

# Asymmetric Ciphers II

Computer and Network Security

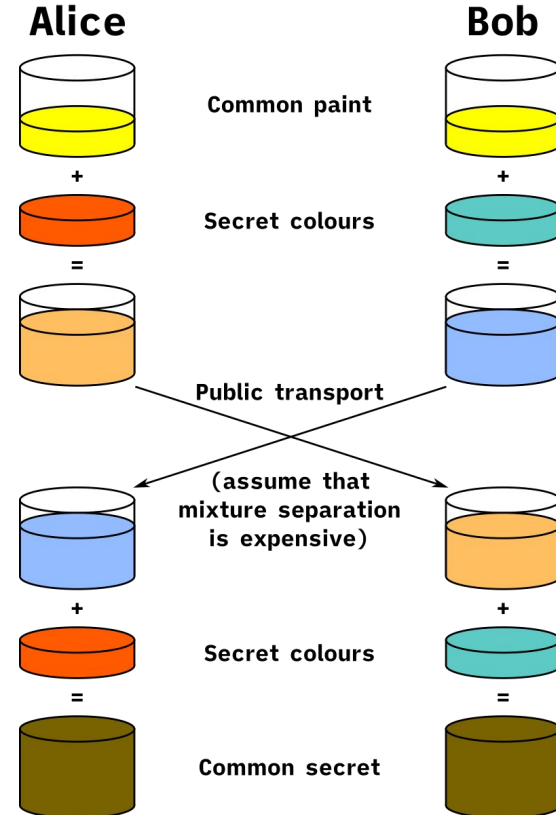
Emilio Coppa

# Diffie and Hellman (DH) Algorithm for Key Exchange

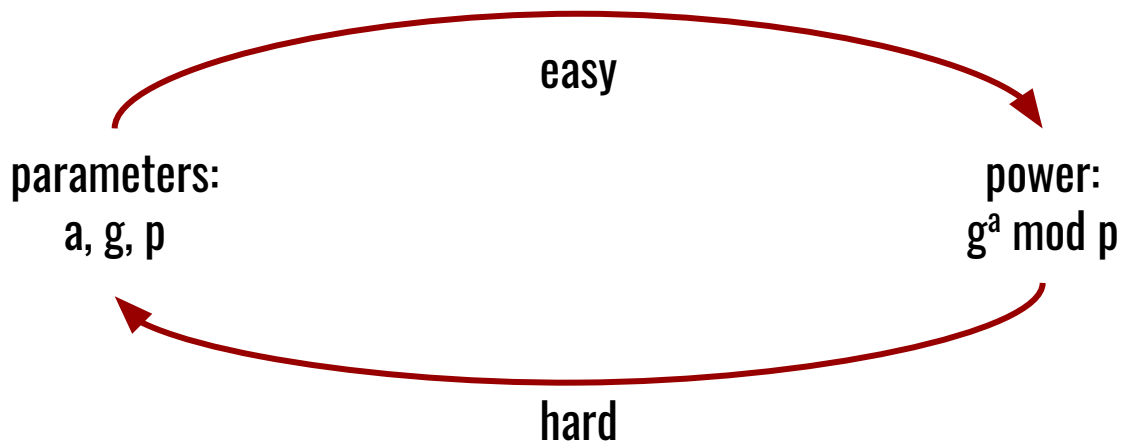
DH is one of the first PK schemes (1976).

Visually:

where mixture separation is based on DLP.



# DH



This PK scheme is based on the discrete logarithm problem. There is no trapdoor in DH.

# Diffie and Hellman (DH) Algorithm for Key Exchange (2)

Public parameters:

1. a **prime**  $p$
2. a **generator**  $g$  of  $\mathbb{Z}_p^*$

Alice

$$a \in [1, p - 1]$$

$$g^a \bmod p$$

Bob

$$b \in [1, p - 1]$$

$$g^b \bmod p$$

$$g^a \bmod p$$

$$g^b \bmod p$$

$$(g^b)^a \bmod p = g^{ab} \bmod p$$

$$(g^a)^b \bmod p = g^{ab} \bmod p$$

# Diffie and Hellman (DH) Algorithm for Key Exchange (3)

- $p$  and  $g$  are public
- $a$  and  $b$  are the private keys
- $g^a \bmod p$  and  $g^b \bmod p$  are the public keys
- $k = g^{ab} \bmod p$  is a secret shared between Alice and Bob

DH is not used for encryption but for key exchange. Given  $k$ , we can perform encryption with a symmetric scheme (e.g., AES).

# Small subgroup confinement attack

If we just consider any generator of  $\mathbb{Z}_p^*$  then an attacker can look at the subgroups generated by  $g^a \bmod p$  and  $g^b \bmod p$  and easily restrict the set of “possible” keys when one of these two subgroups is small.

E.g.,  $\mathbb{Z}_{19}^*$ ,  $g = 2$ ,  $a = 6$ ,  $b = 11$

we have that:

$$\forall x : (g^a \bmod p)^x = (2^6 \bmod 19)^x = 7^x = \{1, 7, 11\}$$

hence, we can easily get that the key  $(g^a)^b \bmod p = g^{ab} \bmod p$  can be only one of  $\{1, 7, 11\}$

In practice, we care also for properties on the subgroups of  $\mathbb{Z}_p^*$

# How “small” are the subgroups?

A property of cyclic groups is that  $\text{ord}(a)$  divides  $|G|$ .

If  $p$  is prime, then  $|G| = p-1$ , which is an even number! Hence, we have, e.g., subgroups of size:

- 1
- 2
- $(p-1)/2$
- $p-1$
- ...any other divisor of  $|G|$

## E.g., subgroups for a group with $p=19$

```
1: # 1: 1, 1
2: #18: 1, 2, 4, 8, 16, 13, 7, 14, 9, 18, 17, 15, 11, 3, 6, 12, 5, 10, 1
3: #18: 1, 3, 9, 8, 5, 15, 7, 2, 6, 18, 16, 10, 11, 14, 4, 12, 17, 13, 1
4: # 9: 1, 4, 16, 7, 9, 17, 11, 6, 5, 1
5: # 9: 1, 5, 6, 11, 17, 9, 7, 16, 4, 1
6: # 9: 1, 6, 17, 7, 4, 5, 11, 9, 16, 1
7: # 3: 1, 7, 11, 1
8: # 6: 1, 8, 7, 18, 11, 12, 1
9: # 9: 1, 9, 5, 7, 6, 16, 11, 4, 17, 1
10: #18: 1, 10, 5, 12, 6, 3, 11, 15, 17, 18, 9, 14, 7, 13, 16, 8, 4, 2, 1
11: # 3: 1, 11, 7, 1,
12: # 6: 1, 12, 11, 18, 7, 8, 1
13: #18: 1, 13, 17, 12, 4, 14, 11, 10, 16, 18, 6, 2, 7, 15, 5, 8, 9, 3, 1
14: #18: 1, 14, 6, 8, 17, 10, 7, 3, 4, 18, 5, 13, 11, 2, 9, 12, 16, 15, 1
15: #18: 1, 15, 16, 12, 9, 2, 11, 13, 5, 18, 4, 3, 7, 10, 17, 8, 6, 14, 1
16: # 9: 1, 16, 9, 11, 5, 4, 7, 17, 6, 1
17: # 9: 1, 17, 4, 11, 16, 6, 7, 5, 9, 1
18: # 2: 1, 18, 1
```



# How to avoid “small” subgroups?

We can require that the prime  $p$  defining the group is a **safe-prime**, i.e.,  $p = 2q + 1$  where  $q$  is a prime.

Under this assumption, we have that the divisors of  $|G| = p-1 = 2q + 1 - 1 = 2q$  are only:

- 1
- 2
- $q$
- $2q$

Hence, we should define rules to avoid groups of size 1 and size 2 and require that  $q$  is very large, i.e., more than 1024 bits.

# Subgroups of order 2

Informally, if  $r$  is a prime divisor of the group order, then there will be a subgroup that contains exactly  $r-1$  elements that generate that subgroup. Together with the neutral-element, this gives us a group of order  $r$ .

There are only two elements from  $G$  that generate subgroups  $H$  of size 2. One of these two elements is 1, which generates itself ( $1^k=1$ ) and the neutral element, while the other one is “-1”, i.e.,  $p-1$ , which generates  $p-1$  (remark:  $(p-1)^2 = 1 \bmod p$ ) and the neutral element.

Hence, if  $p$  is a safe prime, excluding the elements 1 and  $p-1$ , the other elements generate subgroups of size  $q$ .

# Subgroups when $p$ is a safe prime: e.g., $p = 23 = 2 \cdot 11 + 1$

```
1: # 1: 1, 1
2: #11: 1, 2, 4, 8, 16, 9, 18, 13, 3, 6, 12, 1
3: #11: 1, 3, 9, 4, 12, 13, 16, 2, 6, 18, 8, 1
4: #11: 1, 4, 16, 18, 3, 12, 2, 8, 9, 13, 6, 1
5: #22: 1, 5, 2, 10, 4, 20, 8, 17, 16, 11, 9, 22, 18, 21, 13, 19, 3, 15, 6, 7, 12, 14, 1
6: #11: 1, 6, 13, 9, 8, 2, 12, 3, 18, 16, 4, 1
7: #22: 1, 7, 3, 21, 9, 17, 4, 5, 12, 15, 13, 22, 16, 20, 2, 14, 6, 19, 18, 11, 8, 10, 1
8: #11: 1, 8, 18, 6, 2, 16, 13, 12, 4, 9, 3, 1
9: #11: 1, 9, 12, 16, 6, 8, 3, 4, 13, 2, 18, 1
10: #22: 1, 10, 8, 11, 18, 19, 6, 14, 2, 20, 16, 22, 13, 15, 12, 5, 4, 17, 9, 21, 3, 7, 1
11: #22: 1, 11, 6, 20, 13, 5, 9, 7, 8, 19, 2, 22, 12, 17, 3, 10, 18, 14, 16, 15, 4, 21, 1
12: #11: 1, 12, 6, 3, 13, 18, 9, 16, 8, 4, 2, 1
13: #11: 1, 13, 8, 12, 18, 4, 6, 9, 2, 3, 16, 1
14: #22: 1, 14, 12, 7, 6, 15, 3, 19, 13, 21, 18, 22, 9, 11, 16, 17, 8, 20, 4, 10, 2, 5, 1
15: #22: 1, 15, 18, 17, 2, 7, 13, 11, 4, 14, 3, 22, 8, 5, 6, 21, 16, 10, 12, 19, 9, 20, 1
16: #11: 1, 16, 3, 2, 9, 6, 4, 18, 12, 8, 13, 1
17: #22: 1, 17, 13, 14, 8, 21, 12, 20, 18, 7, 4, 22, 6, 10, 9, 15, 2, 11, 3, 5, 16, 19, 1
18: #11: 1, 18, 2, 13, 4, 3, 8, 6, 16, 12, 9, 1
19: #22: 1, 19, 16, 5, 3, 11, 2, 15, 9, 10, 6, 22, 4, 7, 18, 20, 12, 21, 8, 14, 13, 17, 1
20: #22: 1, 20, 9, 19, 12, 10, 16, 21, 6, 5, 8, 22, 3, 14, 4, 11, 13, 7, 2, 17, 18, 15, 1
21: #22: 1, 21, 4, 15, 16, 14, 18, 10, 3, 17, 12, 22, 2, 19, 8, 7, 9, 5, 13, 20, 6, 11, 1
22: # 2: 1, 22, 1
```

# Choosing parameters in DH

- $p$  must be a safe prime
- we enforce that  $a$  and  $b$  must larger than 1 ( $g^1$  is trivial) and smaller than  $p-1$  ( $g^{p-1} \bmod p = 1$  mod  $p$ , which again is a trivial case).
- $p$  must be large in order to have  $q$  large.
- $p$  must be large to prevent attacks such as [Pohlig-Hellman](#) (based on CRT), [Pollard's Rho](#), [Index Calculus](#) (probabilistic), and [others](#).

# DH provides Perfect Forward Secrecy

If the secrets (a, b, k) of one communication session are compromised, then other (past or future) communication sessions will not be compromised.

The value of forward secrecy is that it protects past communication. This reduces the motivation for attackers to compromise keys. For instance, if an attacker learns a long-term key, but the compromise is detected and the long-term key is revoked and updated, relatively little information is leaked in a forward secure system.

# Is DH secure against a passive attacker?

## Passive attacker:

- he knows  $g, p, g^a \bmod p, g^b \bmod p$
- he can compute easily  $g^{a+b} \bmod p$  (not useful) but not  $k = g^{ab} \bmod p$

Why?

To build the shared secret key it needs to recover **a** (or **b**). One way of breaking **DH**, it to solve **DLP**, which is believed to be hard to break. However, there could other ways of breaking DH that do not necessarily need to solve DLP. However, after several years is still believed to be safe.

# Is DH secure against an active attacker?

Man-in-the-Middle attack:

1. Adversary fakes Bob identify and performs DH with Alice
2. Adversary fakes Alice identify and performs DH with Bob

Alice and Bob think that they have a common shared key, instead they are sharing a secret key with the adversary.

**Problem.** Alice and Bob do not authenticate the received public keys.

# Elgamal Encryption Scheme

An extension of the DHKE protocol that allows also encryption. Only a few details have been changed, optimizing the order and number of messages exchanged between Alice and Bob.

Alice

$i \in [2, p - 2]$   
 $k_e \equiv g^i \pmod{p}$   
this is the ephemeral key

$k_m \equiv \beta^i \pmod{p} = g^{di} \pmod{p}$   
this is the masking key

Bob

choose prime  $p$ , generator  $g$

$$k_{pr} = d \in [2, p - 2]$$

$$k_{pub} = \beta = g^d \pmod{p}$$

$(p, g, \beta)$





# Elgamal Encryption Scheme (2)

Alice

given a message  $x$

$$y = x \cdot k_m \bmod p$$

$(y, k_e)$



Bob

$$k_m \equiv (k_e)^d \bmod p = g^{id} \bmod p$$

$$x \equiv y \cdot (k_m)^{-1} \bmod p$$

# Elgamal Encryption Scheme (3)

The scheme works since:

$$\begin{aligned}x &\equiv y \cdot (k_m)^{-1} \bmod p = (x \cdot k_m) \cdot ((k_e)^d)^{-1} \\&\equiv (x \cdot k_m) \cdot (k_e)^{-d} \\&\equiv (x \cdot \beta^i) \cdot (g^i)^{-d} \\&\equiv x \cdot (g^d)^i \cdot (g^i)^{-d} \\&\equiv x \cdot g^{id} \cdot g^{-id} \equiv x \bmod p\end{aligned}$$

# Elgamal Encryption Scheme (4)

- $g$  and  $p$  are chosen by Bob
- public key (Bob) can be fixed across session (advantage over DH + symmetric scheme)
- ephemeral key must change for each message and acts as a randomizer: if two messages  $x_1$  and  $x_2$  are encrypted using the same ephemeral key and the adversary knows  $(x_1, y_1)$  then

$$k_m \equiv y_1 \cdot (x_1)^{-1} \pmod{p}$$

$$x_2 \equiv y_2 \cdot (k_m)^{-1} \pmod{p}$$

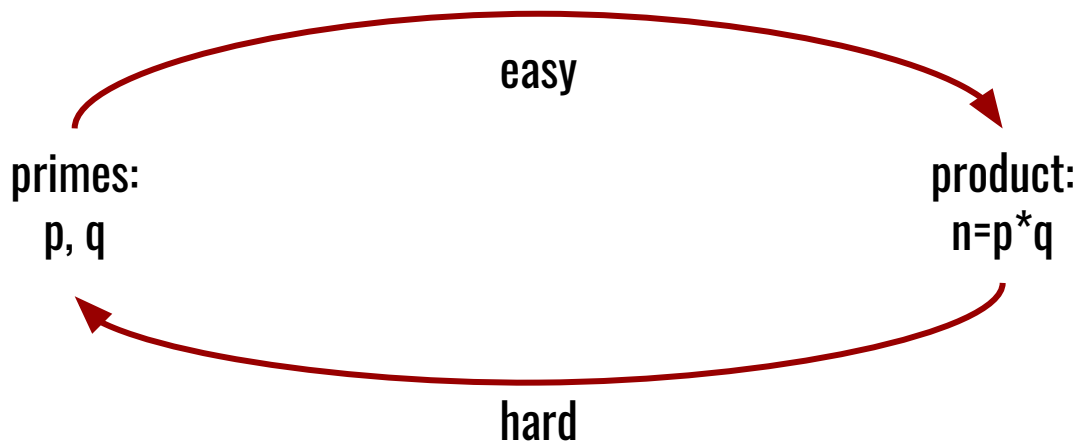
- still affected by MITM as DH

# Elgamal Encryption Scheme (5)

- Elgamal encryption is unconditionally **malleable**:
  - the adversary can replace  $(y, k_e)$  with  $(s \cdot y, k_e)$
  - decryption:
$$\begin{aligned} &\equiv (s \cdot y) \cdot (k_m)^{-1} \bmod p \\ &\equiv s \cdot (x \cdot k_m) \cdot (k_m)^{-1} \bmod p \\ &\equiv s \cdot x \bmod p \end{aligned}$$
  - Adversary cannot decrypt the plaintext  $x$  but can manipulate the ciphertext  $y$  in specific ways. As we will see RSA, a mitigation is to use padding on the plaintext. Textbook implementations of Elgamal are unsafe.

# RSA

PK scheme proposed by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977



This PK scheme is based on the integer factorization problem and devises a trapdoor.

# RSA scheme

What we would like to have in principle:

- public key:  $e$
- private key:  $d$  (the trapdoor)
- scheme:  $y = x^e$  such that  $x = y^d = x^{ed} = x^1$
- work with modular arithmetic (to work easily with a finite set of numbers)

Q. How to achieve this?

# RSA scheme

We can exploit the Euler's Phi Function and the Euler's Theorem:

$$a^{\Phi(n)} \equiv 1 \pmod{n}$$

We can now multiply by a:

$$a \cdot a^{\Phi(n)} \equiv a \cdot 1 \pmod{n}$$

$$a^{\Phi(n)+1} \equiv a \pmod{n}$$

which is very similar to what we wanted:

$$a^{ed} = a^1 \Rightarrow ed = \Phi(n) + 1$$

## RSA scheme (2)

We can generalize the previous idea. Since we are working with modular arithmetic then:

$$a^{\Phi(n)} \equiv 1 \pmod{n}$$

can be seen also as:

$$a^{k \cdot \Phi(n)} \equiv 1^k \equiv 1 \pmod{n}$$

if we multiply by  $a$ :

$$a^{k \cdot \Phi(n) + 1} \equiv a \pmod{n}$$

and we get:

$$a^{ed} = a^1 \Rightarrow ed = k \cdot \Phi(n) + 1$$



## RSA scheme (3)

Hence we need:

$$ed = k \cdot \Phi(n) + 1$$

we know that if:

$$n = p \cdot q \Rightarrow \Phi(n) = (p - 1)(q - 1)$$

thus:

$$ed = k \cdot (p - 1)(q - 1) + 1$$

$$ed \equiv 1 \pmod{\Phi(n)}$$

Hence, to make this work, given a number **e**, we need to find a number **d** that is the multiplicative inverse of **e**, i.e., **e**<sup>-1</sup>. We can compute this number using the EEA.

**Remark.** The multiplicative inverse of **e** exists if  $\gcd(e, \Phi(n)) = 1$

# RSA: encryption

We see a message  $x$  as an element of  $\mathbb{Z}_n = \{0, 1, \dots, n - 1\}$   
where:  $n = p \cdot q$

Given the **public key**  $(n, e)$  and the plaintext  $x$ :

$$y \equiv e_{k_{pub}}(x) = x^e \bmod n$$

where  $x, y \in \mathbb{Z}_n$

This implies that:

$$x < n$$

$$y < n$$

# RSA: decryption

Given the **private key  $d$**  and the ciphertext  $y$ :

$$x \equiv d_{k_{pr}}(y) = y^d \bmod n$$

# RSA: key generation

1. Choose two large primes  $p$  and  $q$
2. Compute:  $n = p \cdot q$
3. Compute:  $\Phi(n) = (p - 1)(q - 1)$
4. Select public exponent:  $e \in \{1, 2, \dots, \Phi(n) - 1\}$  such that  $\gcd(e, \Phi(n)) = 1$
5. Compute the private key:  $d \cdot e \equiv 1 \pmod{\Phi(n)}$

## RSA: key generation (2)

In practice, we choose  $e \in \{1, 2, \dots, \Phi(n) - 1\}$  such that  $\gcd(e, \Phi(n)) = 1$

Then we exploit EEA to find  $d$ :

$$\gcd(e, \Phi(n)) = s \cdot \Phi(n) + t \cdot e$$

we know that  $t$  is the multiplicative inverse of  $e$ :

$$d \equiv t \pmod{\Phi(n)}$$

# RSA: example

Alice

$$x = 4$$

$$y = x^e = 4^3 \equiv 31 \pmod{33}$$

Bob

1.  $p=3, q=11$


2.  $n = 3 \cdot 11 = 33$

3.  $\Phi(n) = 2 \cdot 10 = 20$

4.  $e = 3$

5.  $d \equiv e^{-1} \equiv 7 \pmod{20}$   
 $d \cdot e \equiv 1 \pmod{20}$

$$x = y^d = 31^7 \equiv 4 \pmod{33}$$

$$k_{pub} = (n, e)$$



$$y$$

## RSA: what if $x$ is larger than $N$ ?

If the message  $x$  is bigger than  $N$ , i.e.,  $x > N$ , after decryption we don't get the original message  $x$  but:

$$x' \equiv d_{k_{pr}}(y) = y^d \bmod N$$

which by definition is smaller than  $N$ ! Hence  $x' \neq x$ .

# RSA: properties

- We need to perform exponentiation in both encryption and decryption: there is a design choice since we can make encryption faster (choosing a small value for  $e$ ) but then we get a large  $d$  (hence decryption is expensive), or we can choose to make  $d$  small but  $e$  large (in this case we have to perform key generation a little bit different), getting slow encryption but fast decryption. However, having a small  $d$  is completely unsafe.
- Exponentiation for very large exponent is not very efficient: even when using square-and-multiply, RSA is significantly slower than AES.



# RSA: correctness (proof based on Fermat's Little Theorem)

$$x \equiv x^{ed} \pmod{p \cdot q}$$

Based on the Chinese Remainder Theorem this is true when:

$$x \equiv x^{ed} \pmod{p}$$

$$x \equiv x^{ed} \pmod{q}$$

Since:

$$ed \equiv 1 \pmod{(p-1)(q-1)}$$

$$ed \equiv 1 + k \cdot (p-1)(q-1)$$

Therefore:

$$x^{ed} \equiv x \cdot x^{k(p-1)(q-1)} \pmod{p}$$

$$\equiv x \cdot (x^{p-1})^{k(q-1)} \pmod{p}$$

$$\equiv x \cdot 1^{k(q-1)} \pmod{p}$$

(Fermat's Little Theorem)

$$\equiv x \pmod{p}$$

Symmetric proof for showing that  $x \equiv x^{ed} \pmod{q}$

# RSA: Is it secure?

The main assumption behind RSA is that it should be hard to compute  $d$ , or  $\Phi(n)$ , or  $p$ , or  $q$ .

To make this hard:

1.  $p$  and  $q$  should two large (more than 1024 bits) distinct prime numbers
2.  $e$  should be equal or larger than 3
3.  $(p-1)$  and  $(q-1)$  should have large prime factors otherwise Pollard's rho algorithm is able to perform factorization efficiently

Still, this “textbook” implementation of RSA suffers from different problems.

**RSA: what if two moduli share a common factor  $p$ ?**

$$N_1 = p \cdot q_1$$

$$N_2 = p \cdot q_2$$

$$\gcd(N_1, N_2) = p$$

Then we can easily recover  $q_1$  and  $q_2$ :

$$q_1 = N_1 / p$$

$$q_2 = N_2 / p$$

# RSA: avoid specific “corner case” messages

Messages such as  $x=0$ ,  $x=1$ ,  $x=(n-1)$  do not play well with exponentiation:

- $0^e = 0$
- $1^e = 1$
- if  $e$  is odd then:  $(n-1)^2 = 1 \pmod n$ , hence  $(n-1)^e = (n-1) \pmod n$

# RSA: small message, small exponent e

If x and e are small then:  $y = x^e < n$

Adversary can easily get the plaintext by computing  $\sqrt[e]{y}$

# RSA: same message encrypted with same exponent $e$

Given  $e$  ciphertexts of the same message  $x$  and the same exponent  $e$  is used during encryption but  $(p,q)$  different then the message  $x$  can be decrypted using the Chinese Remainder Theorem (CRT).

E.g.,  $e = 3$

$$y_1 = x^3 \bmod n_1$$

$$y_2 = x^3 \bmod n_2$$

$$y_3 = x^3 \bmod n_3$$

CRT can build ([here a step-by-step example](#)):

$$y = x^3 \bmod (n_1 \cdot n_2 \cdot n_3)$$

And then:

$$x = \sqrt[3]{y}$$

since we know by construction that

$$\begin{aligned} x &< \{n_1, n_2, n_3\} \\ x^3 &< n_1 \cdot n_2 \cdot n_3 \end{aligned}$$

# RSA: same message encrypted with different coprime exponents

$$\begin{aligned}y_1 &= x^{e_1} \bmod n \\ y_2 &= x^{e_2} \bmod n\end{aligned}$$

If we have that:

$$\gcd(e_1, e_2) = 1$$

Then using EEA we can compute:

$$s \cdot e_1 + t \cdot e_2 = 1$$

But then:

$$y_1^s \cdot y_2^t = (x^{e_1})^s \cdot (x^{e_2})^t = x^{s \cdot e_1 + t \cdot e_2} = x$$

[Here](#) a step-by-step example.

# RSA: small exponent $e$ and related messages

When the exponent  $e$  is small, for instance,  $e = 3$  and we have two “related” messages such as:

$$x_2 = x_1 + 1$$

Then:

$$x_1 = \frac{y_2 + 2y_1 - 1}{y_2 - y_1 + 2}$$

This can be generalized for any affine relation:  $x_2 = \alpha \cdot x_1 + \beta$

See paper “[Low Exponent RSA with related messages](#)” (Eurocrypt 1996) for the proof.



# RSA: small message space

Given a ciphertext  $y$ , if the message space is small (e.g.,  $k$  values), then adversary can encrypt all  $k$  values and then compare results with the ciphertext  $y$ . **Textbook RSA has no randomization.**

# RSA: same $n$ for different users

Given  $\{e, d, n\}$ , one user can compute  $(p, q)$  [see Handbook of Applied Cryptography, 8.2.2 (i)] and then compute private keys of other users (he just need to compute the multiplicative inverse of their public key  $e$ !).

Recovery of  $p$  and  $q$  given  $(e, d, n)$  is also explained by Appendix C in [NIST SP 850-56B](#).

# RSA: chosen ciphertext attack and malleability

Adversary wants to decrypt:

$$y = x^e \bmod n$$

He builds:

$$y' = 2^e \cdot y$$

and asks for decryption of  $y'$ :

$$x' = (2^e \cdot y)^d \bmod n$$

$$x' = (2^{de} \cdot (x^e)^d) \bmod n$$

$$x' = (2 \cdot x) \bmod n$$

and then  $x$  can be dividing it by 2, e.g., see use of the multiplicative inverse in modular arithmetic. The adversary can derive  $x$  by asking decryption of  $x'$ . Or, when he cannot ask for decryption, he still can manipulate the ciphertext in a well-defined way (e.g., he knows how to multiply by 2 the msg  $x$ ).

# RSA: How to mitigate/solve most of these attacks?

Textbook RSA is unsafe. Textbook Elgamal is unsafe. In practice, we should always use a standard that provides rules on how to use a PK scheme for real-world usages:

Solution: **PKCS (Public Key Cryptography Standards)**

**PKCS#1:** RSA Cryptography Standard

**PKCS#3:** Diffie Hellman Key Agreement Standard

**PKCS#11:** Elliptic Curve Cryptography Standard

# PKCS#1: RSA Cryptography Standard

It defines two encryption/decryption schemes based on RSA:

1. **RSAES-PKCS1-v1\_5** (older)
2. **RSAES-OAEP** (newer): RSA improved with Optimal Asymmetric Encryption Padding (OEAP)

# RSAES-PKCS1-V1\_5

Message is padded as:

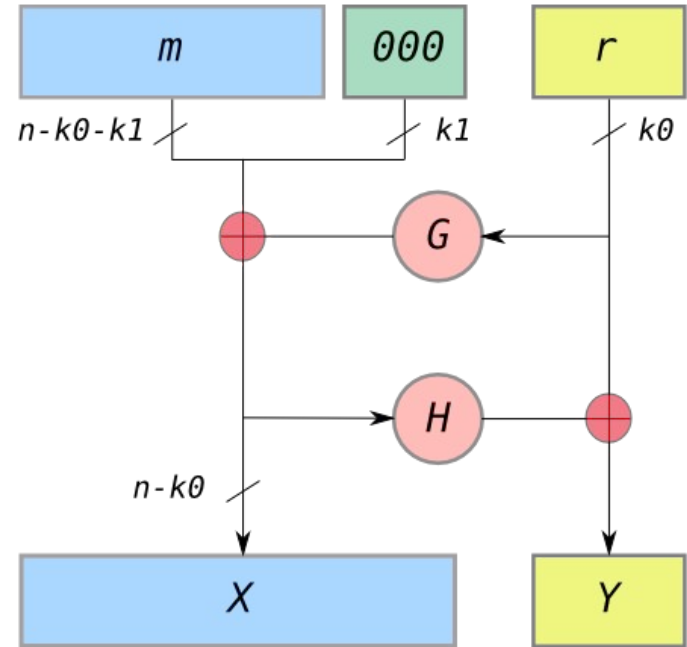
$$x' = 0x00 \parallel 0x02 \parallel \text{at least 8 non-zero bytes} \parallel 0x00 \parallel x$$

After decryption of a ciphertext, check expected patterns in the plaintext (e.g., look for 0x02). Non zero bytes play as the role of a **randomizer**.

Some padding oracle attacks have been proposed hence OAEP should be used instead.

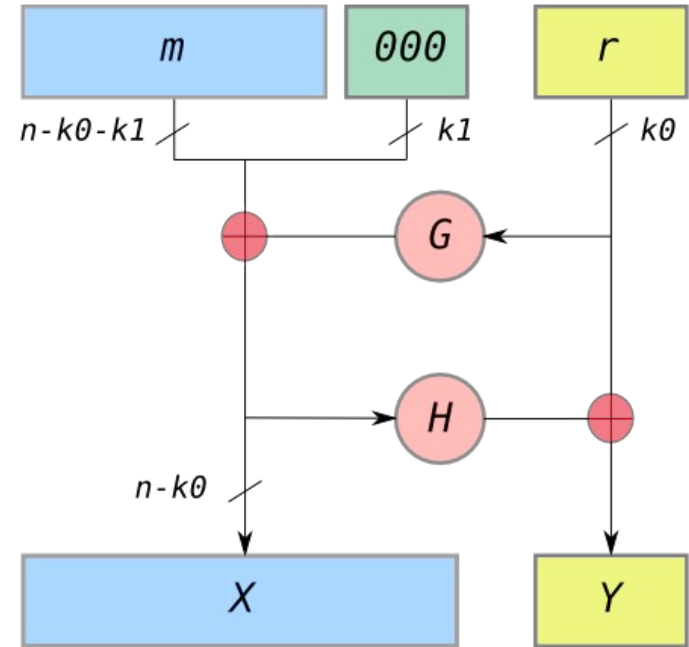
# Optimal Asymmetric Encryption Padding (OAEP)

- $n$  is the number of bits in the RSA modulus
- $k_0$  and  $k_1$  are integers fixed by the protocol
- $m$  is the plaintext message, an  $(n - k_0 - k_1)$ -bit string
- $G$  and  $H$  are random oracles such as cryptographic hash functions.
- $\oplus$  is a xor operation.



# OEAP: encoding

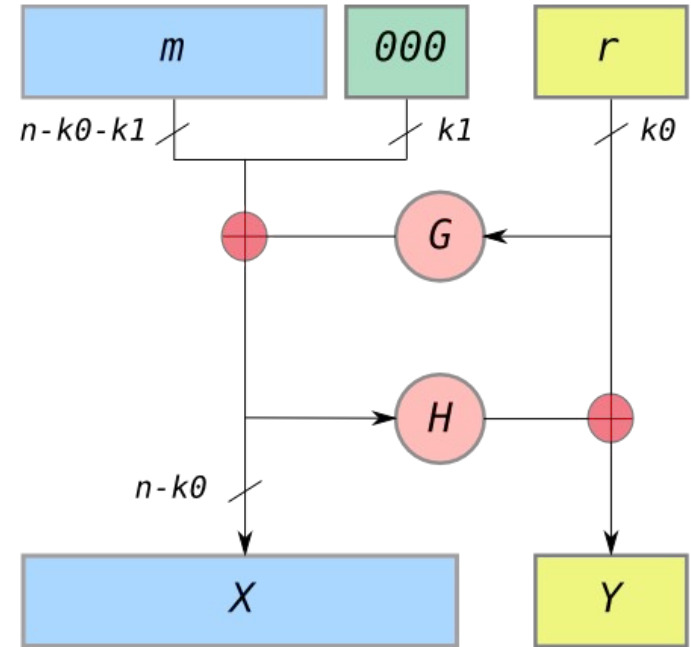
- messages are padded with  $k_1$  zeros to be  $n - k_0$  bits in length
- $r$  is a randomly generated  $k_0$ -bit string
- $G$  expands the  $k_0$  bits of  $r$  to  $n - k_0$  bits.
- $X = m00\dots0 \oplus G(r)$
- $H$  reduces the  $n - k_0$  bits of  $X$  to  $k_0$  bits.
- $Y = r \oplus H(X)$
- The output is  $X || Y$  where  $X$  is shown in the diagram as the leftmost block and  $Y$  as the rightmost block.





# OEAP: decoding

- recover the random string as  $r = Y \oplus H(X)$
- recover the message as  $m00\dots0 = X \oplus G(r)$



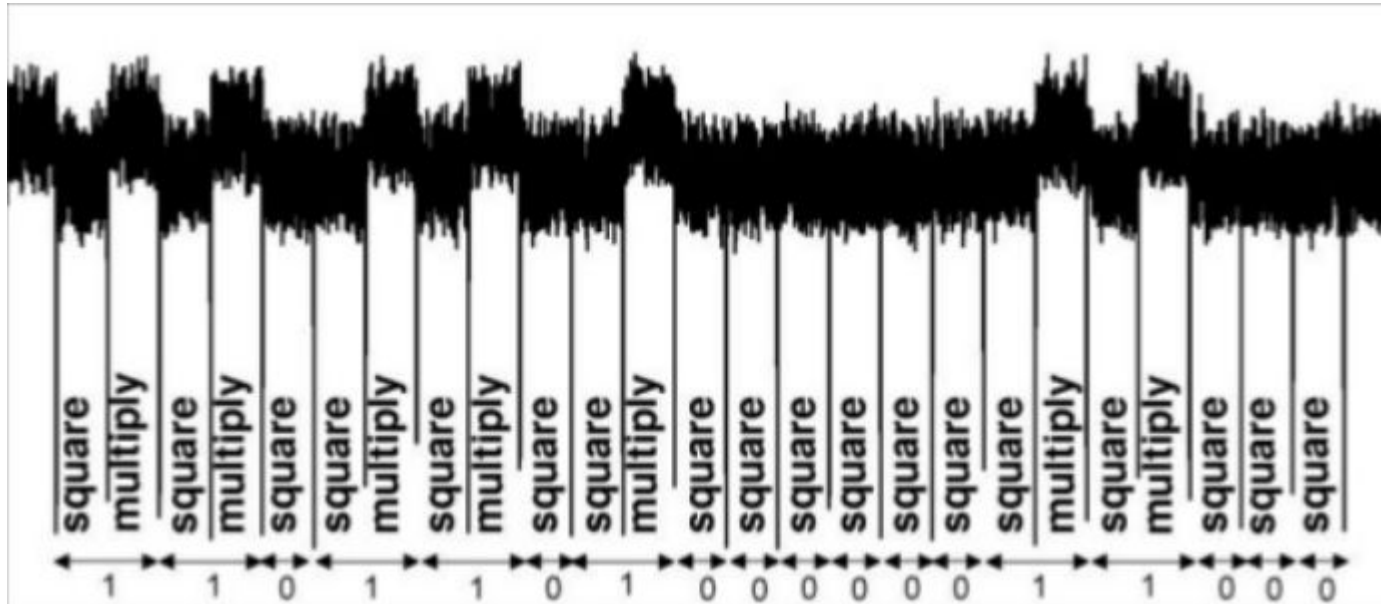
# OEAP: properties

- Based on a Feistel Network.
- The deterministic property of RSA is now avoided by using the OAEP encoding thanks to the use of a random string  $r$ .
- It provides **"all-or-nothing" security** since to recover  $m$ , one must recover the entire  $X$  and the entire  $Y$ ;  $X$  is required to recover  $r$  from  $Y$ , and  $r$  is required to recover  $m$  from  $X$ . Since any changed bit of a cryptographic hash completely changes the result, the entire  $X$ , and the entire  $Y$  must both be completely recovered.

**RSA with proper padding: are we safe now?**

# Side-channel attacks on RSA: power analysis

Power analysis can recover the key when performing exponentiation with the private key:



# Side-channel attacks on RSA (2)

How to prevent effective power analysis or other timing attacks during exponentiation?

Modify square-and-multiply in order to take constant time (power) at each iteration, regardless of the exponent bit value: e.g., add a dummy multiplication in s-a-m. Still, real-world implementations (e.g., based on Montgomery Reduction) are weak to more advanced power analyses.

Another approach is message randomization (blinding) where, before decryption, the system picks a random  $r$  and computes  $r^e$ , then decrypts  $yr^e$  obtaining  $xr$ , and from then divide it by  $r$  to get back  $x$ .

Another strategy is to perform exponent/key randomization (exponent blinding):

$$d' = d + k \cdot \Phi(n)$$

using a random  $k$  for each decryption. Require additional work to be secure!

# Side-channel attacks on RSA (3)

How to prevent effective power analysis or other timing attacks during exponentiation?

Always use up-to-date implementations of RSA (e.g., openssl, libssl) that are specifically designed to prevent these attacks. They are frequently updated to prevent the most cutting-edge attacks.

# RSA: many attacks, what can we do?

**NEVER USE A TEXTBOOK IMPLEMENTATION OF RSA**  
**ALWAYS USE A WELL-KNOWN RSA LIBRARY**  
**DO NOT COPY RSA PARAMETERS FROM INTERNET**

[RsaCtfTool](#) can test many well-known attacks for RSA. [Here](#) you can find examples (.pem files, i.e., a format for storing public/private keys) where you can test these attacks. [Yafu](#) or [msieve](#) can be used for performing factorization of “small” numbers. Library such as SageMath have integrated several algorithms for factorization.

# RSA: so hard to implement/use in the right way

People is starting to suggest to NOT use RSA: after so many years, we still see so many new attacks or wrong usages of RSA in real-world applications. RSA is not fundamentally broken but it is hard to use in practice:

1. *Seriously, stop using RSA* [\[blog post\]](#) and *F\*\*\* RSA* [\[video\]](#) by TrailOfBits
2. Generating a key pair in RSA is relatively expensive: we reuse the same pair more than once but then this means that RSA for key exchange does not provide forward perfect secrecy. Transport Layer Security (TLS) 1.3 dropped support for RSA key exchange. We discuss TLS later in the course.

Still, RSA is the most well-known asymmetric scheme and it is commonly used for instance in signature schemes (e.g., RSA-PSS, we discuss it later in the course).



# Credits

These slides are based on material from:

- Slides of Prof. D'Amore from CNS 2019-2020
- Christof Paar and Jan Pelzl. Understanding Cryptography: A Textbook for Students and Practitioners. Springer. <http://www.crypto-textbook.com/>
- Wikipedia (english version)