UNIVERSITÀ DI PARMA

# Public Key Cryptography

***Tecnologie Internet***
a.a. 2022/2023

## Summary

- Principles of Public-Key Cryptography
- Public-Key Cryptanalysis
- RSA
- Diffie-Hellman Key Exchange

## Principles of Public-Key Cryptography

Public-key cryptography is the greatest and perhaps the only true revolution in the entire history of cryptography.

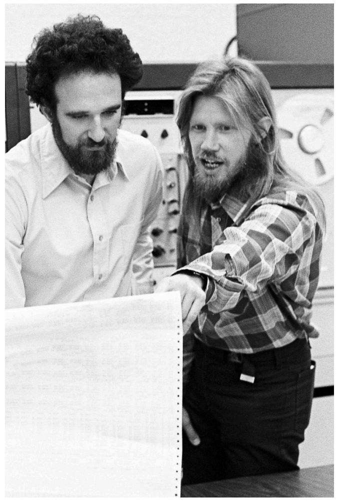
Public-key algorithms are based on **mathematical functions** rather than on substitution and permutation.

More important, public-key cryptography is **asymmetric**, involving the use of two separate keys (a **public key** and a **private key**).

Beware the following false beliefs:

- public-key encryption is more secure from cryptanalysis than is symmetric encryption (no!)
- public-key encryption is a general-purpose technique that has made symmetric encryption obsolete (no!)

## Principles of Public-Key Cryptography



Diffie and Hellman invented the concept of public-key cryptography in 1976.

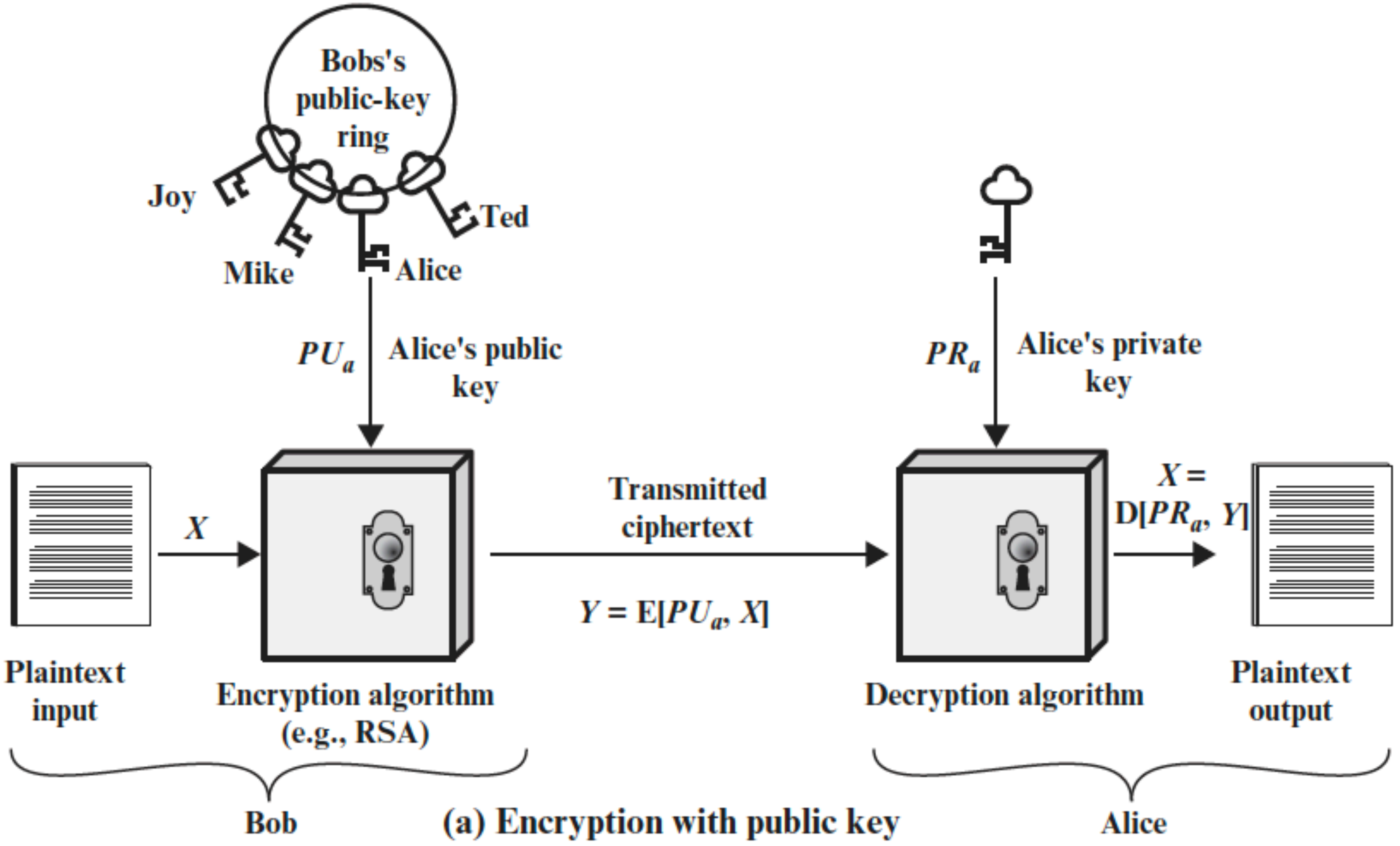
As practical implementations (e.g., RSA) are computationally expensive, public-key cryptography is restricted to two main problems:

- **key distribution**
- **digital signatures**

Main features:

- it is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key
- either of the two related keys can be used for encryption, with the other used for decryption

## Public-Key Cryptography for Confidentiality



(a) Encryption with public key

## Public-Key Cryptography for Confidentiality

Essential steps:

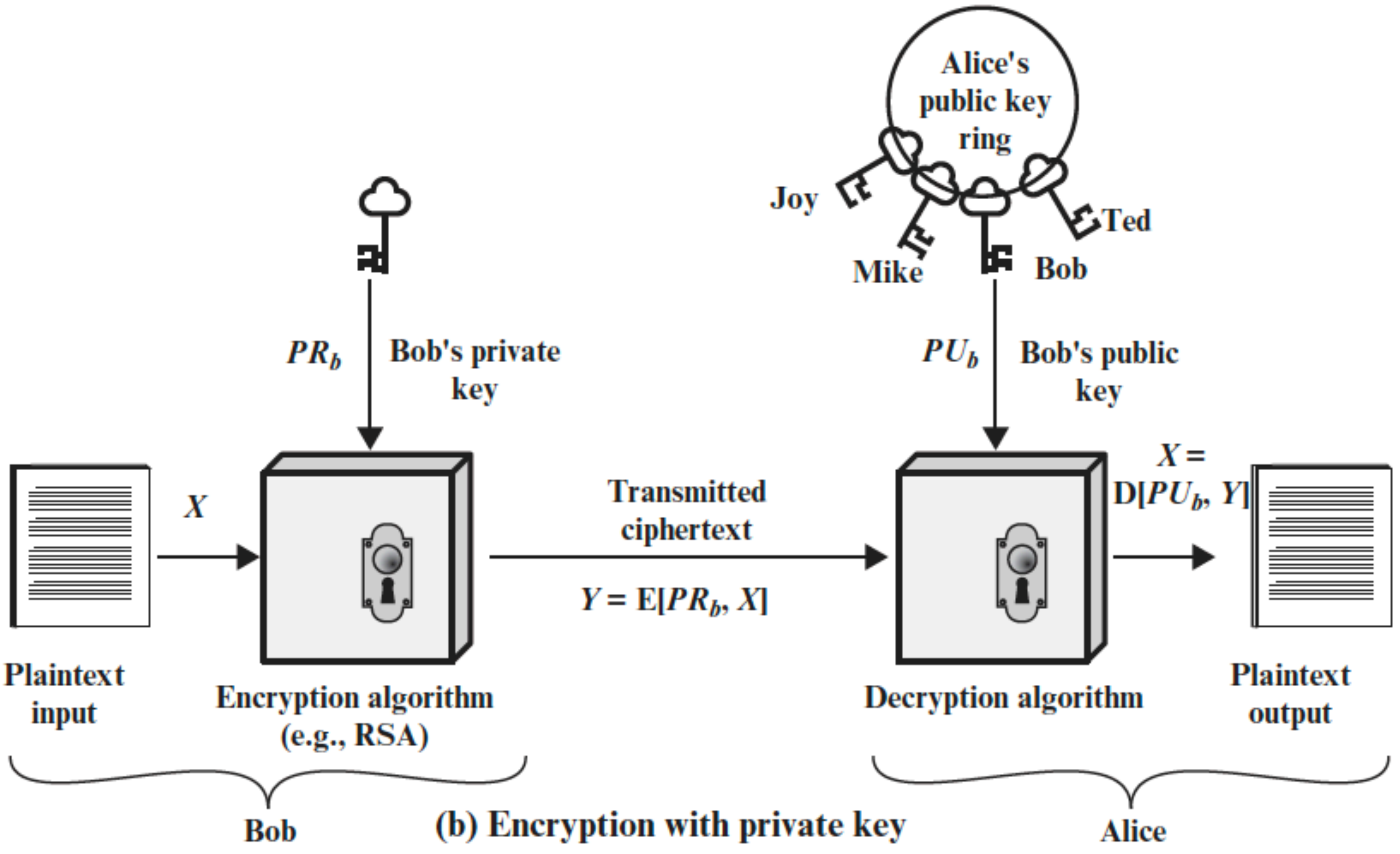
1. Each user generates a pair of keys to be used for the encryption and decryption of messages.

2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private.

3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key: $Y = \mathrm{E}(PU_a, X)$

4. When Alice receives the message, she decrypts it using her private key: $X = \mathrm{D}(PR_a, Y)$. No other recipient can decrypt the message because only Alice knows Alice's private key.

## Public-Key Cryptography for Authentication



## Public-Key Cryptography for Authentication

In this case, Bob prepares a message for Alice and encrypts it using his private key before transmitting it: $Y = \mathrm{E}(PR_b, X)$

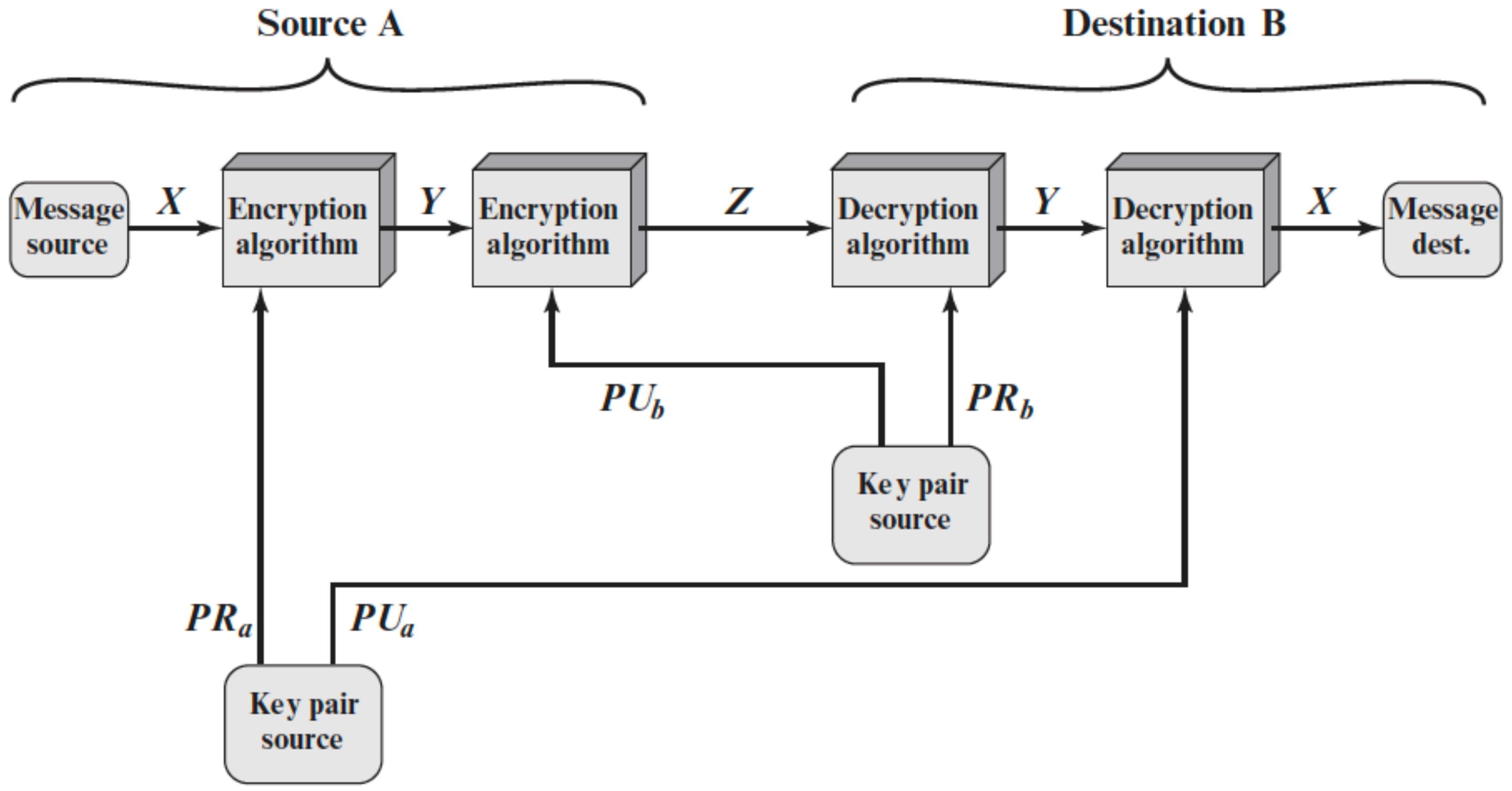
Alice can decrypt the message using Bob's public key: $X = \mathrm{D}(PU_b, Y)$. Because the message was encrypted using Bob's private key, only Bob could have prepared the message.

Therefore, the entire encrypted message serves as a **digital signature**.

In addition, it is impossible to alter the message without access to Bob's private key, so the message is authenticated both in terms of data origin and in terms of data integrity.

More efficiently: **encrypt a small block of bits** (called an **authenticator**) that is a function of the document.

## Public-Key Cryptography for Confidentiality and Authentication



$$Z = E(PU_b, E(PR_a, X))$$
$$X = D(PU_a, D(PR_b, Z))$$

## Requirements for Public-Key Cryptography

1. It is computationally easy for Alice / Bob to generate its key pair

2. It is computationally easy for Alice, knowing Bob's public key and the message *M* to be encrypted, to generate the corresponding ciphertext: $C = \mathrm{E}(PU_b, M)$

3. It is computationally easy for Bob to decrypt the ciphertext using the private key to recover the original message: $M = \mathrm{D}(PR_b, C) = \mathrm{D}(PR_b, \mathrm{E}(PU_b, M))$

4. It is computationally infeasible for an adversary, knowing the public key, $PU_b$, to determine the private key, $PR_b$.

5. It is computationally infeasible for an adversary, knowing the public key, $PU_b$, and a ciphertext, *C*, to recover the original message *M*.

## Requirements for Public-Key Cryptography

An **easy problem** is a problem that can be solved in **polynomial time** as a function of input length.

An **infeasible problem** is a problem such that the effort to solve it grows **faster than polynomial time** as a function of input size.

Only a few algorithms (RSA, elliptic curve cryptography, Diffie–Hellman, DSS) have received widespread acceptance in the several decades since the concept of public-key cryptography was proposed.

All these algorithms are based on a **trapdoor one-way function**, i.e., a family of invertible functions $f_k$ such that

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

## Public-Key Cryptanalysis

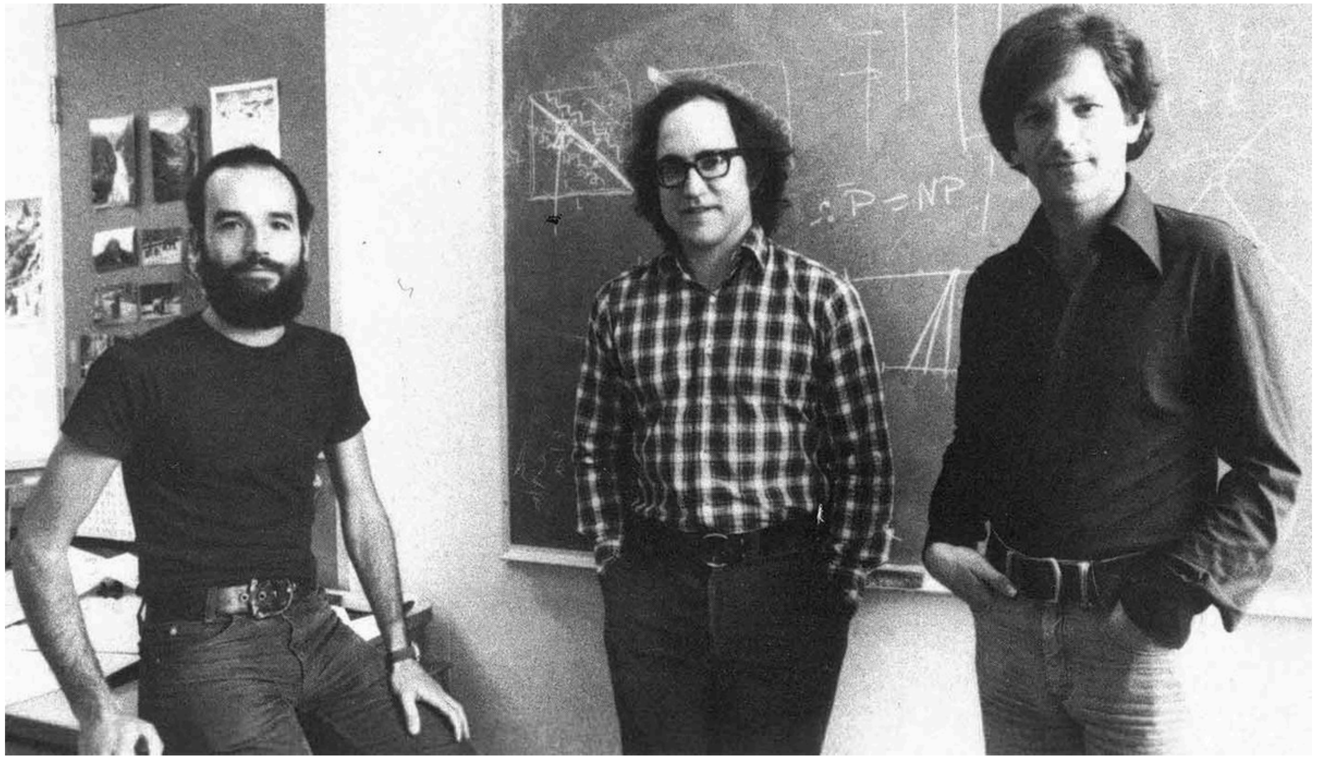
1) The countermeasure against **brute-force attacks** is to **use large keys**. Unfortunately, large keys make public-key encryption/ decryption too slow for general-purpose use.

2) Find some way to **compute the private key given the public key**. To date, it has not been mathematically proven that this form of attack is infeasible for a particular public-key algorithm. Thus, any given algorithm, including the widely used RSA algorithm, is suspect.

3) **Probable message attack**. Suppose, for example, that a message were to be sent that consisted solely of a 56-bit DES key. An adversary could encrypt all possible 56-bit DES keys using the public key and could discover the encrypted key by matching the transmitted ciphertext. Countermeasure: append some random bits to such simple messages.

## RSA

One of the first successful responses to the challenge was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and first published in 1978.



## RSA

Plaintext is encrypted in blocks, with each block having size $i$ such that $2^i < n \leq 2^{i+1}$, for some number $n$.

Plaintext block $M$ and ciphertext block $C$.

Encryption: $\boldsymbol{C = M^e \bmod n}$

Decryption: $\boldsymbol{M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n}$

Both sender and receiver must know the value of $n$.
The sender knows the value of $e$, and only the receiver knows the value of $d$.
Public key of the receiver: $\boldsymbol{PU = \{e,n\}}$
Private key of the receiver: $\boldsymbol{PR = \{d,n\}}$

## RSA

How to get $e,d,n$ such that $\boldsymbol{M^{ed}}$ **mod** $\boldsymbol{n = M}$ for all $M<n$?

Pick $\boldsymbol{p,q}$ **primes** and keep them secret.

Calculate $n = pq$.

Calculate $\boldsymbol{\phi(n) = (p-1)(q-1)}$ and keep it secret.

Choose $e$, with $\mathbf{GCD}(\boldsymbol{\phi(n),e}) \mathbf{= 1}$ and $1 < e < \phi(n)$.

Calculate $\boldsymbol{d \equiv e^{-1}}$ **(mod** $\boldsymbol{\phi(n)}$**)** and keep it secret.

Knowing $e$ and $n$, but not the “trapdoor key” $\phi(n)$, it should be infeasible to compute $d$.

## Example

1. Select two prime numbers, $p = 17$ and $q = 11$.

2. Calculate $n = pq = 17 \times 11 = 187$.

3. Calculate $\phi(n) = (p - 1)(q - 1) = 16 \times 10 = 160$.

4. Select $e$ such that $e$ is relatively prime to $\phi(n) = 160$ and less than $\phi(n)$; we choose $e = 7$.

5. Determine $d$ such that $de \equiv 1 \pmod{160}$ and $d < 160$. The correct value is $d = 23$, because $23 \times 7 = 161 = (1 \times 160) + 1$.

Thus, $PU = \{7,187\}$ and $PR = \{23,187\}$.

## Computational Aspects

Both encryption and decryption in RSA involve **raising an integer to an integer power, mod *n***.

If the exponentiation was done over the integers and then reduced modulo $n$, the intermediate values would be enormous.

Fortunately, we can make use of a property of modular arithmetics:

$$[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$$

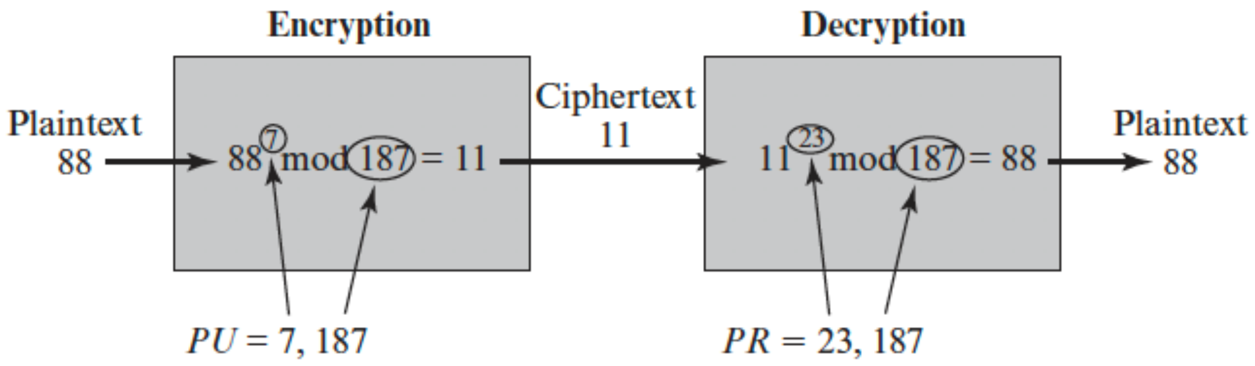
As a consequence:

$$a^b \bmod n = \left[\prod_{b_i \neq 0} a^{(2^i)}\right] \bmod n = \left(\prod_{b_i \neq 0}\left[a^{(2^i)} \bmod n\right]\right) \bmod n$$

## Computational Aspects

Example: using $PU = \{7,187\}$ and $PR = \{23,187\}$, encrypt $M = 88$.

$$\begin{aligned} C &= 88^7 \bmod 187 \\ &= [(88^4 \bmod 187) \times (88^2 \bmod 187) \times (88^1 \bmod 187)] \bmod 187 \\ &= (132 \times 77 \times 88) \bmod 187 \\ &= 11 \end{aligned}$$



Now decrypt $C$.

$$\begin{aligned} M &= 11^{23} \bmod 187 \\ &= [(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \\ &\quad \times (11^8 \bmod 187) \times (11^8 \bmod 187)] \bmod 187 \\ &= (11 \times 121 \times 55 \times 33 \times 33) \bmod 187 \\ &= 88 \end{aligned}$$

## Computational Aspects

In order to prevent the discovery of *p* and *q* by exhaustive methods, these primes must be chosen from a sufficiently large set (i.e., *p* and *q* must be large numbers).

On the other hand, the method used for finding large primes must be reasonably efficient.

At present, there are no useful techniques that yield arbitrarily large primes, so some other means of tackling the problem is needed.

The procedure that is generally used is to pick at random an odd number of the desired order of magnitude and **test whether that number is prime**. If not, pick successive random numbers until one is found that tests prime.

Manindra et al., "PRIMES is in P", 2002

## Computational Aspects

How many numbers are likely to be rejected before a prime number is found?

**Prime number theorem (PNT)**: the primes near $x$ are spaced on the average one every $\ln(x)$ integers.

Thus, on average, one would have to test about $\boldsymbol{\ln(x)/2}$ integers before a prime is found.

Example: for a prime on the order of magnitude of $2^{200}$, about 70 trials are needed.

## Computational Aspects

Having determined *p* and *q*, we need to find *e* and *d*.

To this purpose, we can generate a series of random numbers and test each of them against $\phi(n)$ until a number relatively prime to $\phi(n)$ is found, which is *e*.

Then, *d* can be calculated by means of the **extended Euclid's algorithm**.

It can be shown that the probability that two numbers are relatively prime is about 0.6; thus, very few tests would be needed to find a suitable integer.

## Computational Aspects

To speed up the operation of the RSA algorithm using the public key, **a specific choice of e is usually made**. The most common choice is *65537* ($2^{16} + 1$); two other popular choices are *3* and *17*.

If a value of *e* is selected first and the primes *p* and *q* are generated, it may turn out that $GCD(\phi(n), e) \neq 1$. In that case, the user must reject the *p*, *q* values and generate a new *p*, *q* pair.

We cannot similarly choose a small constant value of *d* for efficient operation. A small value of *d* is vulnerable to a brute-force attack and to other forms of cryptanalysis.

## The Security of RSA

Five possible approaches to attacking the RSA algorithm:

- Brute force
- Mathematical attacks
- Timing attacks
- Hardware fault-based attacks
- Chosen ciphertext attacks

Defense against brute-force attacks: use a large key space.

**The larger the number of bits of *d*, the better (but also, the slower the system will run).**

## The Security of RSA

All mathematical approaches to attacking RSA are equivalent to **factoring *n***.

“Can integer factorization be solved in polynomial time on a classical computer?” (unsolved problem in computer science)

Current best factorization algorithm is the **Generalized Number Field Sieve (GNFS)**, whose running time is $O(exp(cN^{1/3}(\log N)^{2/3}))$ for factoring an integer consisting of $N$ bits.

**Key length**: the number of bits for *n*. NIST recommends using 2048 bits, which should be safe until year 2030.

**Shor’s factorization algorithm** for quantum computers has running time $O(N^3)$, but requires several thousand qubits to factor RSA integers.

## The Security of RSA

To avoid values of $n$ that may be factored more easily, the following constraints on $p$ and $q$ are suggested:

1. **$p$ and $q$ should differ in length by only a few digits**

Example: if $n$ is 1024 bits long, $p$ and $q$ should be on the order of magnitude of $10^{75}$ to $10^{100}$.

2. **Both ($p$-1) and ($q$-1) should contain a large prime factor**

3. **GCD($p$-1,$q$-1) should be small**

Moreover, it has been demonstrated that if $e < n$ and $d < n^{1/4}$, then $d$ can be easily determined.

## The Security of RSA

Paul Kocher, a cryptographic consultant, demonstrated that a snooper can determine a private key by keeping track of how long a computer takes to decipher messages (*timing attack*).

There are simple countermeasures, including the following.

- **Constant exponentiation time**: Ensure that all exponentiations take the same amount of time before returning a result.
- **Random delay**: Add a random delay to the exponentiation algorithm to confuse the timing attack.
- **Blinding**: Multiply the ciphertext by a random number before performing exponentiation.

UNIVERSITÀ DI PARMA

## The Security of RSA

RSA Security (www.rsa.com) incorporates the following blinding feature into some of its products, to implement the private-key operation $M = C^d \bmod n$.

1. Generate a secret random number $r$ between 0 and $n$-1.
2. Compute $C' = Cr^e \bmod n$, where $e$ is the public exponent.
3. Compute $M' = (C')^d \bmod n$ with the ordinary RSA implementation.
4. Compute $M = M'r^{-1} \bmod n$.

The last step is correct, because $r^{ed} \bmod n = r \bmod n$.

## The Security of RSA

**Hardware fault-based attacks** require that the attacker has physical access to the target machine and that the attacker is able to directly control the input power to the processor.

The attack induces faults in the signature computation by reducing the power to the processor. The faults cause the software to produce invalid signatures, which can then be analyzed by the attacker to recover the private key.

Such an analysis has been demonstrated by extracting a 1024-bit private RSA key in approximately 100 hours, using a commercially available microprocessor.

## The Security of RSA

In the **chosen ciphertext attack (CCA)**, the adversary exploits properties of RSA and selects blocks of data that, when processed using the target's private key, yield information needed for cryptanalysis.

For example, consider the following property of RSA:

$$\mathrm{E}(PU, M_1) \times \mathrm{E}(PU, M_2) = \mathrm{E}(PU, [M_1 \times M_2])$$

Decrypt $C = M^e \bmod n$ using CCA as follows:

1. Compute $X = (C \times 2^e) \bmod n$.
2. Submit $X$ and receive back $Y = X^d \bmod n$.

Note that $X = (2M)^e \bmod n$; therefore, $Y = (2M) \bmod n$.
From this, we can deduce $M$.

## The Security of RSA

To overcome CCA, practical RSA-based cryptosystems randomly pad the plaintext prior to encryption.

RSA Security recommends modifying the plaintext using a procedure known as **optimal asymmetric encryption padding (OAEP)**.

## Diffie-Hellman Key Exchange

A number of commercial products employ this key exchange technique, proposed by Diffie and Hellman in 1976.

The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent symmetric encryption of messages.

It is less general than RSA: it does neither encryption nor signature.

The Diffie–Hellman algorithm depends for its effectiveness on the difficulty of computing **discrete logarithms**.

## Diffie-Hellman Key Exchange

A **primitive root** of a prime number *p* is one whose powers modulo *p* generate all the integers from 1 to *p*-1.

If *a* is a primitive root of the prime number *p*, then the numbers

$$a \bmod p,\ a^2 \bmod p,\ \ldots,\ a^{p-1} \bmod p$$

are distinct integers from 1 through *p*-1 in some permutation.

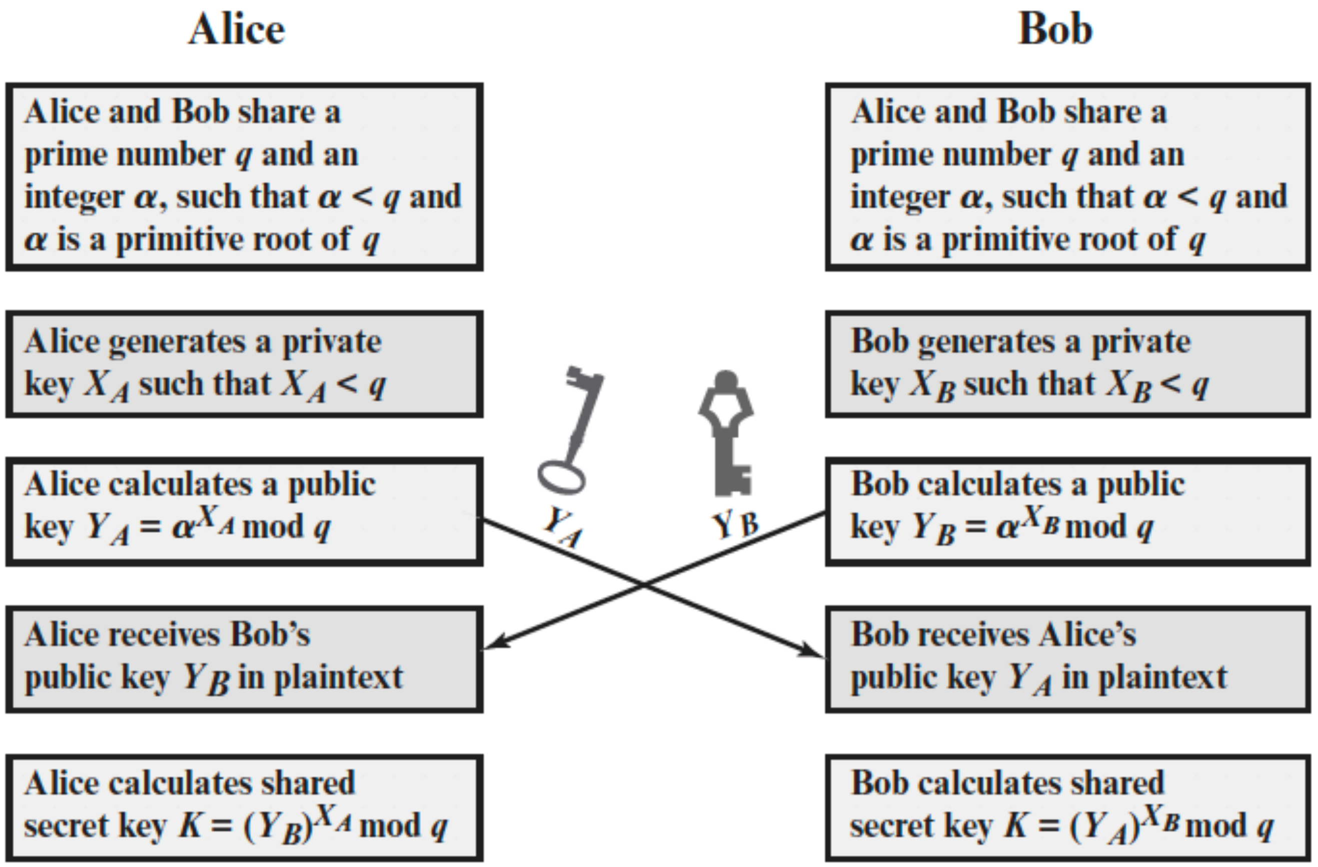
For any integer *b* and a primitive root *a* of prime number *p*, we can find a unique exponent *i* such that

$$b \equiv a^i \pmod{p} \qquad \text{where } 0 \le i \le (p - 1)$$

The exponent *i* is referred as the **discrete logarithm** of *b* for the base *a*, mod *p*.

## Diffie-Hellman Key Exchange

There are two publicly known numbers: a prime number $q$ and an integer $\alpha$ that is a primitive root of $q$.


Alice
Bob
Alice and Bob share a prime number $q$ and an integer $\alpha$, such that $\alpha < q$ and $\alpha$ is a primitive root of $q$
Alice and Bob share a prime number $q$ and an integer $\alpha$, such that $\alpha < q$ and $\alpha$ is a primitive root of $q$
Alice generates a private key $X_A$ such that $X_A < q$
Bob generates a private key $X_B$ such that $X_B < q$
Alice calculates a public key $Y_A = \alpha^{X_A} \bmod q$
Bob calculates a public key $Y_B = \alpha^{X_B} \bmod q$
$Y_A$
$Y_B$
Alice receives Bob's public key $Y_B$ in plaintext
Bob receives Alice's public key $Y_A$ in plaintext
Alice calculates shared secret key $K = (Y_B)^{X_A} \bmod q$
Bob calculates shared secret key $K = (Y_A)^{X_B} \bmod q$

## Diffie-Hellman Key Exchange

The result is that Alice and Bob have exchanged a secret value, which can be used as shared symmetric secret key.

By the rules of modular arithmentic:

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

## Diffie-Hellman Key Exchange

Example:

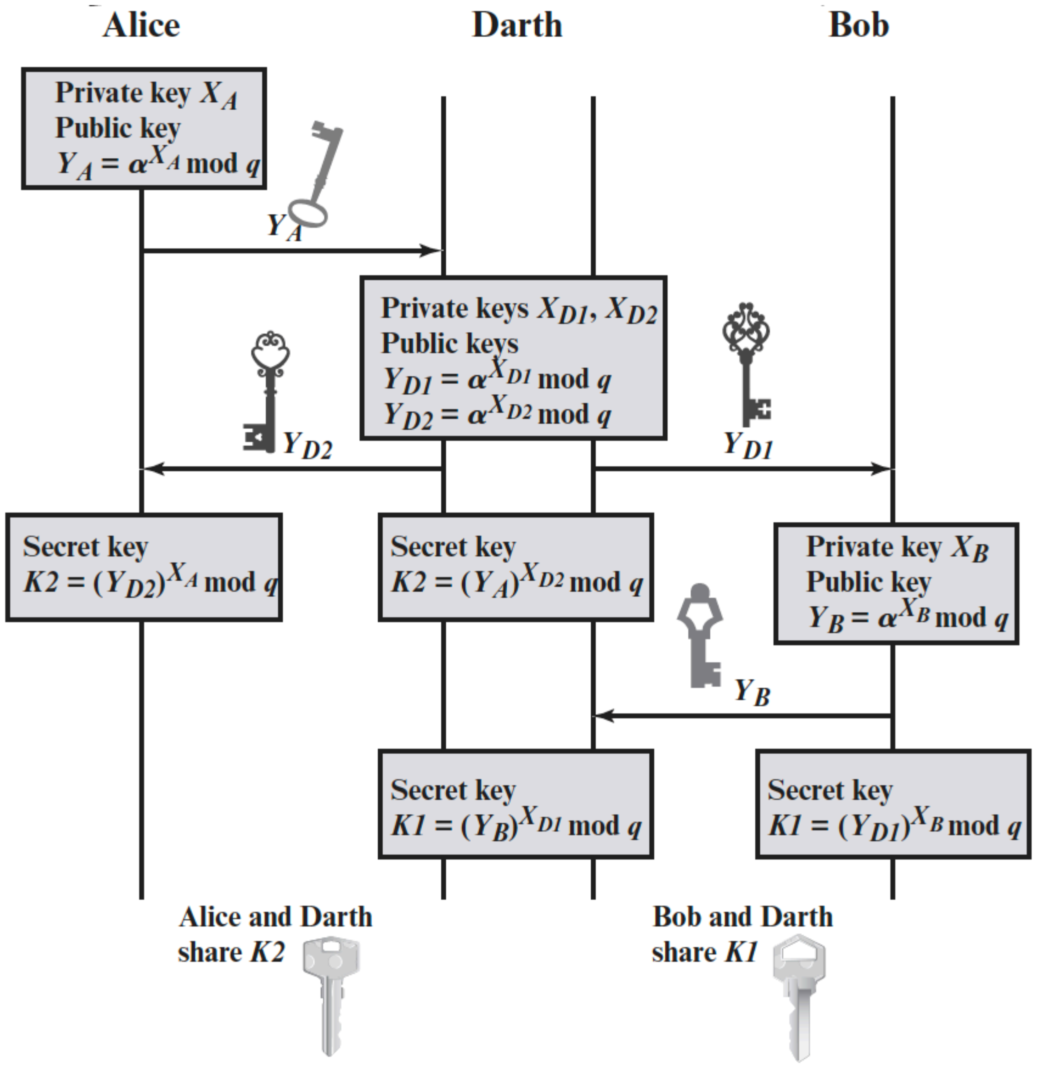
$q = 335$, $\alpha = 3$
$X_A = 97$, $X_B = 233$

$Y_A = 3^{97} \bmod 335 = 40$
$Y_B = 3^{233} \bmod 335 = 248$

$K = 248^{97} \bmod 335 = 40^{233} \bmod 335 = 160$

The security of the Diffie-Hellman key exchange lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms. For large primes, the latter task is considered infeasible.

## Man-in-the-Middle Attack


Alice
Darth
Bob
Private key $X_A$
Public key
$Y_A = \alpha^{X_A} \bmod q$
$Y_A$
Private keys $X_{D1}$, $X_{D2}$
Public keys
$Y_{D1} = \alpha^{X_{D1}} \bmod q$
$Y_{D2} = \alpha^{X_{D2}} \bmod q$
$Y_{D2}$
$Y_{D1}$
Secret key
$K2 = (Y_{D2})^{X_A} \bmod q$
Secret key
$K2 = (Y_A)^{X_{D2}} \bmod q$
Private key $X_B$
Public key
$Y_B = \alpha^{X_B} \bmod q$
$Y_B$
Secret key
$K1 = (Y_B)^{X_{D1}} \bmod q$
Secret key
$K1 = (Y_{D1})^{X_B} \bmod q$
Alice and Darth share $K2$
Bob and Darth share $K1$

## Man-in-the-Middle Attack

All future communication between Bob and Alice is compromised in the following way.

1. Alice sends an encrypted message *M*: E(*K2*, *M*).
2. Darth intercepts the encrypted message and decrypts it to recover *M*.
3. Darth sends Bob E(*K1*, *M*) or E(*K1*, *M′*), where *M′* is any message. In the first case, Darth simply wants to eavesdrop on the communication without altering it. In the second case, Darth wants to modify the message going to Bob.

## Man-in-the-Middle Attack

The Diffie-Hellman key exchange protocol 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**.

## References

William Stallings, ***Cryptography and Network Security - Principles and Practice***, 7th edition, Pearson 2017

Moriarty et al., **PKCS #1: RSA Cryptography Specifications Version 2.2**, RFC 8017, 2016