryptosystem: authentication



Authentication

- Only A, who possesses PR_a, can have generated $Y = E(PR_A, X)$.
- Note that everybody can decrypt Y (and read X) as PU_a is public.

Public-key cryptosystem: secrecy and authentication



$$Z = E(PU_b, E(PR_a, X)) \quad \text{and} \quad X = D(PU_a, D(PR_b, Z))$$

Three applications of public-key cryptosystems

Broadly speaking, we can classify the use of public-key cryptosystems into three categories:

Encryption/decryption

- Sender encrypts a message with recipient's public key.

Digital signature

- Sender "signs" a message with its private key.
- Signing is achieved by a cryptographic algorithm applied to message or to a small block of data that is a function of message (typically, a cryptographic hash of the message).

Key exchange

- Two sides cooperate to exchange a session key.
- Different approaches are possible, involving private key(s) of one or both parties.

Requirements for public-key cryptography

1. It is computationally easy for any principal B to generate a pair (public key PU_b, private key PR_b).
2. It is computationally easy for sender A, knowing PU_b and M, to generate
$$C = E(PU_b, M)\,.$$
3. It is computationally easy for receiver B to decrypt C using PR_b to recover M:
$$M = D(PR_b, C) = D(PR_b, E(PU_b, M))\,.$$
4. It is computationally infeasible for an adversary
   - knowing PU_b to determine PR_b,
   - knowing PU_b and C to recover M.
5. (Useful, but not always necessary) The two keys can be applied in either order:
$$M = D(PU_b, E(PR_b, M)) = D(PR_b, E(PU_b, M))\,.$$

Requirements for Public-Key Cryptography (cont.)

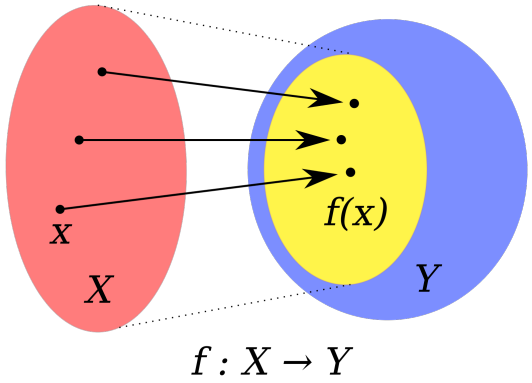
- These are difficult requirements.
- As a matter of fact only a few algorithms enjoying the above requirements have received widespread acceptance so far, e.g.,

Algorithm	**Encryption/Decryption**	**Digital signature**	**Key exchange**
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

- In this module, we will focus on RSA and Diffie-Hellman.

Some terminology about functions

- $f : X \to Y$ is a function from the set X (f's **domain**) to the set Y (f's **codomain**).



$f : X \to Y$

- **Image** of f is subset of f's codomain that is output of f on a subset of its domain (image is the smaller oval inside Y).
  - Image of $x \in X$ under f is $f(x) = y$ (value of f when applied to x).
  - Image of a subset $A \subseteq X$ under f is subset $f[A] \subseteq Y$ defined by

$$f[A] = \{y \in Y \mid y = f(x) \text{ for some } x \in A\}$$

  - Image $f[X]$ of entire domain X of f is called simply the image of f.
- **Inverse image** (or **preimage**) of a particular subset $B \subseteq Y$ of the codomain of a function f is the set of all elements of the domain that map to the members of B.

$$f^{-1}[B] = \{x \in X \mid f(x) \in B\}$$

One-way function

One-way function

A function $f : X \to Y$ is a **one-way function**, if f is "easy" to compute for all $x \in X$, but f^{-1} is "hard" (or "infeasible") to compute.

- **Easy**: generally, defined to mean a problem that can be solved in polynomial time as a function of input length.
  - If input length is n bits, then time to compute function is proportional to n^a, where a is a fixed constant.
- **Infeasible**: effort to solve problem grows faster than polynomial time as a function of input size.
  - Time to compute function proportional to 2^n for input length n bits.
    - Difficult to determine if a particular algorithm exhibits this complexity.
    - Computational complexity traditionally focuses on worst-case or average-case complexity of an algorithm, but cryptography requires that it be infeasible to invert a function for virtually all inputs.

One-way function: examples

- **Square root.**
  - If you know $x = 512$, $f(x) = x^2 = 512^2 = 262144$ is easy to compute.
  - If you know $f(x) = 262144$, $x = \sqrt{x^2} = \sqrt{262144}$ is difficult to compute.
- **Modular cube roots.**
  - Select primes $p = 48611$ and $q = 53993$.
  - Let $n = p \times q = 2624653723$ and $X = \{1, 2, \ldots, n-1\}$.
  - Define $f : X \to \mathbb{N}$ by $f(x) = x^3 \bmod n$.
  - Example: $f(2489991) = 1981394214$.
  - Computing f is easy.
  - Inverting f is hard: find x which is cubed and yields remainder!

This is useful because:

- Encryption is (very) easy whereas decryption is (very) difficult.
- The idea is: "$f(x)$ acts as a public key and x as a private key".

Trapdoor one-way function

- A trapdoor one-way function is easy to calculate in one direction and infeasible to calculate in the other direction unless certain additional information is known.
  - With additional info, inverse can be calculated in polynomial time.

Trapdoor one-way function

A **trapdoor one-way function** is a one-way function $f_k : X \to Y$ where, given extra information k (the trapdoor information) it is feasible to find, for $y \in Image(f)$, an $x \in X$ where $f_k(x) = y$.

- Hence, a *trapdoor one-way function is a family of invertible functions* f_k such that computing

$Y = f_k(X)$	is easy if k and X are known
$X = f_k^{-1}(Y)$	is easy if k and Y are known
$X = f_k^{-1}(Y)$	is infeasible if Y is known but k is not known

- **Example:** Computing modular cube roots is easy when p and q are known (basic number theory).

Public-key cryptanalysis

- **Brute-force attacks** are possible.
  Countermeasure: use large keys!
  - But tradeoff is necessary as complexity of encryption/decryption may not scale linearly with the length of the key.
  - In practice: public-key encryption is confined to *key management* and *digital signature*.

- **Computing private key from public key.**
  - *No proof that this attack is infeasible!* (Not even for RSA)

- **Probable-message attack.**
  - Suppose a short message M (e.g., a 56-bit DES key) is sent encrypted with PU_a, i.e., $C = E(PU_a, M)$.
  - The attacker computes all $Y_i = E(PU_a, X_i)$ for all possible plaintexts X_i for $i = 1, \ldots, 2^{56}$ and stops as soon as $Y_i = C$ concluding that $M = X_i$ (the message sent).
  - Solution: append some random bits to M before encryption.

Table of contents I

Table of contents II

Table of contents I

Table of contents II

Prime factorization

- Numbers: **naturals** $\mathbb{N} = \{0, 1, 2, \ldots\}$, **integers** $\mathbb{Z} = \{0, 1, -1, \ldots\}$, **primes** $\mathcal{P} = \{2, 3, 5, 7, ...\}$.
- To **factor** a number a is to write it as a product of other numbers, e.g., $a = b \times c \times d$.
- Multiplying numbers is easy, factoring numbers appears hard. We cannot factor most numbers with more than 1024 bits.
- The **prime factorization** of a number a amounts to writing it as a product of powers of primes:

$$a = \prod_{p \in \mathcal{P}} p^{a_p} = 2^{a_2} \times 3^{a_3} \times 5^{a_5} \times 7^{a_7} \times 11^{a_{11}} \times \ldots \quad \text{where } a_p \in \mathbb{N}$$

For any particular value of a, most of the exponents a_p will be 0, e.g.,

$$\begin{aligned} 91 &= 7 \times 13 \\ 3600 &= 2^4 \times 3^2 \times 5^2 \\ 11011 &= 7 \times 11^2 \times 13 \end{aligned}$$

Divisors

$a \neq 0$ **divides** b (written $a \mid b$) if there is an m such that $m \times a = b$.

- Examples: $3 \mid 6$ and $7 \mid 21$.

a **does not divide** b (written $a \nmid b$) if there is no m such that $m \times a = b$.

- Examples: $3 \nmid 7$, $3 \nmid 10$ and $7 \nmid 22$.

Relatively prime numbers & greatest common divisor

Two natural numbers a, b are **relatively prime** if they have no common divisors/factors apart from 1, i.e., if their **greatest common divisor** gcd is equal to 1

$$\gcd(a, b) = 1\,.$$

- For example, 8 and 15 are relatively prime since
  - factors of 8 are 1, 2, 4, 8,
  - factors of 15 are 1, 3, 5, 15,
  - and 1 is the only common factor.
- Conversely, we can determine the greatest common divisor by comparing their prime factorizations and using least powers, e.g.
  - $150 = 2^1 \times 3^1 \times 5^2$ and $18 = 2^1 \times 3^2$, thus $\gcd(18, 150) = 2^1 \times 3^1 \times 5^0 = 6$.
  - $60 = 2^2 \times 3 \times 5$ and $14 = 2 \times 7$, thus $\gcd(60, 14) = 2$.

Table of contents I

Table of contents II

Greatest common divisor and Euclid's algorithm

- gcd can be computed quickly using **Euclid's algorithm**.

$$\begin{array}{lcccl} \gcd(60,14) & : & 60 & = & (4 \times 14) + 4 \\ \gcd(14,4) & : & 14 & = & (3 \times 4) + 2 \\ \gcd(4,2) & : & 4 & = & 2 \times 2 \end{array}$$

- **Extended Euclid's algorithm** computes $x, y \in \mathbb{Z}$ such that

$$\gcd(a,b) = (x \times a) + (y \times b)$$

Here $2 = 14 - 3 \times (60 - (4 \times 14)) = (-3 \times 60) + (13 \times 14)$

Euclid's Algorithm

Euclid's algorithm is based on the theorem

$\gcd(a, b) = \gcd(b, a \bmod b)$ for any nonnegative integer a and any positive integer b.

For example:

- $\gcd(55, 22) = \gcd(22, 55 \bmod 22) = \gcd(22, 11) = 11$.

Euclid's algorithm

```
Euclid(a, b)
1 if b = 0
2 then return a
3 else return Euclid(b, a mod b)
```

For example:

- Euclid(30, 21) = Euclid(21, 9) = Euclid(9, 3) = Euclid(3, 0) = 3.

Extended Euclid's Algorithm

Extend Euclid's algorithm to compute integer coefficients x, y such that

$$d = \gcd(a, b) = (a \times x) + (b \times y)$$

Extended Euclid's algorithm

```
Extended-Euclid(a, b)
1 if b = 0
2 then
3    return (a, 1, 0)
4 else
5    (d', x', y') ← Extended-Euclid(b, a mod b)
6    (d, x, y) ← (d', y', x' − (⌊a/b⌋ × y'))
7    return (d, x, y)
```

where $q = \lfloor a/b \rfloor$ is the **quotient of the division** (for $a = (q \times b) + r$).

Note: the d here is the greatest common **d**ivisor, not to be confused with the d that is (part of) an RSA private key (discussed later on).

Extended Euclid's Algorithm: example

Extended-Euclid$(99, 78) = 3 = (99 \times (-11)) + (78 \times 14)$

Extended-Euclid(a, b)
1 **if** $b = 0$
2 **then**
3 **return** $(a, 1, 0)$
4 **else**
5 $(d', x', y') \leftarrow$ Extended-Euclid$(b,\ a \bmod b)$
6 $(d, x, y) \leftarrow (d', y', x' - (\lfloor a/b \rfloor \times y'))$
7 **return** (d, x, y)

a	b	$\lfloor a/b \rfloor$	d	x	y
99	78	1	3	-11	14
78	21	3	3	3	-11
21	15	1	3	-2	3
15	6	2	3	1	-2
6	3	2	3	0	1
3	0	–	3	1	0

Each line shows one level of the recursion.

Table of contents I

Table of contents II

Modular arithmetics

Remainder

- $\forall a\, n.\ \exists q\, r.\ (a = (q \times n) + r)$ where $0 \leq r < n$.

  Here r is the **remainder**, which we write as

$$r = a \bmod n\,.$$

Congruent modulo

- $a, b \in \mathbb{Z}$ are **congruent modulo** n, if $a \bmod n = b \bmod n$.

  We write this as

$$a =_n b\,.$$

Modulo operator has following properties (of congruences)

- Reflexivity: $a =_n a$.
- Symmetry: If $a =_n b$ then $b =_n a$.
- Transitivity: If ($a =_n b$ and $b =_n c$) then $a =_n c$.

Other properties of the modulo operator

$(a \bullet b) =_n (a \bmod n) \bullet (b \bmod n)$ for $\bullet \in \{+, -, \times\}$

i.e., $(a \bullet b) \bmod n = [(a \bmod n) \bullet (b \bmod n)] \bmod n$

Example:

$$\begin{aligned} 2 &= (5 \times 6) \bmod 4 \\ &= [(5 \bmod 4) \times (6 \bmod 4)] \bmod 4 \\ &= (1 \times 2) \bmod 4 = 2 \bmod 4 = 2 \end{aligned}$$

If $a \times b =_n a \times c$ and a relatively prime to n, then $b =_n c$.

Example:

- $8 \times 4 =_3 8 \times 1$.
- 8 is relatively prime to 3.
- So: $4 =_3 1$.

If $a_1 =_n b_1$ and $a_2 =_n b_2$, then

$$(a_1 + a_2) =_n (b_1 + b_2) \quad \text{and} \quad (a_1 \times a_2) =_n (b_1 \times b_2)$$

- This can also be expressed as

$$[(a_1 \bmod n) + (a_2 \bmod n)] \bmod n = (a_1 + a_2) \bmod n$$

and

$$[(a_1 \bmod n) \times (a_2 \bmod n)] \bmod n = (a_1 \times a_2) \bmod n$$

- Example: Let $r_a = a \bmod n$ and $r_b = b \bmod n$.
Then, there are integers j and k such that

$$a = r_a + jn \text{ and } b = r_b + kn$$

and we can proceed as follows:

$$\begin{aligned}(a+b) \bmod n &= (r_a + jn + r_b + kn) \bmod n \\ &= (r_a + r_b + (j+k)n) \bmod n \\ &= (r_a + r_b) \bmod n \\ &= [(a \bmod n) + (b \bmod n)] \bmod n\end{aligned}$$

- $a = q \times n + r \quad$ with $q = \lfloor a/n \rfloor$ and $0 \leq r < n$ and $r = a \bmod b$
- For any integer a, we can rewrite this as follows:
  $a = \lfloor a/n \rfloor \times n + (a \bmod n)$

Then, for example:

- $11 \bmod 7 = 4$
- $-11 \bmod 7 = -4$ ($= 3$ when reasoning modulo 7)
- $73 =_{23} 4$
- $21 =_{10} -9$
- $147 =_{220} -73$

- If $a =_n 0$ then $n \mid a$
- $a =_n b$ if $n \mid (a - b)$

- To demonstrate the last point, if $n \mid (a - b)$, then $(a - b) = k \times n$ for some k.

  So we can write $a = b + (k \times n)$.

  Therefore, (a mod n) = (remainder when $b + (k \times n)$ is divided by n) = (remainder when b is divided by n) = (b mod n).
- Then, for example:
  - $23 =_5 8$ because $23 - 8 = 15 = 5 \times 3$
  - $-11 =_8 5$ because $-11 - 5 = -16 = 8 \times (-2)$
  - $81 =_{27} 0$ because $81 - 0 = 81 = 27 \times 3$

Modular arithmetics: two theorems

Theorem

Suppose that $a, b \in \mathbb{Z}$ are relatively prime. There is a $c \in \mathbb{Z}$ satisfying $(b \times c) \bmod a = 1$, i.e., we can compute $b^{-1} \bmod a$.

Proof: From Extended Euclidean Algorithm, there exist $x, y \in \mathbb{Z}$ where

$$1 = (a \times x) + (b \times y)$$

Since $a \mid (a \times x)$, we have $(b \times y) \bmod a = 1$.
Assertion follows with $c = y$.

Fermat's little theorem

For a and n relatively prime and n prime

$$a^{n-1} =_n 1$$

Example: $4^6 \bmod 7 = (16 \times 16 \times 16) \bmod 7 = (2 \times 2 \times 2) \bmod 7 = 1$.

Proof of Fermat's Little Theorem

- Consider the set $\{1, 2, \ldots, n-1\}$ of positive integers less than n.
- Multiply each element by a, modulo n, to get the set $X = \{a \bmod n, 2a \bmod n, \ldots, (n-1)a \bmod n\}$.
- None of the element of X is equal to 0 because n does not divide a.
- Furthermore, no two of the integers in X are equal.
  - To see this, assume that $x \times a =_n y \times a$ for $1 \leq x < y \leq n-1$.
  - Because a and n are relatively prime, we can eliminate a from both sides and obtain $x =_n y$, which is impossible as we assumed that x and y are both positive integers less than n, with $x < y$.
- Therefore, we know that the $(n-1)$ elements of X are all positive integers with no two elements equal.
- Hence, X consists of the set of integers $\{1, 2, \ldots, n-1\}$ in some order.
- Multiplying the numbers in both sets, and taking the result mod n yields

$$\begin{aligned} a \times 2a \times \ldots \times (n-1)a &=_n [1 \times 2 \times \ldots \times (n-1)] \\ a^{n-1} \times (n-1)! &=_n (n-1)! \end{aligned}$$

- As n is prime, $(n-1)!$ is relatively prime to n; so canceling the $(n-1)!$ yields

$$a^{n-1} =_n 1$$

as desired.

Table of contents I

Table of contents II

Euler Totient Function

- When doing arithmetic modulo n.
- Complete set of **residues** is $0, \ldots, n-1$.
- **Reduced set of residues** consists of those numbers (*residues*) that are relatively prime to n.

  For instance, for $n = 10$:
  - complete set of residues is $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$,
  - reduced set of residues is $\{1, 3, 7, 9\}$.
- Number of elements in reduced set of residues is called the **Euler Totient Function** $\phi(n)$.
  - In other words, $\phi(n)$ **is the number of positive integers less than** n **which are relatively prime to** n, i.e.,
    $\phi(n)$ is the number of $a \in \{1, 2, \ldots, n-1\}$ with $\gcd(a, n) = 1$.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\phi(n)$	1	1	2	2	4	2	6	4	6	4	10	4	12	6	8

Euler's Totient Function and Euler's Theorem

Properties:

- $\phi(1) = 1$.
- $\phi(p) = p - 1$ if p is prime.
- $\phi(p \times q) = \phi(p) \times \phi(q) = (p-1) \times (q-1)$ if p and q are prime and $p \neq q$.

So that Fermat's little theorem (for a and n relatively prime and n prime) can be rewritten to

Euler's Theorem

$a^{\phi(n)} =_n 1$ for all a, n such that $\gcd(a, n) = 1$.

Examples:

- If $a = 3$ and $n = 10$, then $\phi(10) = 4$ and $3^4 = 81 =_{10} 1$
- If $a = 2$ and $n = 11$, then $\phi(11) = 10$ and $2^{10} = 1024 =_{11} 1$

Table of contents I

Table of contents II

Rivest, Shamir, Adleman: RSA Algorithm

- Named after inventors: Rivest, Shamir, Adleman, 1978.
- Published after 1976 challenge by Diffie and Hellman.
- RSA algorithm is a block cipher in which the plaintext and ciphertext are integers between 0 and $n - 1$ for some n.
  - A typical size for n is 1024 bits, or 309 decimal digits.
  - That is, n is less than 2^{1024}.
- Security comes from difficulty of factoring large numbers. Keys are functions of a pairs of large, ≥ 100 digits, prime numbers.
- Most popular public-key algorithm.
  Used in many applications, e.g., PGP, PEM, SSL, ...

The RSA Algorithm

- Let n be a number known by sender and receiver.
- Plaintext is encrypted in blocks, with each block having a binary value less than n.
  - That is, block size must be less than or equal to $\log_2(n) + 1$.

  Thus each block represents a number M such that $M < n$.

RSA encryption and decryption of M are defined as follows:

For some (properly chosen) values of e and d,

$$\begin{aligned} C &= M^e \mod n \\ M &= C^d \mod n = (M^e)^d \mod n = M^{ed} \mod n \end{aligned}$$

- It is a public-key encryption algorithm with

  public key $PU = (e, n)$ and **private key** $PR = (d, n)$.

The RSA Algorithm (cont.)

For the RSA algorithm to work, the following requirements must be met:

1. It is possible to find values of e, d, and n such that $M^{ed} \bmod n = M$ for all $M < n$.
2. It is relatively easy to calculate $M^e \bmod n$ and $C^d \bmod n$ for all values of $M < n$.
3. It is infeasible to determine d given e and n.

Correctness of RSA

We must show that there exist e, d, and n such that

$$M^{ed} \mod n = M$$

for all $M < n$ (where we write "ed" instead of $e \times d$ for brevity).
It can be shown that the above equation holds if *e and d are multiplicative inverses* $\mod \phi(n)$ (the Euler totient function), i.e.,

$$d =_{\phi(n)} e^{-1} \text{ if and only if}$$
$$de =_{\phi(n)} 1 \text{ if and only if}$$
$$ed \mod \phi(n) = 1$$

But this is true only if d (and hence e) is relatively prime to $\phi(n)$, i.e., if $\gcd(\phi(n), d) = 1$.

RSA algorithm

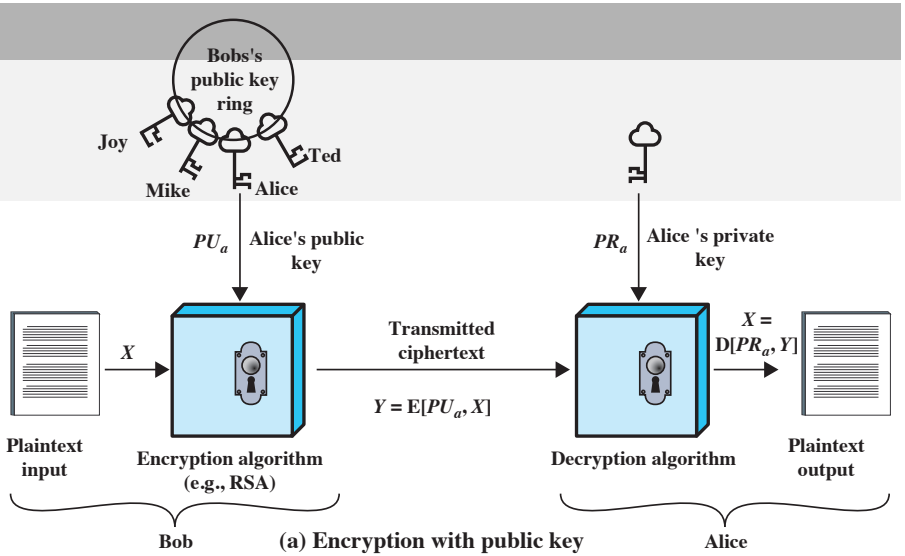
Ingredients:	p, q, two prime numbers	private, chosen
	$n = p \times q$ (or pq for short)	public, calculated
	e, with $\gcd(\phi(n), e) = 1$; $1 < e < \phi(n)$	public, chosen
	$d = e^{-1} \bmod \phi(n)$	private, calculated

- Generation of a public/private key pair:
  1. Generate two (large) distinct primes p and q.
  2. Compute $n = p \times q$ and $\phi(n) = (p-1) \times (q-1)$.
  3. Select an e, with $1 < e < \phi(n)$, relatively prime to $\phi(n)$.
  4. Compute $d = e^{-1} \bmod \phi(n)$.
  5. Publish (e, n), keep (d, n) private, discard p and q.
- Encryption with key (e, n)
  1. Break message M into blocks $M_1 M_2 \cdots$ with $M_i < n$
  2. Compute $C_i = M_i^e \bmod n$.
- Decryption with key (d, n):
  1. Compute $M_i = C_i^d \bmod n$.

RSA algorithm: example

- Generation of a public/private key pair:
  1. Generate $p = 47$, $q = 71$.
  2. Compute $n = p \times q = 3337$ and $\phi(n) = (p-1) \times (q-1) = 46 \times 70 = 3220$
  3. Choose $e = 79$ (randomly in the interval [1..3220])
  4. Compute $d = 79^{-1} \bmod 3220 = 1019$.
  5. Public key $(e, n) = (79, 3337)$, private key $(d, n) = (1019, 3337)$
- Encryption with key $(e, n) = (79, 3337)$:
  1. Break message M into blocks, e.g., 688 232 687 966 668 ...
  2. Compute $C_1 = 688^{79} \bmod 3337 = 1570$, $C_2 = ...$
- Decryption with key $(d, n) = (1019, 3337)$:
  1. Compute $M_1 = 1570^{1019} \bmod 3337 = 688$, $M_2 = ...$

RSA: another example



Alice generates a public/private key pair.
Bob encrypts using Alice's public key.
Alice decrypts using her private key.

- Keys can be generated as follows:
  - Select two prime numbers, $p = 17$ and $q = 11$.
  - Calculate $n = p \times q = 17 \times 11 = 187$.
  - Calculate $\phi(n) = (p-1) \times (q-1) = 16 \times 10 = 160$.
  - Select e such that e is relatively prime to $\phi(n) = 160$ and less than $\phi(n)$; we choose $e = 7$.
  - Determine d (e.g., using Extended Euclid's algorithm) such that $d \times e = 1 \bmod 160$ and $d < 160$.
    The correct value is $d = 23$, because $23 \times 7 = 161 = (1 \times 160) + 1$.

Note: the d here is the private key, not to be confused with the d that is the greatest common **d**ivisor in the Extended Euclid's algorithm.

Resulting keys are public key $PU_a = (e, n) = (7, 187)$ and private key $PR_a = (d, n) = (23, 187)$.

RSA algorithm: another example (cont.)

- We have $n = 187$, $\phi(n) = 160$, $e = 7$.
- d can be computed using the Extended Euclid algorithm

$$D = \gcd(A, B) = A \times x + B \times y$$

as follows:

- Since d is such that $e \times d =_{\phi(n)} 1$, we can compute

$$1 = \gcd(\phi(n), e) = \phi(n) \times x + e \times d$$

- It must be $1 < d < \phi(n)$, so when $y < 0$ we simply reason modulo $\phi(n)$.

- In this case:

$$1 = \gcd(160, 7) = 160 \times x + 7 \times y$$

A	B	$\lfloor A/B \rfloor$	D	x	y
160	7	22	1	-1	23
7	6	1	1	1	-1
6	1	6	1	0	1
1	0	–	1	1	0

That is, $1 = \gcd(160, 7) = 160 \times (-1) + 7 \times 23$. Check: $7 \times 23 =_{160} 1$.

So, we can pick $d = y = 23$.

RSA algorithm: another example (cont.)

- We have $n = 187$, $\phi(n) = 160$.
- Note that if we had picked $e = 23$, then $d = 7$.
  - Since d is such that $e \times d =_{\phi(n)} 1$, we can compute

$$1 = \gcd(\phi(n), e) = \phi(n) \times x + e \times d$$

- In this case:

$$1 = \gcd(160, 23) = 160 \times x + 23 \times y$$

A	B	$\lfloor A/B \rfloor$	D	x	y
160	23	6	1	-1	7
23	22	1	1	1	-1
22	1	22	1	0	1
1	0	$-$	1	1	0

That is, $1 = \gcd(160, 23) = 160 \times (-1) + 23 \times 7$. Check: $23 \times 7 =_{160} 1$.

So, we can pick $d = y = 7$.

RSA algorithm: a remark on the computed d

It must be $1 < d < \phi(n)$, so when $y < 0$ we simply reason modulo $\phi(n)$.

Consider, for example, $\phi(n) = 220$ and $e = 3$:

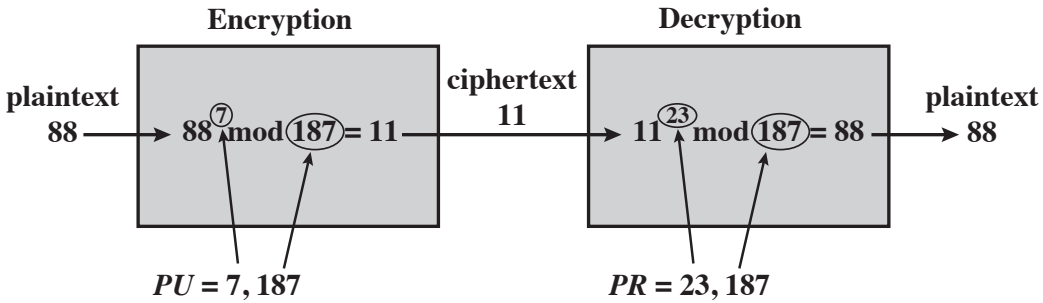
$$1 = \gcd(\phi(n), e) = \phi(n) \times x + e \times d = \gcd(220, 3) = 220 \times x + 3 \times y$$

A	B	$\lfloor A/B \rfloor$	D	x	y
220	3	73	1	1	-73
3	1	3	1	0	1
1	0	$-$	1	1	0

That is, $1 = \gcd(220, 3) = 220 \times 1 + 3 \times (-73) = 220 - 219$.

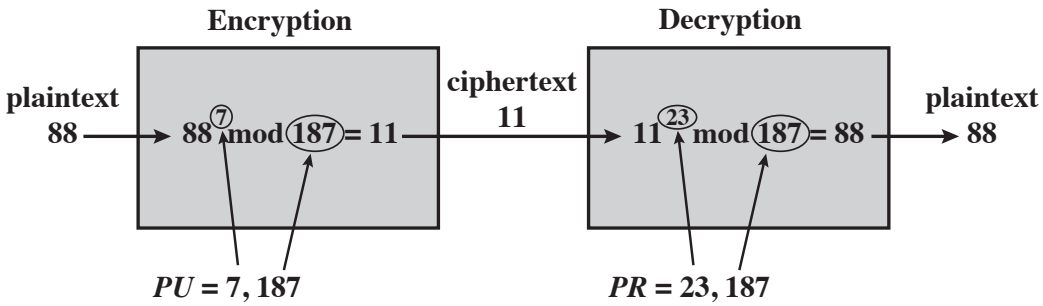
So, we can pick $d = 147$, i.e., $-73 \mod 220$.

RSA algorithm: another example (cont.)



- Let's continue the previous example.
- To encrypt a plaintext input $M = 88$, we need to calculate $C = M^e \bmod n = 88^7 \bmod 187 = 11$.
- We can do this by exploiting properties of modular arithmetic:
  - $88^7 \bmod 187 = ((88^4 \bmod 187) \times (88^2 \bmod 187) \times (88^1 \bmod 187)) \bmod 187$
  - $88^1 \bmod 187 = 88$
  - $88^2 \bmod 187 = 7744 \bmod 187 = 77$
  - $88^4 \bmod 187 = 59,969,536 \bmod 187 = 132$
  - $88^7 \bmod 187 = (88 \times 77 \times 132) \bmod 187 = 894,432 \bmod 187 = 11$

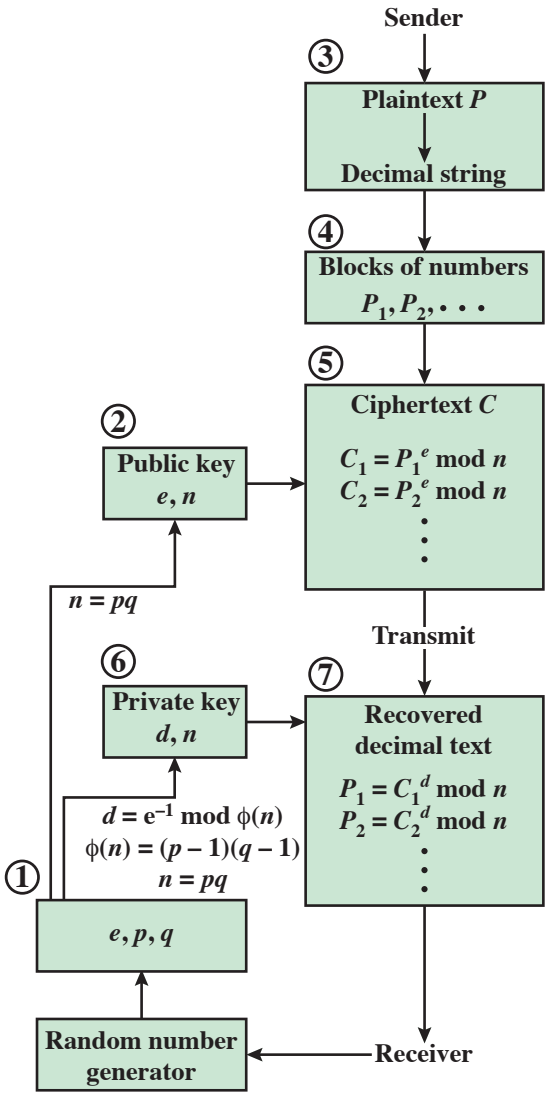
RSA algorithm: another example (cont.)



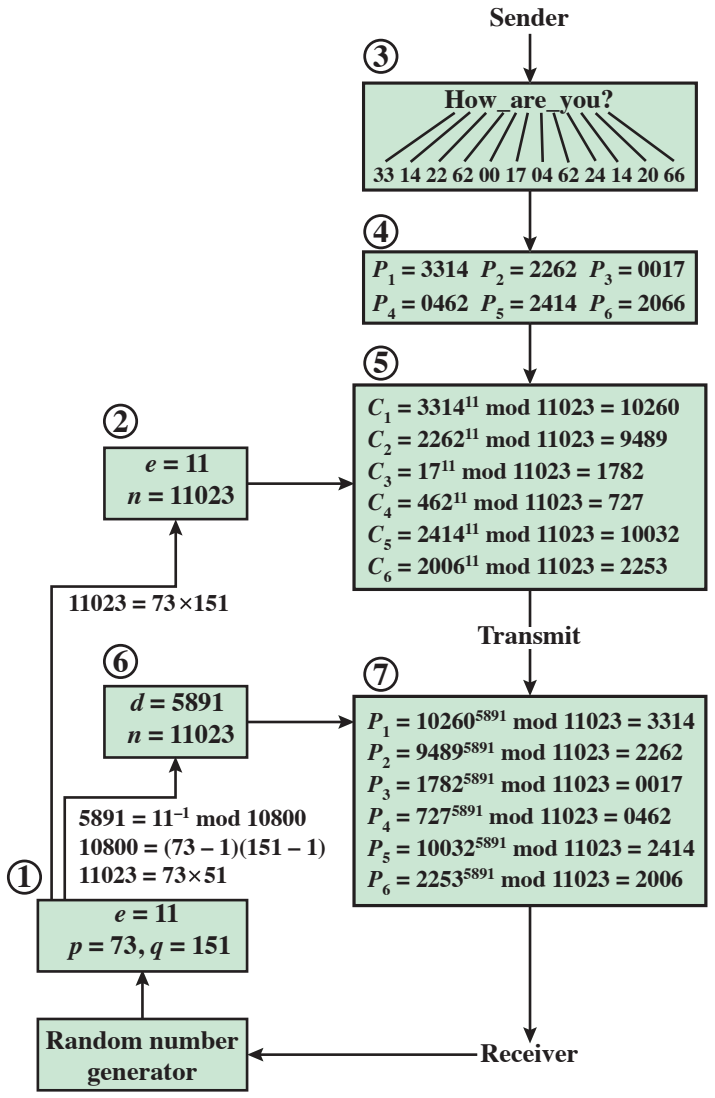
- For decryption, we calculate $M = C^d \bmod n = 11^{23} \bmod 187$:
  - $11^{23} \bmod 187 = ((11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \times (11^8 \bmod 187) \times (11^8 \bmod 187)) \bmod 187$
  - $11^1 \bmod 187 = 11$
  - $11^2 \bmod 187 = 121$
  - $11^4 \bmod 187 = 14,641 \bmod 187 = 55$
  - $11^8 \bmod 187 = 214,358,881 \bmod 187 = 33$
  - $11^{23} \bmod 187 = (11 \times 121 \times 55 \times 33 \times 33) \bmod 187 = 79,720,245 \bmod 187 = 88$

Use of RSA to process multiple blocks of data: example

- In this simple example, plaintext is an alphanumeric string.
- Each plaintext symbol is assigned a unique code of 2 decimal digits (e.g., a = 00, A = 26).
- A plaintext block consists of 4 decimal digits, or 2 alphanumeric characters.
- Circled numbers indicate order in which operations are performed.



(a) General approach



(b) Example

Correctness of RSA: more details

Since $ed \bmod \phi(n) = 1$, there exists $k \in \mathbb{N}$ such that $ed = 1 + k\phi(n)$.
Now consider whether or not $gcd(M, p) = 1$, i.e., whether $p \mid M$.
Case 1: If $gcd(M, p) = 1$ then by Fermat's theorem

$$M^{p-1} =_p 1$$

Raising both sides to power $k(q-1)$ and multiplying by M yields

$$M^{1+k(p-1)(q-1)} =_p M^{1+k\phi(n)} =_p M$$

Case 2: If $gcd(M, p) = p$ then last congruence is trivially valid since each side is congruent to 0 mod p.
Hence in both cases

$$M^{ed} =_p M$$

By the same argument $M^{ed} =_q M$ and since p and q are distinct primes, it follows that $M^{ed} =_n M$ and hence

$$C^d =_n (M^e)^d =_n M$$

RSA Security

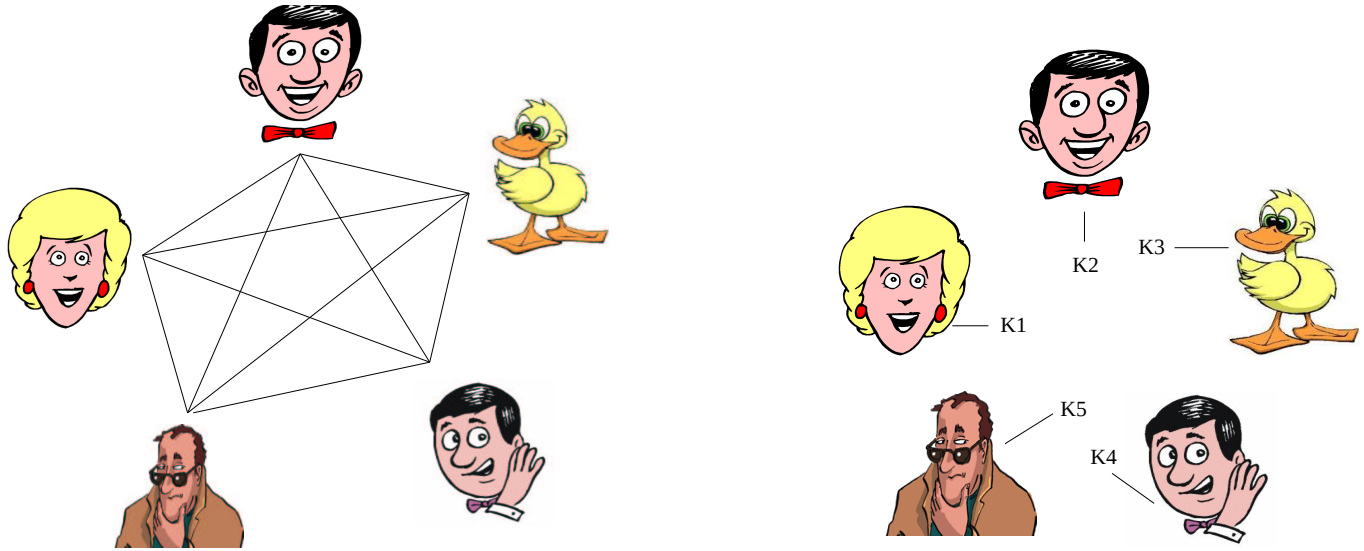
- Computation of secret d given (e, n).
  - As difficult as factorization. If we can factor $n = p \times q$ then we can compute $\phi(n) = (p-1) \times (q-1)$ and hence $d = e^{-1} \bmod \phi(n)$.
  - No known polynomial time algorithm.
    But given progress in factoring, n should have at least 1024 bits.
- Computation of M_i, given C_i, and (e, n).
  - Unclear (= no proof) whether it is necessary to compute d, i.e., to factorize n.

Hence: Progress in number theory could make RSA insecure.

Table of contents I

Table of contents II

The key distribution problem



- For $n = 1000$, we need 499,500 symmetric versus 2000 asymmetric keys.
- Trust:

  What good would it do after all to develop impenetrable cryptosystems, if their users were forced to share their keys with a key distribution center that could be compromised by either burglary or subpoena?"

  Whitfield Diffie, The first-ten years of public key cryptography, 1988

Asymmetric algorithms for secret key distribution

- Use public key encryption algorithms to support (faster) symmetric cryptography.
- We will see two approaches:
  - Secret key distribution with RSA.
  - Diffie-Hellman key exchange.

Table of contents I

Table of contents II

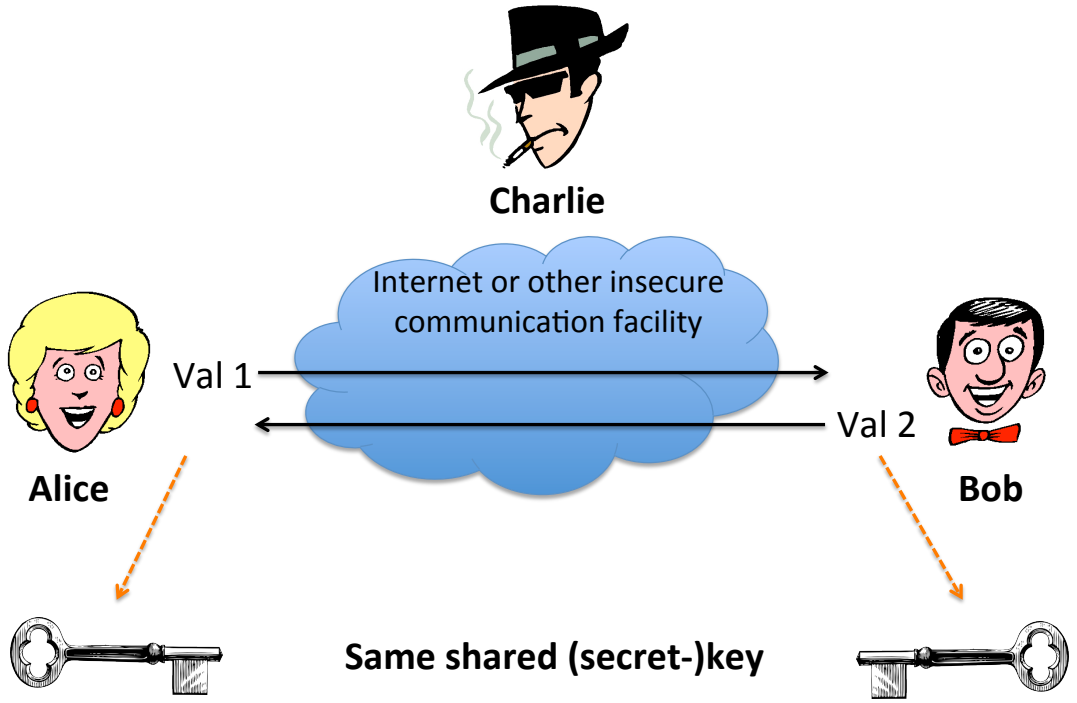
Secret key distribution with RSA

- Encryption of M (with public key (e, n)):
  - Choose secret key K randomly.
  - $C = (K^e \bmod n, E_K(M))$.
- Decryption (with private key (d, n)).
  - Split C into (C_1, C_2).
  - $K = C_1^d \bmod n \quad \textit{and} \quad M = D_K(C_2)$
- **Example:** SSL
  Alice chooses a secret, encrypts it with Bob's public key and rest of the session is protected based on that secret.
- **Problem:** if the private key (d, n) gets compromised, then K can be recovered by an intruder from previously observed traffic.

Table of contents I

Table of contents II

Diffie-Hellman key exchange: in a nutshell



- A simple public-key algorithm that enables two users to establish a secret key using a public-key scheme based on discrete logarithms.
- The protocol is secure only if the authenticity of the two participants can be established.

Background on discrete logarithms

- A **primitive root** s of a prime number p is a number whose powers generate $1, \ldots, p-1$.
  So $s^1 \bmod p$, $s^2 \bmod p$, ..., $s^{p-1} \bmod p$ are distinct, i.e., a permutation of 1 through $p-1$. Hence:

$$\forall b \in \mathbb{Z}.\, \exists i \in \{1, \ldots, p-1\}.\ b = s^i \bmod p$$

  In words: for any integer b and a primitive root s of prime number p, we can find a unique exponent i such that

$$b = s^i \bmod p$$

  where $1 \leq i \leq (p-1)$.
  i is called the **discrete logarithm** of b for base s, mod p.

- Computing discrete logarithms appears infeasible today.

Examples of primitive roots

- 2 is a primitive root of 5 since
  - $2^1 \bmod 5 = 2 \bmod 5 = 2$
  - $2^2 \bmod 5 = 4 \bmod 5 = 4$
  - $2^3 \bmod 5 = 8 \bmod 5 = 3$
  - $2^4 \bmod 5 = 16 \bmod 5 = 1$
- 3 is a primitive root of 5 since
  - $3^1 \bmod 5 = 3 \bmod 5 = 3$
  - $3^2 \bmod 5 = 9 \bmod 5 = 4$
  - $3^3 \bmod 5 = 27 \bmod 5 = 2$
  - $3^4 \bmod 5 = 81 \bmod 5 = 1$
- 3 is a primitive root of 7 since
  - $3^1 \bmod 7 = 3 \bmod 7 = 3$
  - $3^2 \bmod 7 = 9 \bmod 7 = 2$
  - $3^3 \bmod 7 = 27 \bmod 7 = 6$
  - $3^4 \bmod 7 = 81 \bmod 7 = 4$
  - $3^5 \bmod 7 = ((3^4 \bmod 7) \times (3 \bmod 7)) \bmod 7 = (4 \times 3) \bmod 7 = 5$
  - $3^6 \bmod 7 = ((3^5 \bmod 7) \times (3 \bmod 7)) \bmod 7 = (5 \times 3) \bmod 7 = 1$

Diffie-Hellman key exchange

1. Principals share a prime number q and an integer α that is a primitive root of q. Both q and α may be public, or A could send them in the first message.
2. A and B generate random numbers X_A and X_B (respectively) both less than q.
   X_A and X_B are the **private keys**.
3. A computes $Y_A = \alpha^{X_A} \bmod q$, B computes $Y_B = \alpha^{X_B} \bmod q$.
   Y_A and Y_B are the **public keys** (a.k.a. "Diffie-Hellman half keys").
4. A and B exchange Y_A and Y_B.
5. A computes $K_A = Y_B^{X_A} \bmod q$, B computes $K_B = Y_A^{X_B} \bmod q$.
   Keys are equal, i.e., $K_A = K_B$:

$$\begin{aligned} K_A &= Y_B^{X_A} \bmod q \\ &= (\alpha^{X_B} \bmod q)^{X_A} \bmod q = (\alpha^{X_B})^{X_A} \bmod q \\ &= \alpha^{X_A X_B} \bmod q = (\alpha^{X_A})^{X_B} \bmod q \\ &= (\alpha^{X_A} \bmod q)^{X_B} \bmod q = Y_A^{X_B} \bmod q \;=\; K_B \end{aligned}$$

Diffie-Hellman key exchange: ingredients

Global Public Elements

q	prime number
α	$\alpha < q$ and α a primitive root of q

User A Key Generation

Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation

Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

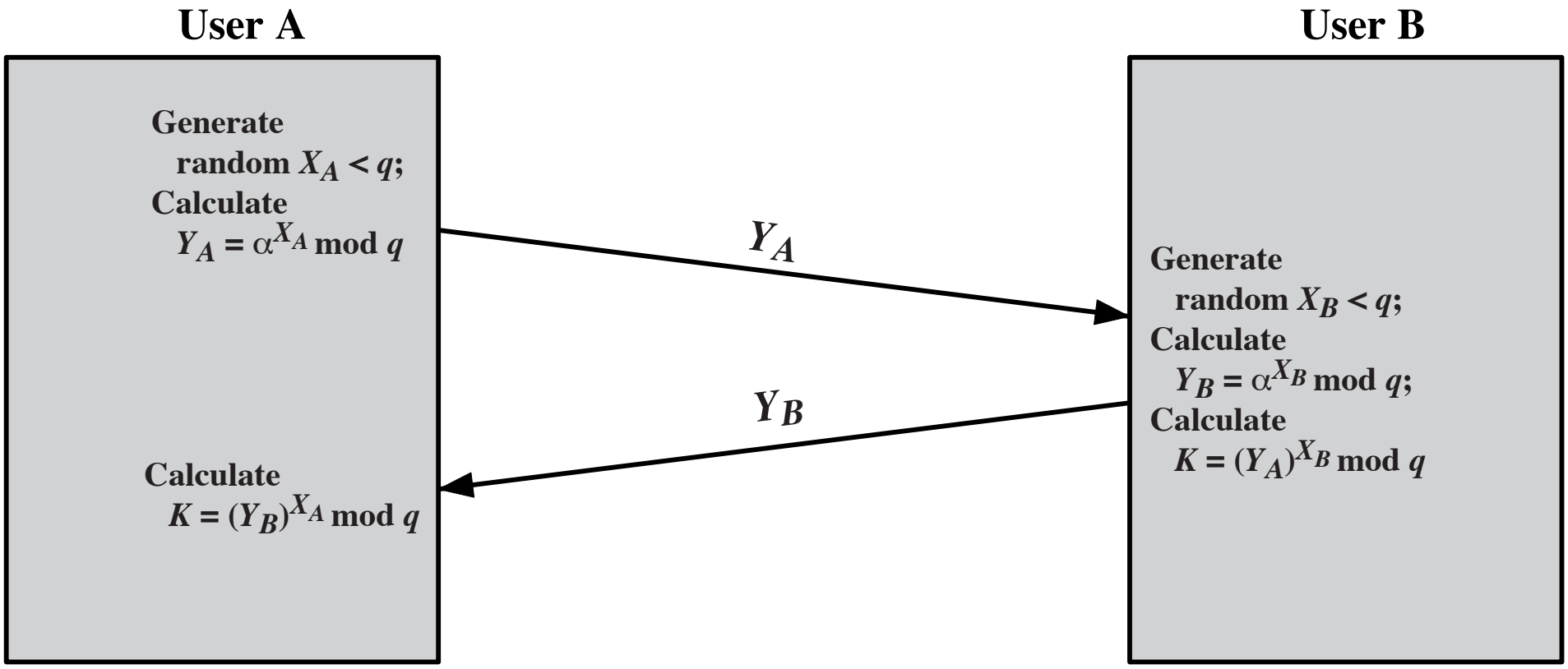
Calculation of Secret Key by User A

$K = (Y_B)^{X_A} \bmod q$

Calculation of Secret Key by User B

$K = (Y_A)^{X_B} \bmod q$

Diffie-Hellman key exchange: figure



Diffie-Hellman key exchange: strengths

- The shared secret key is never transmitted (not even in encrypted form)... it is created "out of nothing"!
  - $Y_A = \alpha^{X_A} \bmod q$ and $Y_B = \alpha^{X_B} \bmod q$ are the public keys.
  - X_A and X_B are the private keys.
  - Because X_A and X_B are private, an adversary C only has the following ingredients to work with: q, α, Y_A and Y_B.
  - Thus, C must take a discrete logarithm to determine the key. For example, to determine the private key of user B, C must compute
$$X_B = \mathrm{dlog}_{\alpha,q}(Y_B)$$
- Security of Diffie-Hellman key exchange lies in the fact that
  - it is relatively easy to calculate exponentials modulo a prime, but
  - it is very difficult to calculate discrete logarithms (e.g., it is considered infeasible for large primes).

  Security depends on the difficulty of computing discrete logarithms.

Diffie-Hellman: example (calculating the secret key)

- A and B choose prime number $q = 353$ and $\alpha = 3$ (which is one of the primitive roots of 353).
- A and B select private keys $X_A = 97$ and $X_B = 233$.
- Each computes its public key:
  - A computes $Y_A = \alpha^{X_A} \bmod q = 3^{97} \bmod 353 = 40$.
  - B computes $Y_B = \alpha^{X_B} \bmod q = 3^{233} \bmod 353 = 248$.
- After they exchange public keys, each can compute the common secret key K:
  - A computes $K = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160$.
  - B computes $K = (Y_A)^{X_B} \bmod 353 = 40^{233} \bmod 353 = 160$.
- Now they case use the symmetric key K to encrypt the messages they want to exchange.

Diffie-Hellman: example (attacking the key)

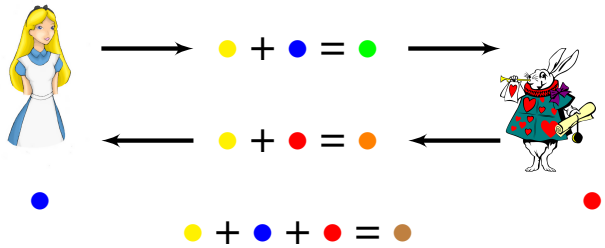
- Attacker C knows: $q = 353$, $\alpha = 3$, $Y_A = 40$ and $Y_B = 248$.
  - In this simple example, it would be possible by brute force to determine the secret key $K = 160$.
  - In particular, C can determine K by discovering a solution to
    - the equation $3^a \bmod 353 = 40$ or
    - the equation $3^b \bmod 353 = 248$.
  - Brute-force approach: calculate powers of 3 mod 353, stopping when the result equals either 40 or 248.
    - Desired answer is reached with the exponent value of 97, which provides $3^{97} \bmod 353 = 40$.
- With larger numbers, the problem becomes impractical.

An illustration of Diffie-Hellman with colors

The illustration, proposed by Singh, relies on two properties of colors:

- It is easy to mix two colors, for instance mixing yellow and blue gives green.
- Given a color that was obtained by mixing two colors, it is difficult to "separate" the color and obtain the initial colors.

Colored version of the Diffie-Hellman protocol:



- Assumptions:
  - yellow ● is a public color, known by everybody;
  - the secret color of Alice is the color blue ●;
  - the secret color of Bob is the color red ●.
- To agree on a new shared secret color, Alice and the white rabbit Bob can execute the following protocol:
  - In the first step, Alice sends the color green ● to Bob, which is the result of the public color ● mixed with her secret color ●.
  - Then Bob sends the color orange ● to Alice, which is the result of the public color ● mixed with his secret color ●.
  - Now they can both compute a common shared secret color brown ●:
    - Alice mixes the received orange color ● with her secret blue color ●.
    - Bob mixes the received green color ● with his secret red color ●.

See *The Code Book* by S. Singh (2000) and YouTube v=U1kybvKaUeQ or YouTube v=3QnD2c4Xovk

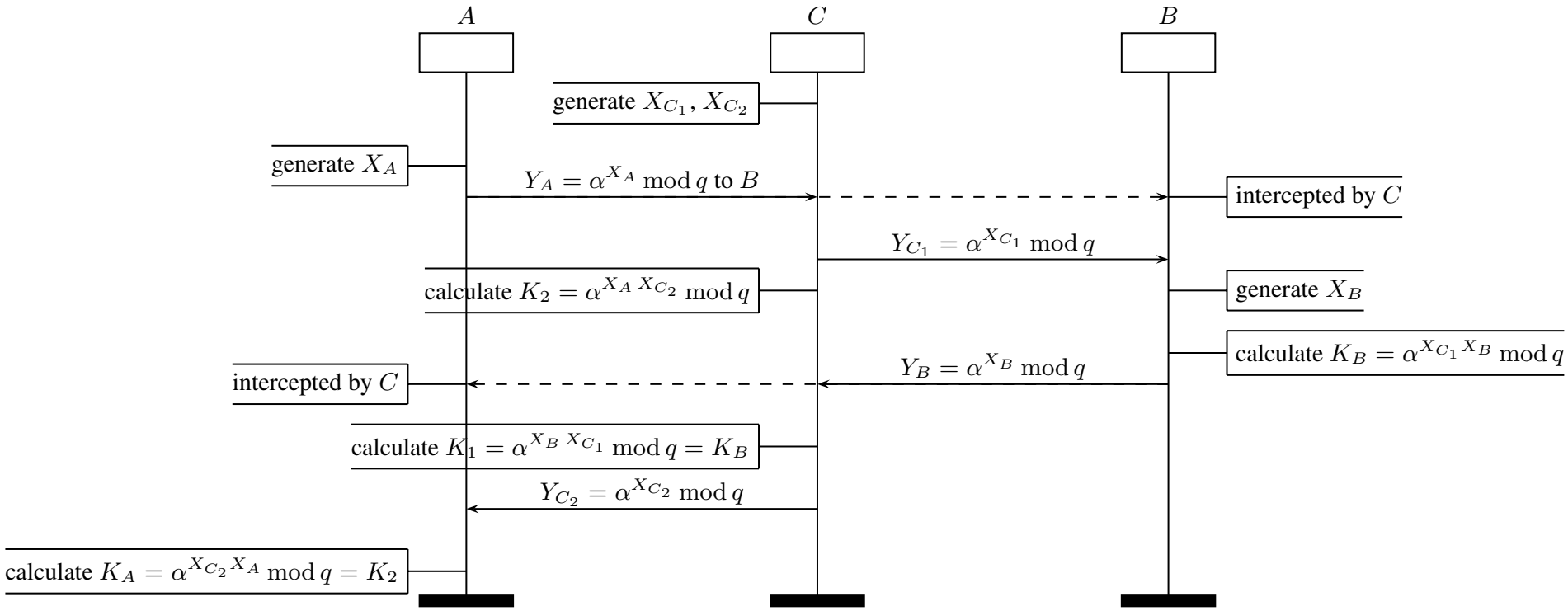
Diffie-Hellman key exchange: weakness

Keys are **unauthenticated** and thus Diffie-Hellman key exchange is vulnerable to the following **man-in-the-middle attack**:

0. Attacker C prepares for the attack by generating two random private keys X_{C_1} and X_{C_2} and then computing the corresponding public keys $Y_{C_1} = \alpha^{X_{C_1}} \bmod q$ and $Y_{C_2} = \alpha^{X_{C_2}} \bmod q$ (since α and q are public).
1. A generates X_A and transmits $Y_A = \alpha^{X_A} \bmod q$ to B.
2. C intercepts Y_A and transmits $Y_{C_1} = \alpha^{X_{C_1}} \bmod q$ to B. C also calculates $K_2 = (Y_A)^{X_{C_2}} \bmod q = (\alpha^{X_A} \bmod q)^{X_{C_2}} \bmod q = \alpha^{X_A X_{C_2}} \bmod q$.
3. B receives Y_{C_1}, generates X_B and calculates $K_B = (Y_{C_1})^{X_B} \bmod q = \alpha^{X_{C_1} X_B} \bmod q$.
4. B transmits $Y_B = \alpha^{X_B} \bmod q$ to A.
5. C intercepts Y_B and transmits $Y_{C_2} = \alpha^{X_{C_2}} \bmod q$ to A. C also calculates $K_1 = (Y_B)^{X_{C_1}} \bmod q = (\alpha^{X_B} \bmod q)^{X_{C_1}} \bmod q = \alpha^{X_B X_{C_1}} \bmod q = K_B$.
6. A receives Y_{C_2} and calculates $K_A = (Y_{C_2})^{X_A} \bmod q = \alpha^{X_{C_2} X_A} \bmod q = K_2$.

Now A and B think that they share a secret key, but instead A shares secret key $K_A = K_2$ with C and B shares secret key $K_B = K_1$ with C.

DH key exchange: man-in-the-middle attack



Now *A* and *B* think that they share a secret key, but instead *A* shares secret key $K_A = K_2$ with *C* and *B* shares secret key $K_B = K_1$ with *C*.

DH key exchange: man-in-the-middle attack

- All future communication between Bob and Alice is compromised in the following way.
  1. A sends an encrypted message M, i.e., $E(K_2, M)$.
  2. C intercepts the encrypted message and decrypts it to recover M.
  3. C sends to Bob either
     - $E(K_1, M)$, if C simply wants to eavesdrop on the communication without altering it, or
     - $E(K_1, M')$, where M' is any message, if C wants to modify the message going to B.

- The Diffie-Hellman key exchange is vulnerable to such an attack because it does not authenticate the participants.
  - This vulnerability can be overcome with the use of **digital signatures and public-key certificates** to achieve mutual authentication between A and B.
  - Typically: add an exchange of digitally signed identification (ID) tokens.

Group Diffie-Hellman (for three or more parties)

Given a Diffie-Hellman group (α, q), three honest parties Alice, Bob and Carol can generate together a secret key $K = \alpha^{X_A X_B X_C} \bmod q$ by:

1. Alice chooses a random large integer X_A and sends to Bob: $Y_A = \alpha^{X_A} \bmod q$
2. Bob chooses a random large integer X_B and sends to Carol $Y_B = \alpha^{X_B} \bmod q$
3. Carol chooses a random large integer X_C and sends to Alice: $Y_C = \alpha^{X_C} \bmod q$
4. Alice sends to Bob $Y'_C = Y_C^{X_A} \bmod q$
5. Bob sends to Carol $Y'_A = Y_A^{X_B} \bmod q$
6. Carol sends to Alice $Y'_B = Y_B^{X_C} \bmod q$
7. Alice computes: $K = {Y'_B}^{X_A} \bmod q$
8. Bob computes: $K = {Y'_C}^{X_B} \bmod q$
9. Carol computes: $K = {Y'_A}^{X_C} \bmod q$

Can be extended to more parties by adding more rounds of computations.

Table of contents I

Table of contents II

El Gamal variant of Diffie-Hellman key exchange

- Setup: same as Diffie-Hellman. Public prime q and generator α.
- Moreover, let E be any symmetric encryption function.
- Schema
  1. A chooses X_A and computes $Y_A = \alpha^{X_A} \bmod q$.
     $$A \to B\text{: } Y_A.$$
  2. B chooses integer X_B, computes $Y_B = \alpha^{X_B} \bmod q$, and computes key $K = Y_A^{X_B} \bmod q$.
     $$B \to A\text{: } (E(M, K), Y_B)$$
  3. A computes $K = Y_B^{X_A} \bmod q$ and uses K to decrypt $E(M, K)$.

  Red parts are Diffie-Hellman.

Table of contents I

Table of contents II

Massey-Omura scheme

- Encryption without shared keys based on discrete logarithm problem.
- Principals share (public) prime p.
- $u \in \{A, B\}$ chooses (private) $e_u, d_u \in \mathbb{Z}$ such that

$$e_u d_u \bmod (p-1) = 1\,.$$

So $(p-1) \mid (e_u d_u - 1)$. So there is a k where $e_u d_u = k(p-1)+1$. Hence, by Euler's theorem, for all $m \in \{1, \ldots p-1\}$

$$m^{e_u d_u} \bmod p = m^{k(p-1)} m \bmod p = m \bmod p = m$$

Massey-Omura Protocol

Step 1 $A \rightarrow B$: $m^{e_A} \bmod p$

Step 2 $B \rightarrow A$: $m^{e_A e_B} \bmod p$

Step 3 $A \rightarrow B$: $m^{e_A e_B d_A} \bmod p \quad (= m^{e_B})$

Step 4 $m^{e_A e_B d_A d_B} \bmod p \quad (= m)$

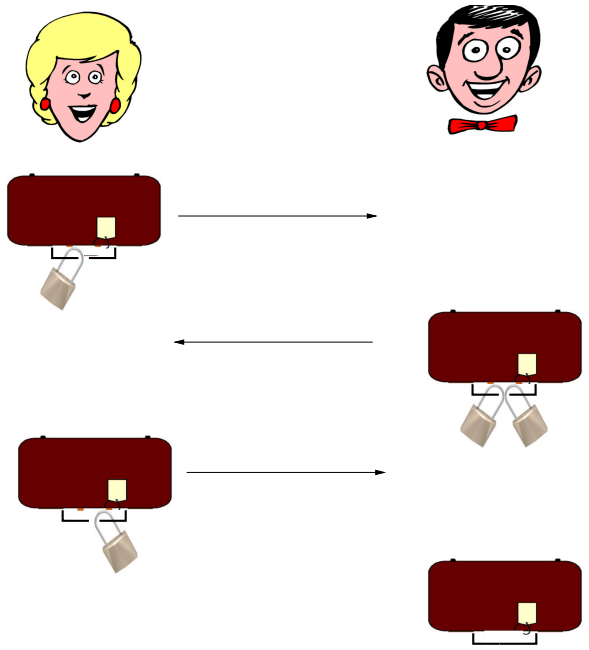
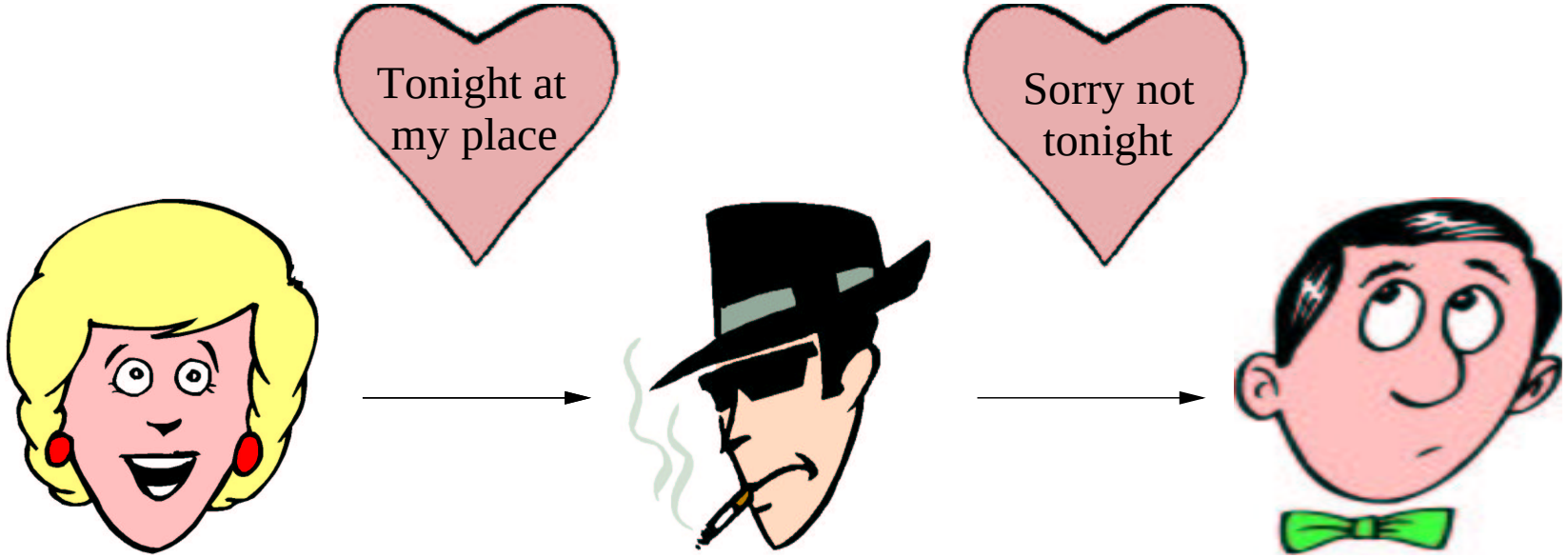


Table of contents I

Table of contents II

Message Integrity



- Message or data integrity is the property that data has not been altered in an unauthorized manner since the time it was created, transmitted, or stored by an authorized source.
- In operating systems, access control can be used to ensure data integrity. In open networks, one uses cryptographic means.

Cryptographic hash requirements

- Motivation: create a data “fingerprint".
- A hash function $h(x)$ (in the general sense) has the properties:
  1. Compression: h maps an input x of arbitrary bit length to an output $h(x)$ of fixed bit length n.
  2. Polynomial time computable.
- **Example** (longitudinal redundancy check):
  Given m blocks of n-bit input $b_1, \ldots, b_m$, form the n-bit checksum c from the bitwise xor of every block. I.e., (for $1 \leq i \leq n$)

$$c_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}$$

- Cryptographic techniques can be seen as a refinement of checksum techniques to handle active forger.

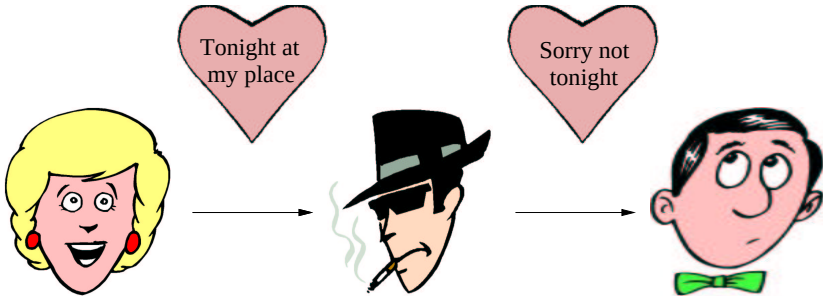
Cryptographic hash requirements (cont.)

- $h(x)$ is a cryptographic hash function if it is additionally:
  - One-way (or pre-image resistant).
    Given y, it is hard to compute an x where $h(x) = y$.
  - And usually either

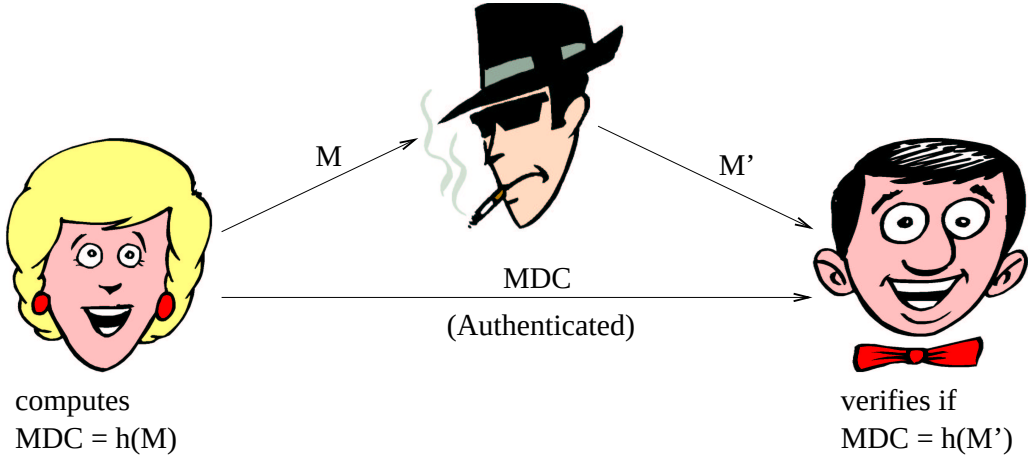
    2nd-preimage resistance: It is computationally infeasible to find a second input that has the same output as any specified input, i.e., given x to find an $x' \neq x$ such that $h(x) = h(x')$.

    Collision resistance: It is difficult to find two distinct inputs x, x' where $h(x) = h(x')$.
- Hash value also called message digest or modification detection code.
- An application: password files.
  - For password p, store $h(p)$ in password file.
  - Requires only pre-image resistance. Why?
  - Often combined with *salt* s, i.e., store pair $(s, h(s, p))$.

- Message or data integrity is the property that data has not been altered in an unauthorized manner since the time it was created, transmitted, or stored by an authorized source.



- Message integrity: modification detection code provides checkable fingerprint.



Requires 2nd-preimage resistance and authenticated MDC. Typical application: signed hashes.

Collision resistance and the birthday paradox

- When attacker controls input to hash function, collision resistance is important.
  Unimportant for password hashing. What about signing contracts?
- **Birthday paradox**
  - How many people must be in a room such that the probability p that one has your birthday (intended as day and month, not year) is $p > .5$?
    The probability that one of n people has your birthday is $n/365$, so for $n = 183$, $p > 0.5$.
  - How many people must be in a room such that the probability p that any two share the same birthday is $p > .5$? 23 persons!
  - Generalization: Let h have 2^m possible outputs. h must be applied to $2^{m/2}$ inputs so that the probability of a collision is greater than 0.5.

A birthday attack

- One might think a 64 bit hash code is secure.
  Preimage resistance means, on average 2^{63} messages must be tried.
- Birthday attack for finding collisions:
  - Suppose A is willing to sign a contract with B. Then B prepares version x, good for A, and version y, which bankrupts her.
    - B generates "good" messages $x_1, \ldots$.
    - Similarly he dovetails the generation of the "bad" $y_1, \ldots$.
    - He stops when $h(x_i) = h(y_j)$.
  - A signs $h(x_i)$. Later B uses signature for y_j.
- On average, work required is order of 2^{32}.
  So double your hash size if collision avoidance important!

A letter in 2^{37} variations

Dear Anthony,

{ This letter is / I am writing } to introduce { you to / to you } { Mr. / } Alfred { P. / }

Barton, the { new / newly appointed } { chief / senior } jewellery buyer for

{ our / the } Northern { European / Europe } { area / division }. He { will take / has taken }

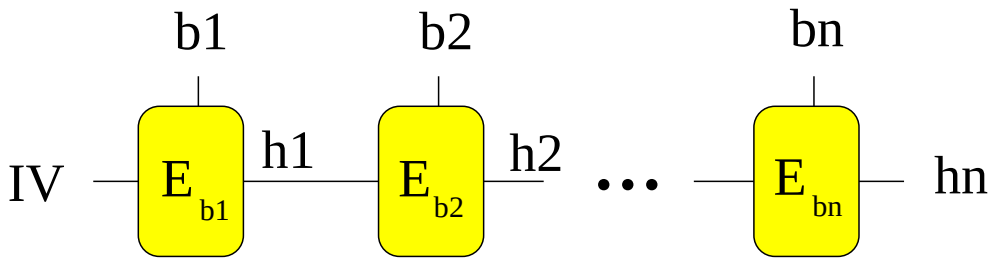
over { the / } responsibility for { all / the whole of } our interests in

{ watches and jewellery / jewellery and watches } in the { area / region }. ...

Constructing a cryptographic hash function

- Simplest constructions use block chaining techniques (Rabin 1978):
  - Divide message M into fixed size blocks $b_1, \ldots b_n$.
  - Use symmetric encryption algorithm, e.g., DES

$$h_0 = IV \text{ (initial value)} \qquad \text{and} \qquad h_i = E_{b_i}(h_{i-1})$$



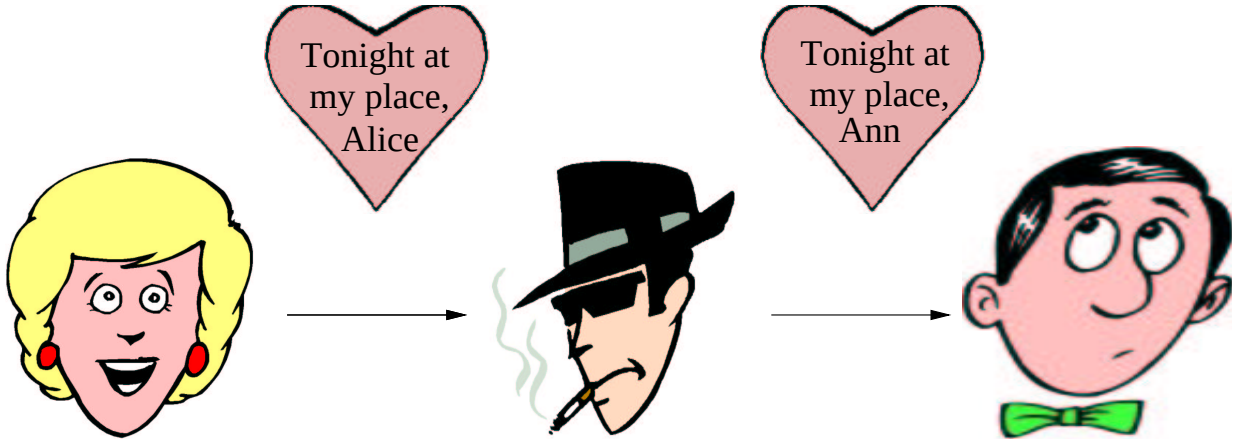
  - Similar to Cipher Block Chaining, but no secret key.
- Modern algorithms are considerably more complex.
  - **MD5** (Message Digest; by Rivest): 128 bit hashes. Known weakness.
  - **SHA** (Secure Hash Algorithm; US Standard): 160 bit hashes. SHA-1 widely used but deprecated. SHA-3 secure(?).

Table of contents I

Table of contents II

Message authentication codes

- Message authentication also called data-origin authentication.



Observe that message authentication implies message integrity.

- Contrast to entity (or user) authentication

- **Message Authentication Codes (MACs)** and **digital signatures** are two main techniques for establishing message authentication.

Table of contents I

Table of contents II

Message Authentication Codes (MACs)

MAC algorithm

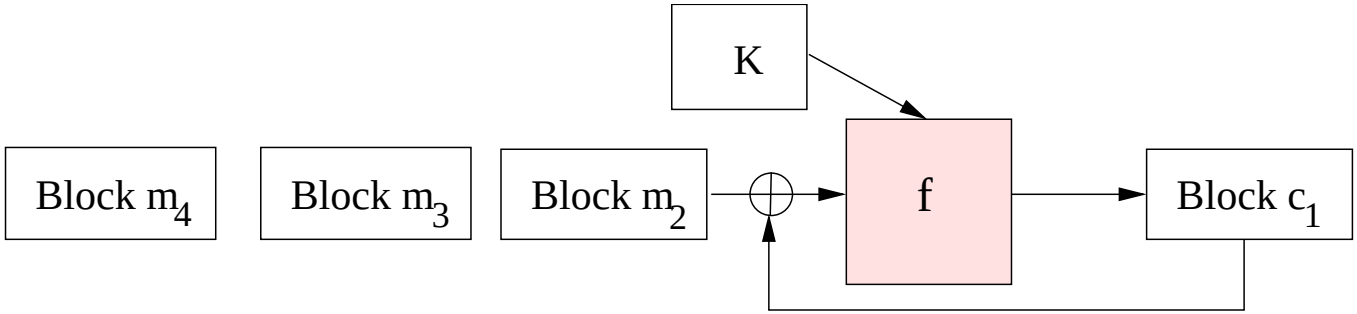
A MAC algorithm is a family of hash functions h_k parameterized by a secret key k. The h_k must be computation-resistant: given zero or more MAC pairs $(x_i, h_k(x_i))$, it is infeasible to compute $(x, h_k(x))$ for any new input $x \neq x_i$.

- Message authentication with MACs

  - h_k any enciphering algorithm that returns fixed length output.
  - Message authentication? Yes, for verified messages.
  - Non-repudiation? Not with symmetric cipher.
  - Freshness? No detection of message replay.

Constructing MACs (I)

- Simplest realization based on cipher-block chaining.



$$c_1 = E_K(m_1 \oplus 0)$$
$$c_i = E_K(c_{i-1} \oplus m_i)$$

- E is a block cipher, e.g., DES.
- c_n is MAC. Alternatively i left-most bits can be used.

Constructing MACs (II)

- MACs can be built from MDCs.
- Secret prefix method: $h(\{K, M\})$ (K shared between A and B)
  Weakness: Suppose h computed as a block chaining MDC, i.e.,

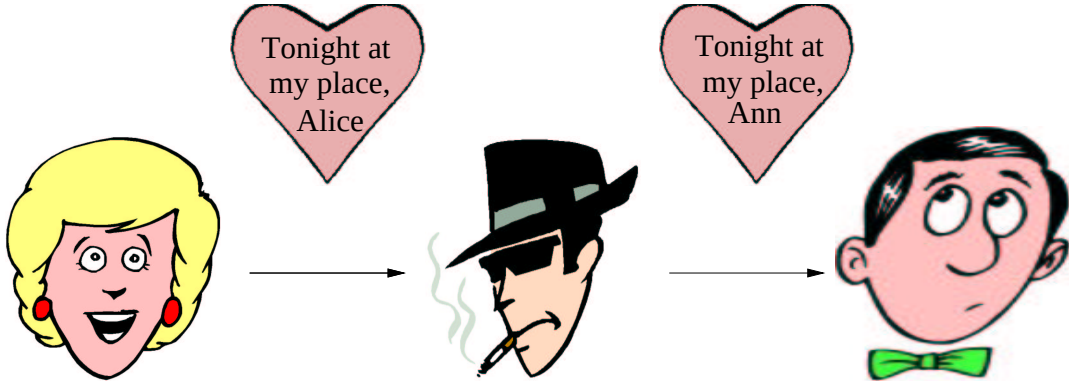
$$h_0 = IV \text{ (initial value)} \quad \text{and} \quad h_i = E_{b_i}(h_{i-1})$$

  - Suppose $M = b_1, \ldots, b_n$.
  - Then $h(\{K, M, b_{n+1}\}) = E_{b_{n+1}}(h(\{K, M\}))$.
  - So given $h(\{K, M, \}))$, $C \notin \{A, B\}$ can forge MAC for any $\{M, b_{n+1}\}$.
- Alternatives $h(\{M, K\})$, $h(\{k, p, x, k\})$, and $h(\{k, p, h(\{k, p', x\})\})$, where p, p' pad to fill up block.

Table of contents I

Table of contents II

The digital signature problem



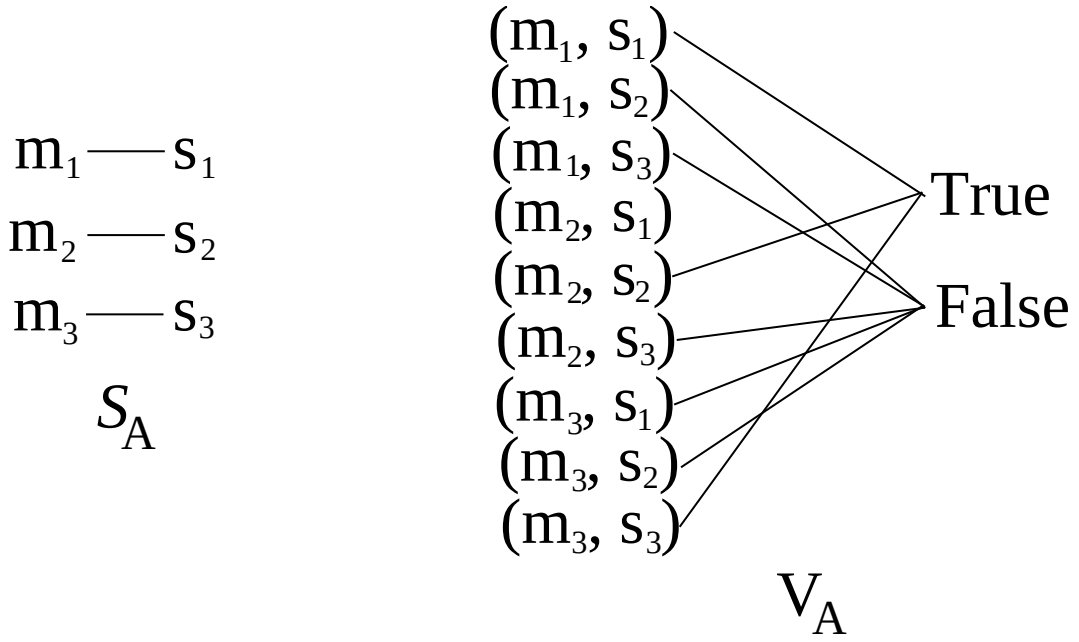
- Problem of proof of data origin.
  How do we know, or prove, that a message originated from a particular person?
- Public key cryptography supports both knowing and proving to others (non-repudiation).
  Would this be possible using a shared key?

Digital signature requirements

- Signature is fundamental in authentication and nonrepudiation.
- Nomenclature and set-up
  - $\mathcal{M}$ is set of messages that can be signed.
  - $\mathcal{S}$ is set of elements called signatures, e.g., n-bit strings.
  - $S_A : \mathcal{M} \to \mathcal{S}$, is a signing transformation for entity A, and kept secret by A.
  - $V_A : \mathcal{M} \times \mathcal{S} \to \{true, false\}$ is a verification transformation for A's signature and is publicly known.
- S_A and V_A provide digital signature scheme for A.

Signature schema (cont.)

- Example



- Signing procedure: A creates a signature for $m \in \mathcal{M}$ by: Compute $s = S_A(m)$ and transmit pair (m, s).
- Verification procedure. B verifies A's signature of (m, s) by: Compute $u = V_A(m, s)$. Accept signature only if $u = true$.
- Secrecy requires: it is hard for any entity other than A to find, for any $m \in \mathcal{M}$, an $s \in \mathcal{S}$, where $V_A(m, s) = true$.

Implementing digital signatures

- Can be based on (reversible) public-key encryption systems.
- Consider $E_e : \mathcal{M} \to \mathcal{C}$ is a public-key transformation. Moreover, suppose that $\mathcal{M} = \mathcal{C}$. If D_d is the decryption transformation corresponding to E_e, then since both are permutations

$$D_d(E_e(m)) = E_e(D_d(m)) = m \qquad \text{for all } m \in \mathcal{M}.$$

  A public-key encryption scheme of this type is called reversible.
- Construction for a digital signature schema
  1. Let $\mathcal{M}$ and $\mathcal{C}$ be message and signature space, with $\mathcal{M} = \mathcal{C}$.
  2. Let (e,d) be a key pair for the public-key encryption scheme.
  3. Define signing function S_A to be D_d. I.e., $s = D_d(m)$.
  4. Define V_A by

$$V_A(m, s) = \begin{cases} \mathit{true}, & \text{if } E_e(s) = m, \\ \mathit{false}, & \text{otherwise} \end{cases}$$

Implementing digital signatures

- Previous schema admits a forgery attack:
  1. Attacker B selects a random $s \in \mathcal{S}$ and computes $m = E_e(s)$.
  2. Since $\mathcal{S} = \mathcal{M}$, he can submit (m, s) as message with signature.
  3. Verification returns *true* even though A did not sign m!
- Solution: let some subset $\mathcal{M}' \subset \mathcal{M}$ constitute the signable messages.
  Redefine verifier $V_A : \mathcal{S} \to \{true, false\}$ as:

$$V_A(s) = \begin{cases} true, & \text{if } E_e(s) \in \mathcal{M}' \\ false, & \text{otherwise} \end{cases}$$

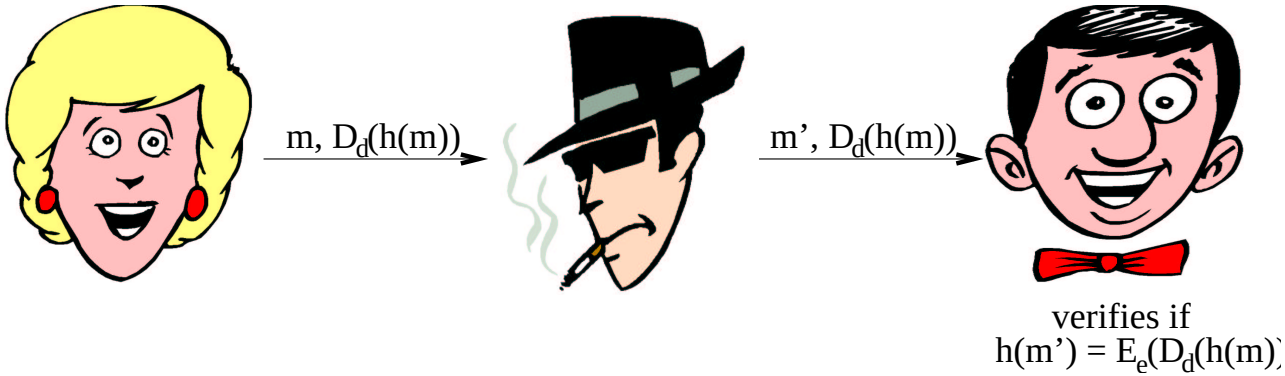
  Messages can be recovered since $m = E_e(s)$.
- Secure when $\mathcal{M}'$ is a sufficiently small subset of $\mathcal{M}$.

RSA and DSA

- RSA provides a realization of digital signatures:

$$D_d(E_e(m)) = E_e(D_d(m)) = m$$

Forgery prevented by signing messages with fixed structure, e.g.,
  - Message names its sender, or (more typically).
  - Cryptographic hash signed, sent with the message.



Pair can additionally be encrypted for confidentiality.

- Also: **Digital Signature Algorithm (DSA)**, a Federal Information Processing Standard for digital signatures. It was proposed by the National Institute of Standards and Technology (NIST) in August 1991 for use in their **Digital Signature Standard (DSS)**.

Table of contents I

Table of contents II

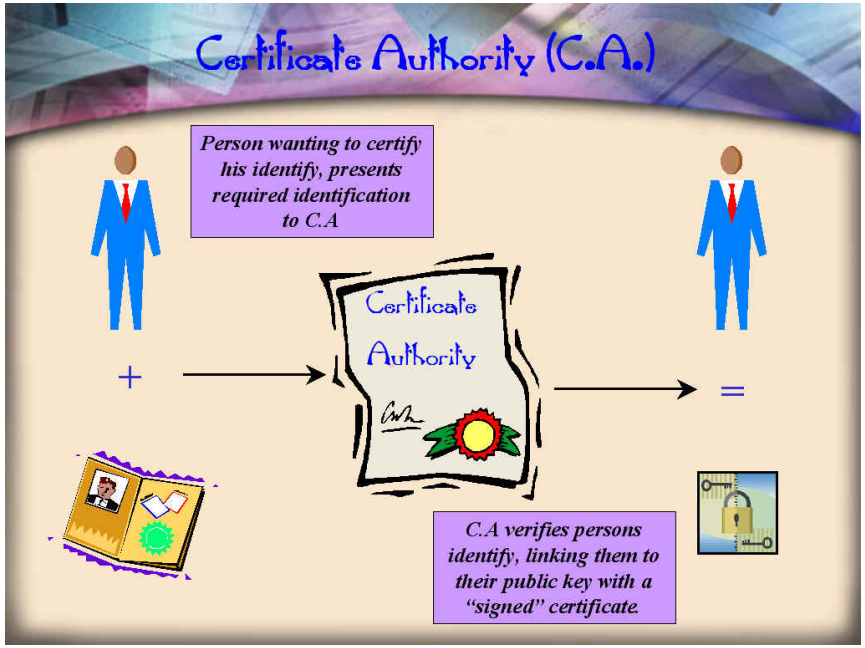
Key management

- Key management refers to
  - the distribution of cryptographic keys,
  - the mechanisms used to bind an identity to a key,
  - the generation, maintenance and revoking of such keys.
- We have seen (and will see) protocols for authentication and key generation and distribution/exchange, and we have seen digital signatures.
- We now look at public key infrastructures:
  - How it is possible to bind a public key to (a representation of) an identity?
  - How can we represent identity (“naming”)?

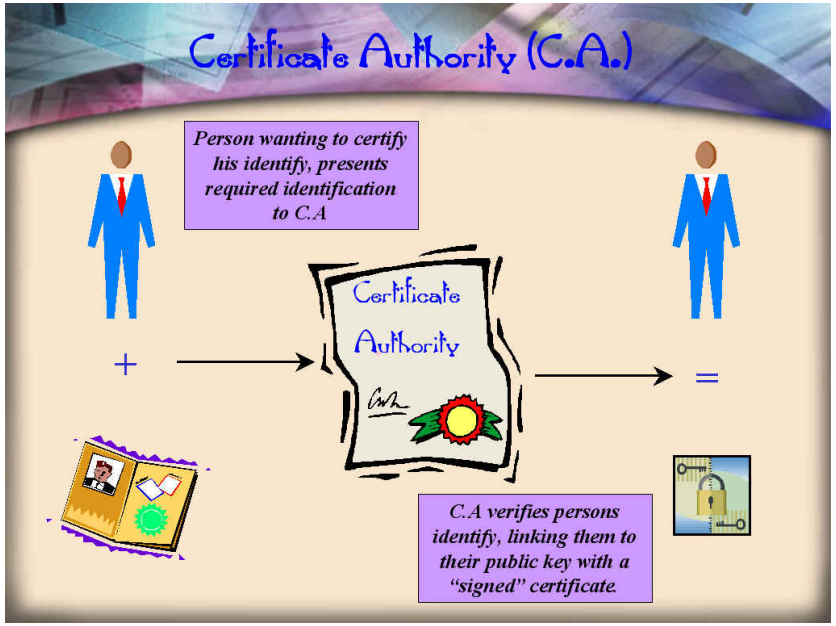
Public Key Infrastructure (PKI)

A PKI is an infrastructure that allows principals to recognize which public key belongs to whom (i.e. to bind public keys to principals).

- To join the PKI, Alice
  - generates her own public/private key pair,
  - takes her public key PU_A to a certification authority (CA) that everybody trusts and says "I am Alice and PU_A is my public key".
- The CA verifies that Alice is who she says she is, and then signs a digital certificate that states "Key PU_A belongs to Alice".



PKI: mutual trust through CA



- Any Bob can now check the certificate to obtain Alice's public key PU_A and accept it as valid.
- Alice can similarly obtain Bob's public key PU_B.

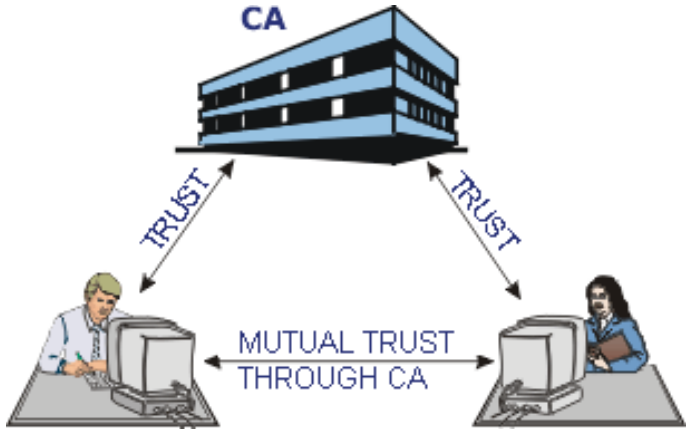
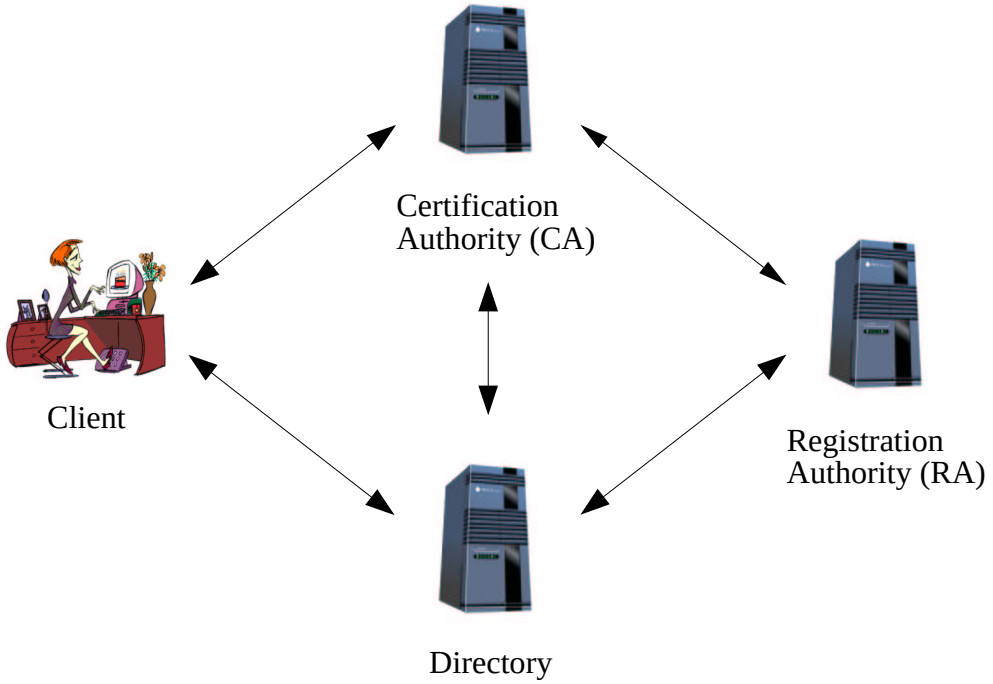


Table of contents I

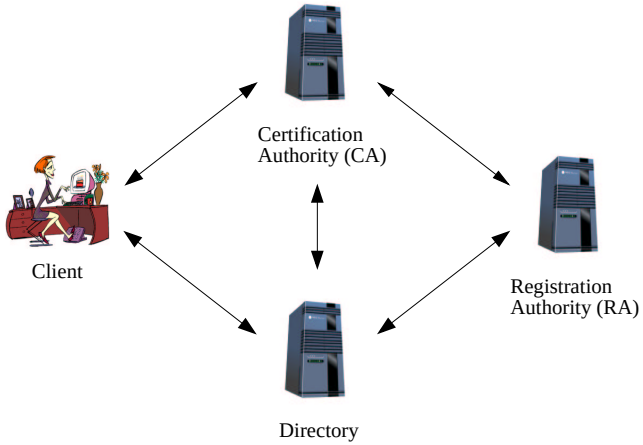
Table of contents II

PKI components



- PKI services:
  - Linking public keys to entities (certificates).
  - Key life-cycle management (key revocation, recovery, updates).

PKI components (cont.)



- Certification Authority (CA):
  - Creates certificates and publishes them in directory.
  - Maintains the Certificate Revocation List (CRL) in the directory.
    CRL checked actively by single clients or by validation services.
  - Backs up certain keys (for key recovery or escrow).
- Directory:
  - Makes user certificates and CRLs available.
  - Must identify users uniquely (needs fresh/accurate user data).
  - Must be highly available.
- Registration Authority (RA):
  - Manages the process of registering users and issuing certificates.
  - Ensures proper user identification.

PKI components (cont.)

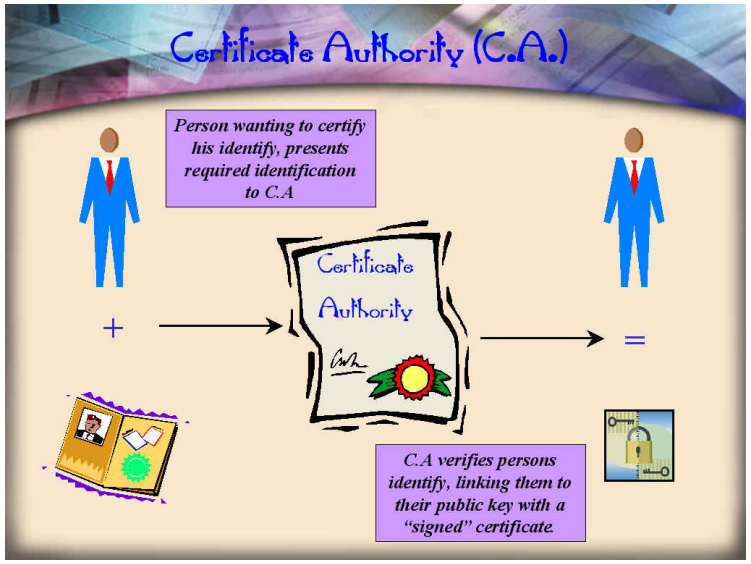
- Clients:
  - Different uses of a PKI, e.g.
    - authentication (one-way, two-way),
    - file encryption,
    - signed documents and transactions.
  - Different kinds of PKI integration:
    - PKI-enabled commercial applications.
    - Platform support, e.g. Windows Crypto API (CAPI).
    - Custom applications.
- Two different kinds of PKIs:
  - Open: used across the borders of companies/communities.
  - Closed: limited to a closed user group (e.g. a company's VPN).

Table of contents I

Table of contents II

Certificates

- A certificate is a token that binds an identity to a key.
  CA *Cathy* signs (with PR_{Cathy}) a message containing a representation of identity (*Alice*), the corresponding public key (PU_{Alice}), and a timestamp (T):



$$C_{Alice} = M || E(PR_{Cathy}, H(M))$$

  where $M = \langle PU_{Alice}, Alice, T \rangle$

- Bob can check the certificate to obtain Alice's public key and accept it as valid.
  However, Bob must know Cathy's public key to validate the certificate.
  This pushes the problem to another level: how can the issuer's certificate be validated?

Certificate signature chains (cont.)

- Two approaches to this problem are
  - to construct a tree hierarchy, with the public key of the root known out of band,
  - or to allow an arbitrary arrangement of certifiers and rely on each individual's knowledge of the certifiers.

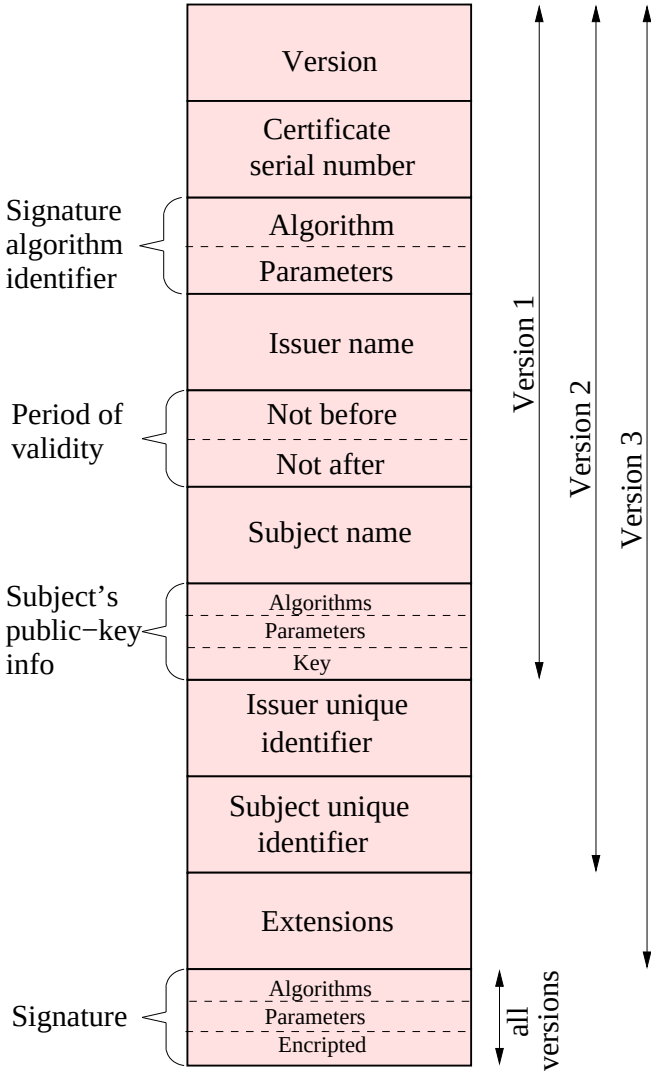
  To illustrate these approaches, and certificates and certification in general, we consider the example of X.509.

X.509: Certification signature chains

- X.509: a standard, part of the X.500 series of ITU-T recommendations, that defines a framework for the provision of authentication services.
- The certificate structure (certificate formats and certification validation) and authentication protocols defined in X.509 are used in a variety of contexts, e.g. in IPSEC, SSL/TLS, SET, and S/MIME.
- It is based on public-key cryptography (it recommends RSA), hashes, and digital signatures.
- X.509v3 issued in 1995 and revised in 2000.

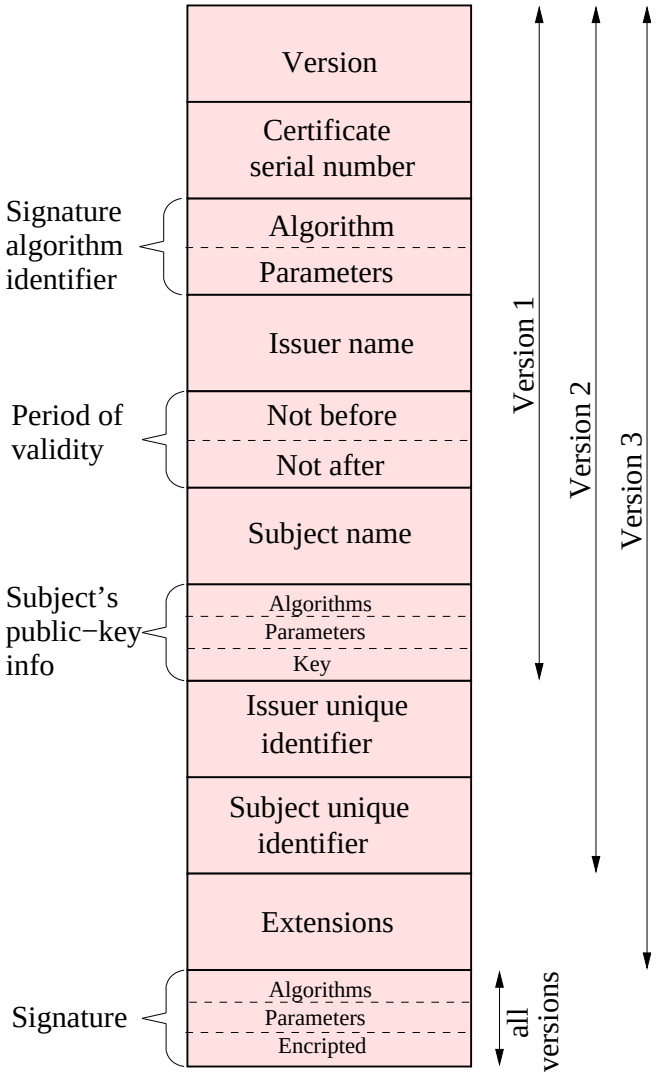
The X.509 certificate

- The heart of the X.509 scheme is the public-key certificate associated with each user.
- Certificates are created by the CA and are placed in the directory by the CA or by the user.
  The directory server merely provides an easily accessible location for users to obtain certificates.



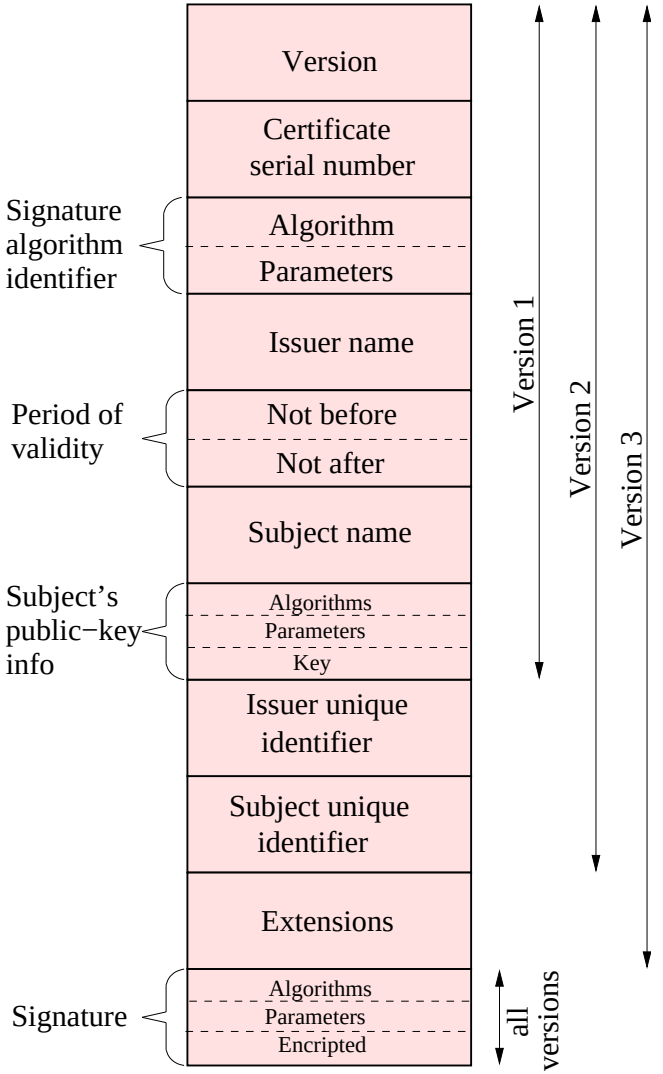
X.509 certificate components

- Serial number must be unique among the certificates issued by this issuer.
  I.e. pair *(issuer name, serial number)* must be unique.
- Signature algorithm identifier of algorithm, and any parameters, used to sign the certificate.
- Issuer name is X.500 name of CA that created and signed this certificate.
  Optional string issuer unique identifier in the event the X.500 name has been reused for different entities.



X.509 certificate components (cont.)

- Period of validity.
- Subject name is the name of the user to whom the certificate refers (i.e. the user whose public key is certified).
  Optional bit string subject unique identifier in the event the X.500 name has been reused for different entities.
- Subject public-key info identifies the algorithm, its parameters, and the subject's public key.
- Signature contains the hash code of the other fields, encrypted with the CA's private key.



X.509: obtaining a user's certificate

The certificate of user A issued (and signed with PR_{CA}) by the certification authority CA is

$$CA \ll A \gg = M \| E(PR_{CA}, H(M))$$

where $M = \langle V, SN, AI, CA, T_A, A, Ap \rangle$

- To validate $CA \ll A \gg$, and verify the user public key that was generated, Bob obtains CA's public key for the particular signature algorithm and deciphers the signature.
- Bob then uses the information in the signature field to recompute the hash value from the other fields. If it matches the deciphered signature, the signature is valid if the issuer's public key is correct.
- Bob then checks the period of validity to ensure that the certificate is current.

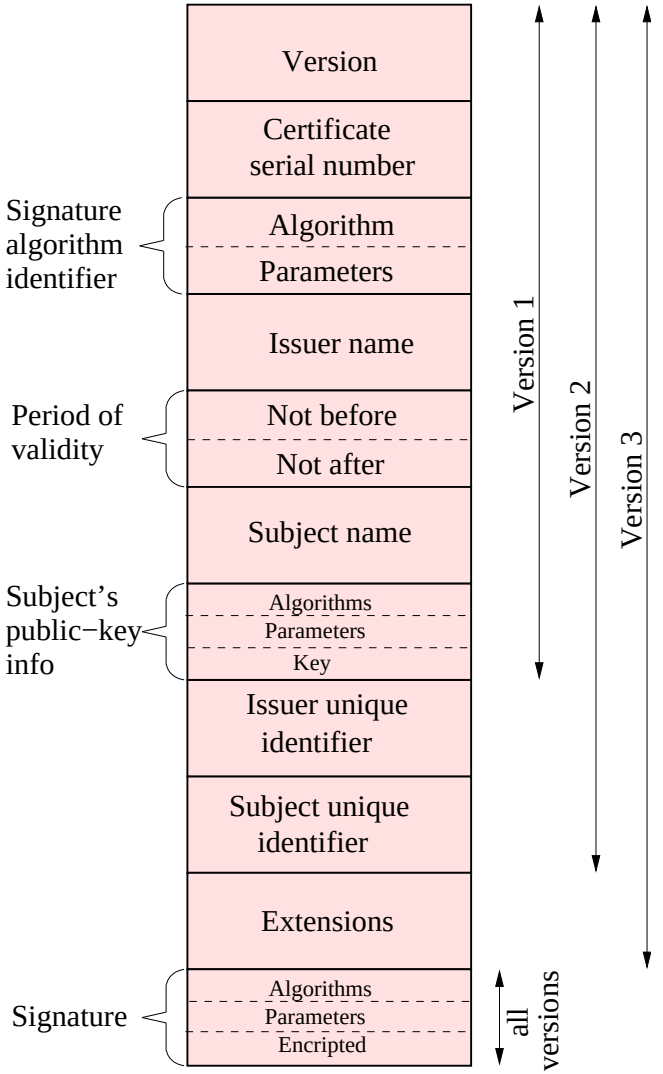


Table of contents I

Table of contents II

Trust models

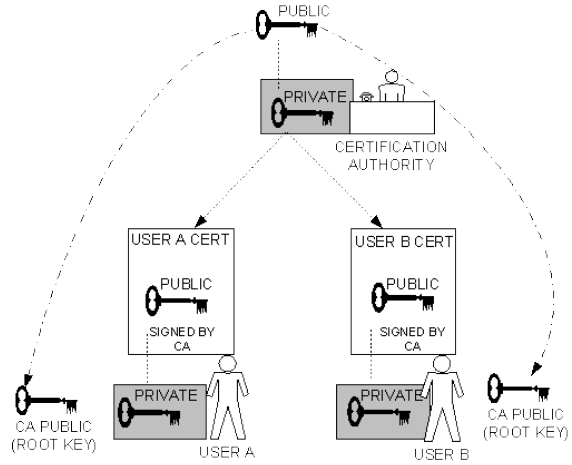
Different trust models dictate how users will go about establishing certificate validity:

- Direct trust
- Hierarchical trust
- Web of trust

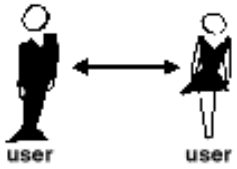
Trust models: direct trust

Direct trust: if all users subscribe to the same *CA*, then there is a common trust of that *CA*.

All user certificates can be placed in the same directory for access by all users.

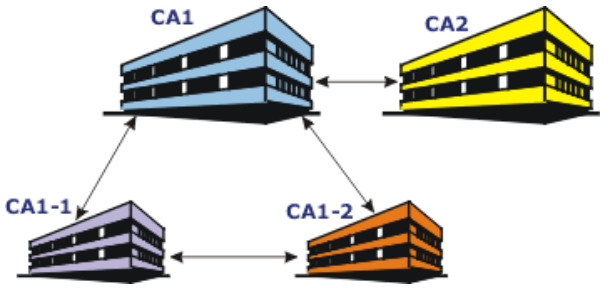


Also, certificate transmission from user to user (as in PGP).



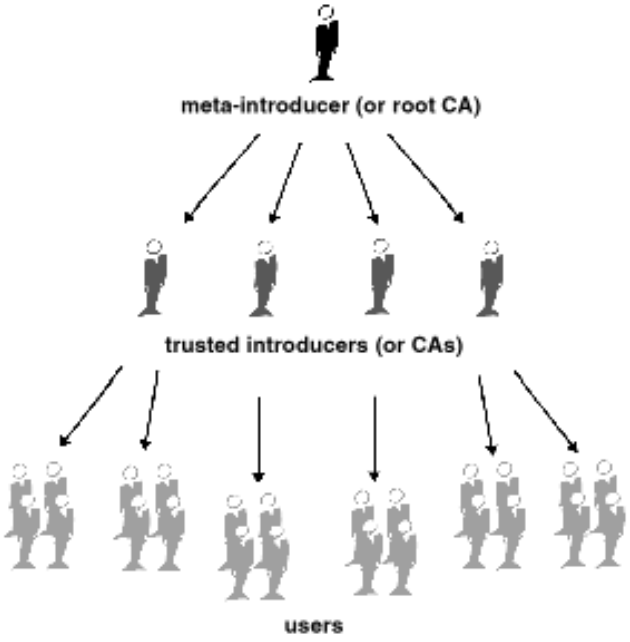
Trust models: hierarchical trust

Hierarchical trust: for a large community of users, it is more practical to have a number of CA's, each of which securely provides its public key to some fraction of the users.



Trust tree:

- Trust extends from a number of root certificates.
- These certificates may certify certificates themselves, or they may certify certificates that certify still other certificates down some chain.
- The leaf certificate's validity is verified by tracing backward from its certifier, to other certifiers, until a directly trusted root certificate is found.

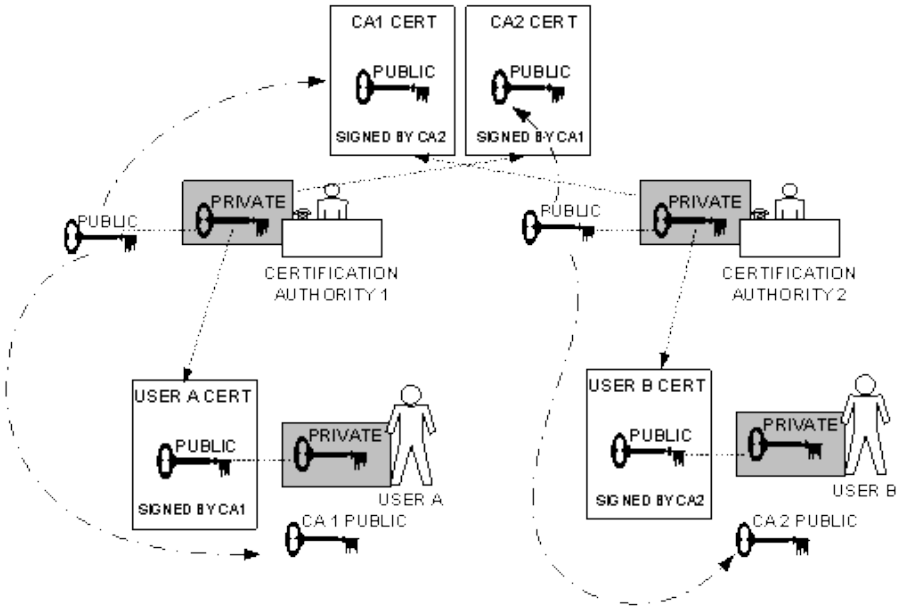


Hierarchical trust and cross-certification

- Suppose that A and B have obtained certificates from CAs X_1 and X_2, respectively.
  If A does not securely know X_2's public key, then A cannot validated B's certificate.
- Cross-certification: if the CAs have exchanged their own public keys, then A can obtain B's public key by a chain of certificates.

- A obtains, from the directory, the certificate of X_2 signed by X_1.

  A can thus get hold of X_2's public key (and verify it by means of X_1's signature on the certificate).
- A then goes back to the directory and obtains the certificate of B signed by X_2, which A can now verify with the trusted copy of X_2's public key.



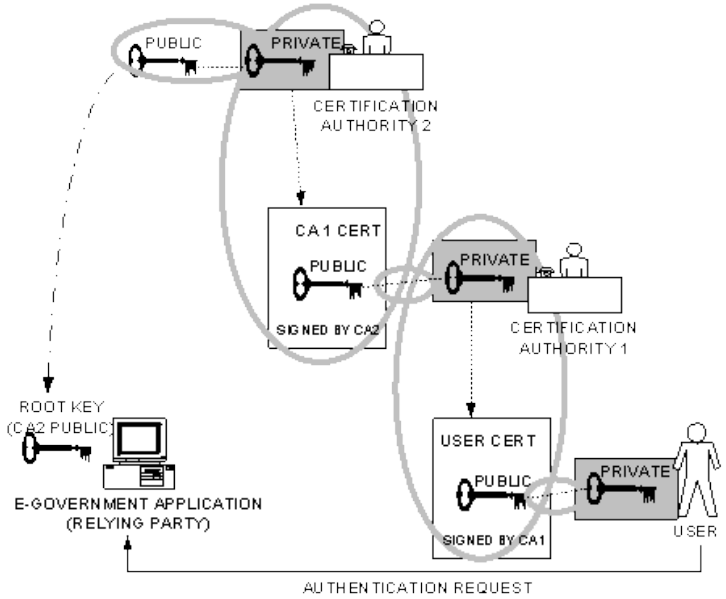
Hierarchical trust and cross-certification: X.509

- In X.509, this chain of certificates is expressed as

$$X_1 \ll X_2 \gg X_2 \ll B \gg$$

B obtains *A*'s public key with the reverse chain

$$X_2 \ll X_1 \gg X_1 \ll A \gg$$



- A chain with *n* elements is

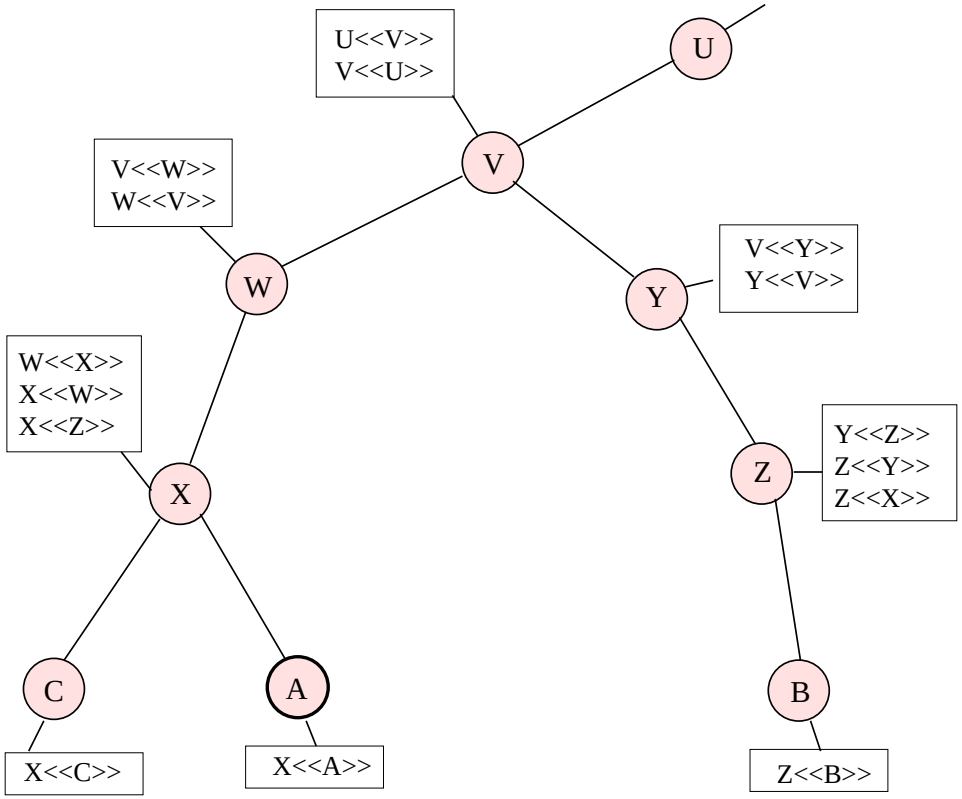
$$X_1 \ll X_2 \gg X_2 \ll X_3 \gg \ldots X_n \ll B \gg$$

where each pair (X_i, X_{i+1}) of CAs in the chain have created certificates for each other (stored in the directory).

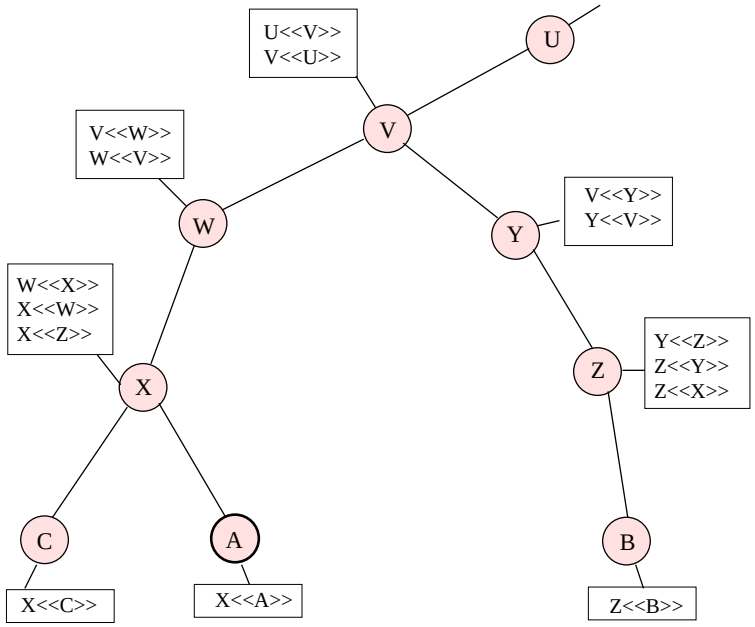
X.509 hierarchy

X.509 suggests that CAs be arranged in a hierarchy.

- Connected circles indicate hierarchical relationship between the CAs.
- Associated boxes indicate certificates maintained in the directory for each CA entry.
- Forward certificates: certificates of *X* generated by other CAs.
- Reverse certificates: certificates of other CAs generated by *X*.



X.509 hierarchy: example



User *A* can acquire the following certificates from the directory to establish a connection path to *B*:

$$X \ll W \gg \; W \ll V \gg \; V \ll Y \gg \; Y \ll Z \gg \; Z \ll B \gg$$

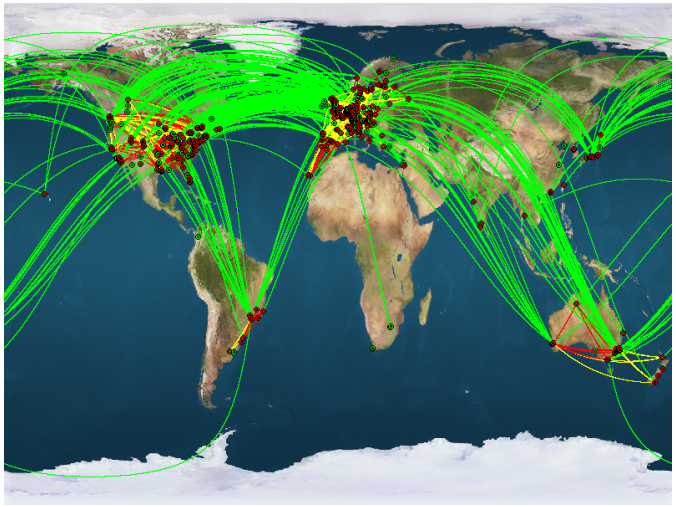
and then unwrap to obtain a trusted copy of *B*'s public key. Similarly, *B* can obtain *A*'s public key from

$$Z \ll Y \gg \; Y \ll V \gg \; V \ll W \gg \; W \ll X \gg \; X \ll A \gg$$

Trust models: web of trust

- Web of trust
  - encompasses direct and hierarchical trust,
  - adds the ideas that trust is in the eye of the beholder (which is the real-world view) and that more information is better.

- A certificate is trusted directly, or trusted in some chain going back to a directly trusted root certificate (the meta-introducer), or by some group of introducers.



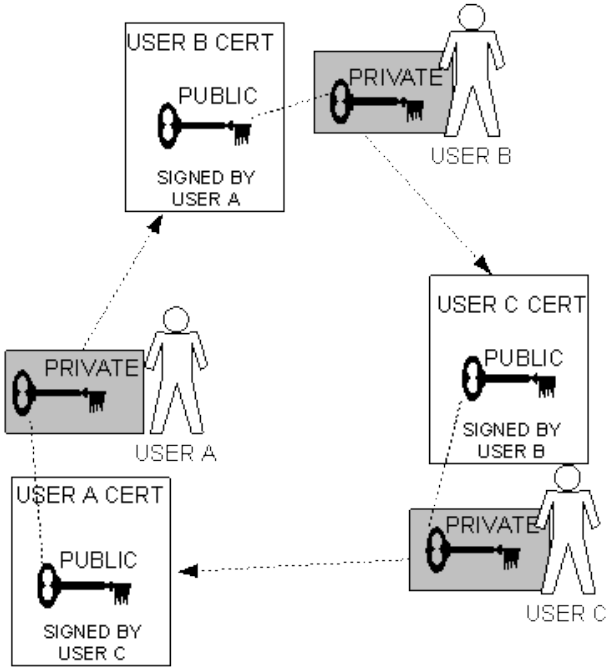
Web of trust: PGP

- PGP uses a certificate-based key management infrastructure for users' public keys.
- PGP certificates (and key management) differ from X.509 certificates in several important ways, e.g.
  - A PGP key may have multiple signatures (even "self-signing"). Each user creates and signs certificates for the people he or she knows (hence, no need for central infrastructure).
  - A notion of "trust" is embedded in each signature, and the signatures for a single key may have different levels of trust (and the users of a certificate act according to trust level).

Web of trust: PGP (cont.)

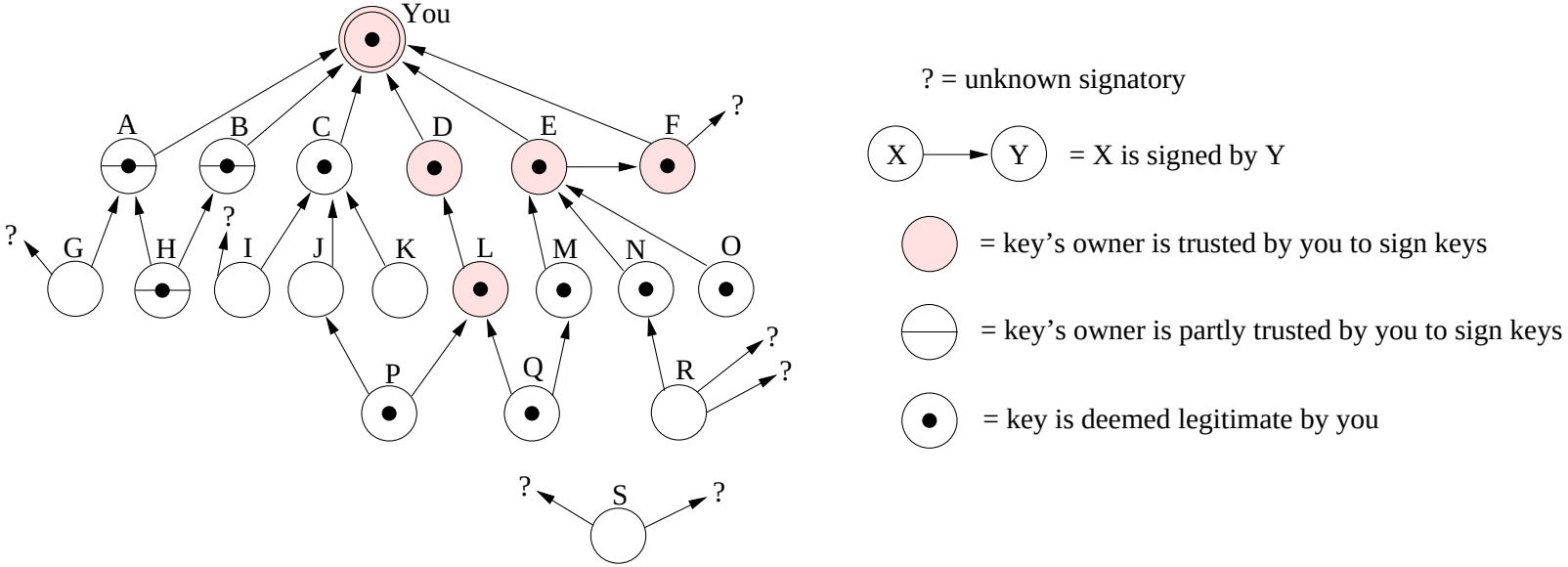
- In a PGP environment, any user can act as a certifying authority.
  - Digital signatures as form of introduction: when any user signs another's key, he or she becomes an introducer of that key.
  - As this process goes on, it establishes a web of trust.

- Any PGP user can validate another user's public key certificate, but such a certificate is only valid to another user if he recognizes the validator as a trusted introducer.
  - I.e. you trust my opinion that others' keys are valid only if you consider me to be a trusted introducer.
  - Otherwise, my opinion on other keys' validity is unimportant.



Web of trust: PGP (cont.)

- Stored on each user's public keyring are indicators of
  - whether or not the user considers a particular key to be valid,
  - the level of trust the user places on the key that the key's owner can serve as certifier of others' keys.
- You indicate, on your copy of my key, whether you think my judgement counts. It's really a reputation system: certain people are reputed to give good signatures, and people trust them to attest to other keys' validity.



PGP example

- Alice wants to communicate with Bob.
- She obtains Bob's public key PGP certificate, e.g.

$$\mathit{Ellen},\ \mathit{Fred},\ \mathit{Giselle},\ \mathit{Bob} \ll \mathit{Bob} \gg$$

- Alice knows none of the signers, so from a certificate server she gets Giselle's PGP certificate

$$\mathit{Henry},\ \mathit{Irene},\ \mathit{Giselle} \ll \mathit{Giselle} \gg$$

- She knows Henry vaguely, so to verify Giselle's certificate she obtains Henry's

$$\mathit{Ellen},\ \mathit{Henry} \ll \mathit{Henry} \gg$$

PGP example (cont.)

$$\textit{Ellen}, \textit{Fred}, \textit{Giselle}, \textit{Bob} \ll \textit{Bob} \gg$$
$$\textit{Henry}, \textit{Irene}, \textit{Giselle} \ll \textit{Giselle} \gg$$
$$\textit{Ellen}, \textit{Henry} \ll \textit{Henry} \gg$$

- She notes that Henry's signature is at the "casual" trust level, so she decides to look elsewhere for confirmation.
- She obtains Ellen's certificate

$$\textit{Jack}, \textit{Ellen} \ll \textit{Ellen} \gg$$

- She immediately recognizes Jack as her cousin and, since she has Jack's certificate, she uses it to validate Ellen's certificate: she notes that his signature is at the "positive" trust level, and so she accepts Ellen's certificate as valid and uses it to validate Bob's.
- She notes that Ellen signed the certificate with "positive" trust also, so she concludes that the certificate, and the public key it contains, are trustworthy.

PGP example (cont.)

- Alice followed two signature chains:

  $$\textit{Henry} \ll \textit{Henry} \gg \textit{Henry} \ll \textit{Giselle} \gg \textit{Giselle} \ll \textit{Bob} \gg$$
  $$\textit{Jack} \ll \textit{Ellen} \gg \textit{Ellen} \ll \textit{Bob} \gg$$

  (where the unchecked signatures have been dropped).
  The trust levels affected how Alice checked the certificate.
- Distinction between X.509 and PGP certificates:
  - X.509 certificates include an element of trust, but the trust is not indicated in the certificate.
  - PGP certificates indicate the level of trust, but the same level of trust may have different meanings to different signers (cf. also “name resolution”).

Table of contents I

Table of contents II

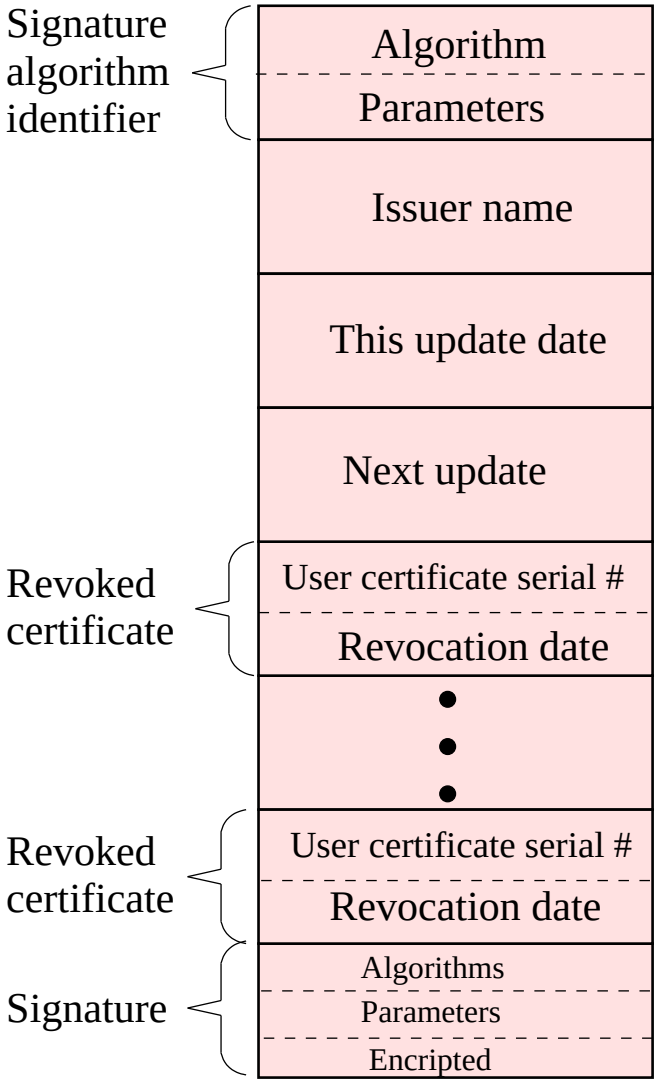
PKI – Key/certificate revocation

- Certificate Revocation List (CRL) signed and maintained by CA.
  - Posted on the directory.
  - Either clients check themselves actively (also with local caches), or use validation service that collects and checks CRLs centrally.
  - Each CA maintains a list of all revoked but not expired certificates issued by that CA (both to users and to other CAs).
- Reasons for revocation:
  - The user's private key is assumed to be compromised.
  - The user is no longer certified by the CA.
  - The CA's certificate is assumed to be compromised.
- X.509:
  - Each certificate includes a period of validity.
  - Typically, a new certificate is issued before the old one expires.

X.509 certificate revocation

Each CRL includes:

- the issuer's name,
- the date the CRL was created,
- the date the next CRL is scheduled to be issued,
- an entry for each revoked certificate.



PKI – Key recovery

- How can one recover a key that is lost, or if the people who know it are unable or unwilling to reveal it?
  Important, e.g., for keys belonging to roles.
- Three alternatives: either the key or the cryptosystem is weak, or a copy of the key can be placed somewhere.
- A key escrow system is a system which allows a third party to recover a cryptographic key.
  - For business (e.g. recovery of backup keys), or
  - law enforcement (recovery of keys used to encipher communications to which an authority requires access, such as enciphered letters or telephone messages).

PKI – Key recovery (cont.)

- A user can have multiple key pairs, e.g. for encryption and for signing (with/without non-repudiation characteristics).
  - Decryption key:
    - Backed-up inside the CA.
    - Can be recovered when lost.
    - Key escrow possible.
  - Signing key and non-repudiation key:
    - Back-up inside the CA not desirable.
    - Only exist on user's token.

Table of contents I

Table of contents II

Naming and Identity

Now what else is the whole life of mortals but a sort of comedy, in which the various actors, disguised by various costumes and masks, walk on and play each one his part, until the manager waves them off the stage? Moreover, this manager frequently bids the same actor to go back in a different costume, so that he who has but lately played the king in scarlet now acts the flunkey in patched clothes. Thus all things are presented by shadows.

Erasmus from Rotterdam, The Praise of Folly

- A **principal** is a unique entity. An **identity** specifies a principal.
- Authentication binds a principal to a representation of identity internal to the computer.
- Each system has its own way of expressing this representation, but all decisions of access control and resource allocation assume that the binding is correct.
  - Local objects identified by assigning names (e.g. to users, files...).
  - If the object resides on a different system, the name must encode the object's location (**uniform resource locator URL**, **uniform resource identifier URI**).

Names

A PKI binds Alice's public key to her name, but what is a name?

- In a small village...
  - Everybody knows everybody else by sight.
  - Everybody has a name that is either unique or is made unique (e.g. Big John and Little John).
  - For each name there is one person, but one person might have several names (e.g. Big John might also be called Sheriff).

  N.B.: a name is really any kind of data that is used to refer to an entity (the "official" name, e.g. birth name, is just one of many names).

Names (cont.)

- As the village grows into a town...
  - Names start to lose immediate association with a person (you no longer know all people). There might be a single J. Smith in town, but you might not know him.
  - Also: you start talking about people you have never actually met (e.g. the guy who just moved here and is going to sponsor the football team).
- As the town grows into a city...
  - You only know a small subset of the people.
  - Names are no longer unique and the meaning of a name may depend on context:
    - Alice knows three Johns, but when at work she talks about John it is clear from context that she means the John who is upstairs in sales.
    - Joe's poker buddies.

Names (cont.)

- Now consider the Internet...
  - E-mail addresses instead of traditional names.
    - `jsmith533@yahoo.com` is certainly a unique name, but in practice it does not link to a person in the sense of someone you will ever meet.
    - Even if you find out information such as his address and phone number, he is just as likely to live on the other side of the world.
  - Also: different online personalities and names are possible.

This could be very dangerous (lead to attacks)!
E.g.: `give@blood.org`

- Also: problems with name-sharing (e.g. people sharing an e-mail address) and naming for groups and roles.

Names (cont.)

- To have a bijection identifiers–persons, organizations and governments try to assign unique names to everybody, e.g. full name + date of birth + address + ...
  - But some of these data may change over time.
  - Also: some people are stateless and others have dual nationalities...
- Unique numbers are also employed (e.g., Tax Payer Id, Social Security Number, ...), but they have weaknesses and do not provide global coverage or global uniqueness.

Names in PKIs

- Policies for **name assignment**:
  - What forms do the names have?
  - Who does the assignment (in a trusted way)?
  - ...
- Policies for **name resolution**:
  - Who are Joe's poker buddies?
  - `give@blood.org` = `VladTepesDracula@transilvania.ro` ?
  - ...
- Groups and roles?
- Delegation of rights and privileges (indirect naming)?

Names in X.509

- X.509 certificates employ **distinguished names (DNs)** from the X.500 directory specification.

  Distinguished names are used to identify both issuers and subjects.

- DNs are also used in LDAP directories.

  (Lightweight Directory Access Protocol is an application protocol for querying and modifying directory services running over TCP/IP).

- Main criteria:
  - Permanence: DNs should not contain any volatile components.
  - Uniqueness: DNs should be unique, even over time.

Names in X.509 (cont.)

- Permanence:
  - *“old school":* cn=John Doe, ou=accounting, ou=finance, l=verona, c=it, o=mycompany
  - *better:* cn=John Doe, ou=people, dc=mycompany, dc=com

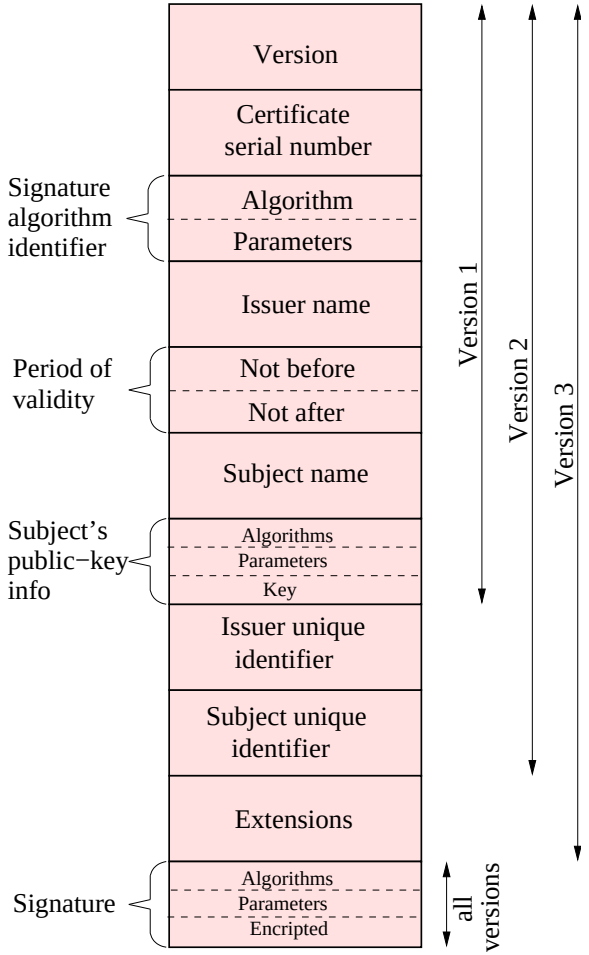
  (cn = common name, ou = organization unit, dc = domain component, l = locality, c = country, o = organization, uid = user id)
- Uniqueness: examples
  - cn=John Doe_1, ou=people, dc=mycompany, dc=com
  - cn=John Doe AB12345, ou=people, dc=mycompany, dc=com
  - uid=AB12345, ou=people, dc=mycompany, dc=com
  - uid=VR123456, ou=student, dc=univr, dc=it
- Note that the latter version might conflict with the requirements for *qualified certificates*.

Names in X.509 (cont.)

X.509 certificates employ **distinguished names**:

- Issuer name is X.500 name of CA that created and signed this certificate.
- Subject name is the name of the user to whom the certificate refers (i.e. the user whose public key is certified).

Optional strings subject unique identifier and issuer unique identifier in the event the X.500 names have been reused for different entities.



Names in X.509 (cont.)

CAs vouch, at some level, for the identity of the principals to whom the certificates are issued.

CA authentication policy : describes the level of authentication required to identify the principal to whom the certificate is to be issued.
Establishes the level of proof of identity needed for the CA to accept the principal's claim of identity.

CA issuance policy : describes the principals to whom the CA will issue certificates.
Given the identity of the principal, will the CA issue a certificate?

CAs should also prevent certificate conflicts (but both X.509 and PGP are silent about this).

Table of contents I

Table of contents II

Conclusions

- Symmetric/asymmetric cryptography ensure confidentiality.
  Assuming (un)conditional security and properly distributed keys.
- Asymmetric cryptography simplifies key distribution.
  - Simplifies logistics, due to substantially reduced number of keys.
  - But still need an authenticated channel to distribute keys.
- Implements message authentication, once keys distributed.
  - Public Key Infrastructures:
    - A PKI is an infrastructure that binds public keys to (representations of) a principal identities.
      Allows also for key revocation and recovery.
    - Different certificate schemes and different trust models.
    - Different representations of identity (“naming”).

Table of contents I

Table of contents II

Fast Exponentiation

- There is an efficient algorithm (cf. the literature) for computing powers modulo n in a monoid $G = (H, \circ, e)$ (where H is a set, $\circ$ is an associative operation on H, and $e \in H$ is a neutral element such that $e \circ a = a \circ e = a$ for all $a \in H$).
- Let $g \in G$ and e be a positive integer with binary expansion

$$e = \sum_{i=0}^{k} e_i 2^i \qquad \text{(observe that the coefficients } e_i \text{ are either 0 or 1)}$$

Then

$$g^e = g^{\sum_{i=0}^{k} e_i 2^i} = \prod_{i=0}^{k} (g^{2^i})^{e_i} = \prod_{0 \leq i \leq k,\, e_i = 1} g^{2^i}$$

Fast Exponentiation (cont.)

- $g^e = g^{\sum_{i=0}^{k} e_i 2^i} = \prod_{i=0}^{k} (g^{2^i})^{e_i} = \prod_{0 \leq i \leq k,\, e_i = 1} g^{2^i}$ yields the following idea:
  1. Compute the successive squares g^{2^i} for $0 \leq i \leq k$.
  2. Determine g^e as the product of those g^{2^i} for which $e_i = 1$.

  N.B.: we can compute $g^{2^{i+1}} = (g^{2^i})^2$ from g^{2^i} by one squaring.
- An example: $6^{73} \bmod 100$.
  - Binary expansion of exponent: $73 = 1 + 2^3 + 2^6$.
  - Determine successive squares of 6:

    $6^2 = 36$ $\qquad$ $6^{2^2} = 36^2 = -4 \bmod 100$

    $6^{2^3} = 16 \bmod 100$ $\qquad$ $6^{2^4} = 16^2 = 56 \bmod 100$

    $6^{2^5} = 56^2 = 36 \bmod 100$ $\qquad$ $6^{2^6} = -4 \bmod 100$

  $6^{73} = 6 \times 6^{2^3} \times 6^{2^6} = 6 \times 16 \times (-4) \bmod 100 = 16 \bmod 100$.

  Hence, 6 squares and 2 products instead of 72 multiplications modulo 100.

Fermat Test

It is expensive to prove that a given positive integer is prime, but there are efficient algorithms (primality tests) that prove the primality of a positive integer with high probability.
The Fermat Test is a primality test based on Fermat's theorem in the following version:

> *if n is a prime number, then $a^{n-1} =_n 1$ for all positive integers a such that a and n are relatively prime ($\gcd(a, n) = 1$).*

- Choose a positive integer $a \in \{1, 2, \ldots, n-1\}$.
- Compute $y = a^{n-1} \bmod n$ (e.g. using fast exponentiation).
- If $y \neq 1$, then n is composite (by the above theorem).
- If $y = 1$, then we do not know whether n is prime or composite, as the following example shows.

Fermat Test (cont.)

Consider $n = 341 = 11 \times 31$. Although n is composite we have

$$2^{340} =_{341} 1$$

Therefore, if we use Fermat Test with $n = 341$ and $a = 2$, then we obtain $y = 1$, which proves nothing.
On the other hand, if we use Fermat Test with $n = 341$ and $a = 3$, then n is proven composite as

$$3^{340} =_{341} 56$$

N.B.: if the Fermat Test proves that n is composite, it does not find a divisor of n. It only shows that n lacks a property that all prime numbers have. Therefore, the Fermat Test cannot be used as a factoring algorithm.
Exercise: Use Fermat Test to show that 1111 is not a prime number.
Additional knowledge: Look up Miller-Rabin's Test.

Bibliography

Most of the figures in this lecture are taken from:

- William Stallings. *Cryptography and Network Security. Principles and Practice*, 7th ed., Prentice Hall, 2016.

Other interesting sources:

- The International PGP Home Page: `http://www.pgpi.org/`
- SDSI/SPKI (and PKI and PGP): `http://world.std.com/~cme/html/spki.html`
- Dieter Gollmann. *Computer Security*. Wiley, 2011.
- Bruce Schneier. *Applied Cryptography*, John Wiley & Sons, 1996 (and 20th anniversary edition in 2016).
- Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. Available online.
- Arthur E. Hutt, Seymour Bosworth, Douglas B. Hoyt. *Computer Security Handbook*. Wiley, 1995.