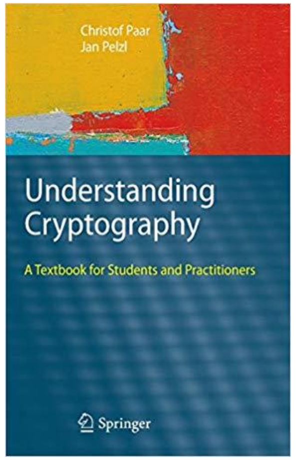
COM/BLM 426
Information Security

Asst. Prof. Dr. Bulent TUGRUL
btugrul@eng.ankara.edu.tr

Slides are mainly based on:

Understanding Cryptography: A Textbook for Students and Practitioners



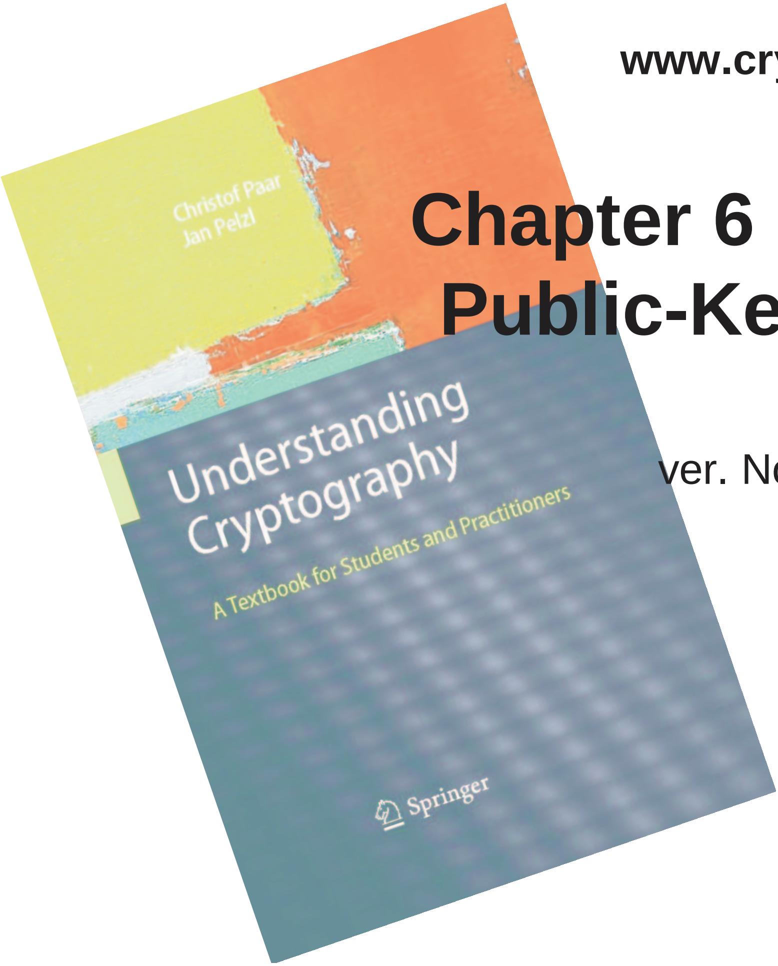
by Christof Paar and Jan Pelzl

Springer, 1st Edition, 2010

Understanding Cryptography – A Textbook for Students and Practitioners

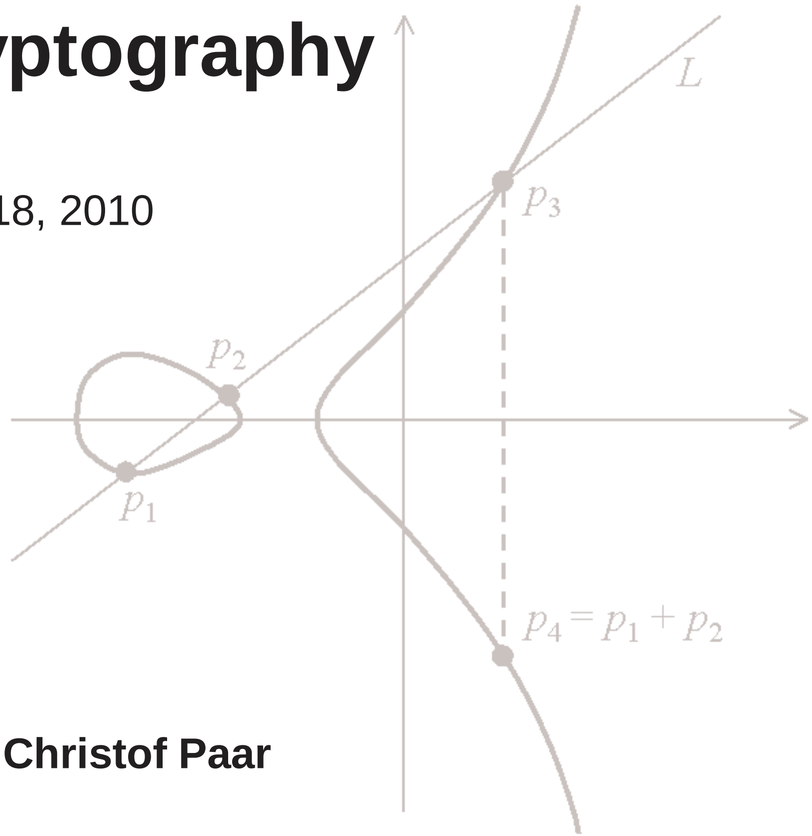
by Christof Paar and Jan Pelzl

www.crypto-textbook.com



Chapter 6 – Introduction to Public-Key Cryptography

ver. November 18, 2010



These slides were prepared by Timo Kasper and Christof Paar

Some legal stuff (sorry): Terms of Use

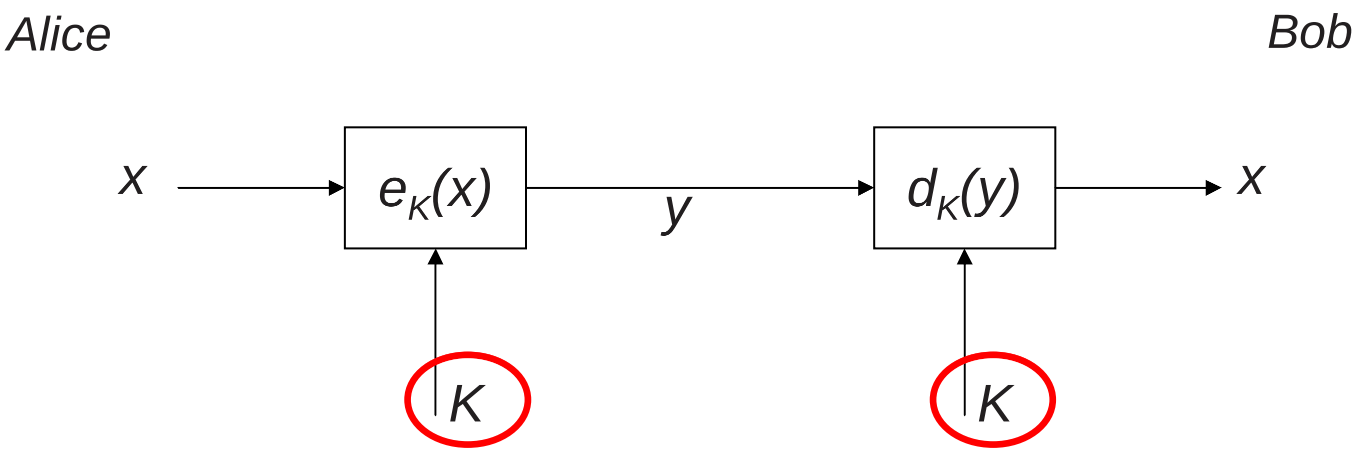
Content of this Chapter

- Symmetric Cryptography Revisited
- Principles of Asymmetric Cryptography
- Practical Aspects of Public-Key Cryptography
- Important Public-Key Algorithms
- Essential Number Theory for Public-Key Algorithms

Content of this Chapter

- **Symmetric Cryptography Revisited**
- Principles of Asymmetric Cryptography
- Practical Aspects of Public-Key Cryptography
- Important Public-Key Algorithms
- Essential Number Theory for Public-Key Algorithms

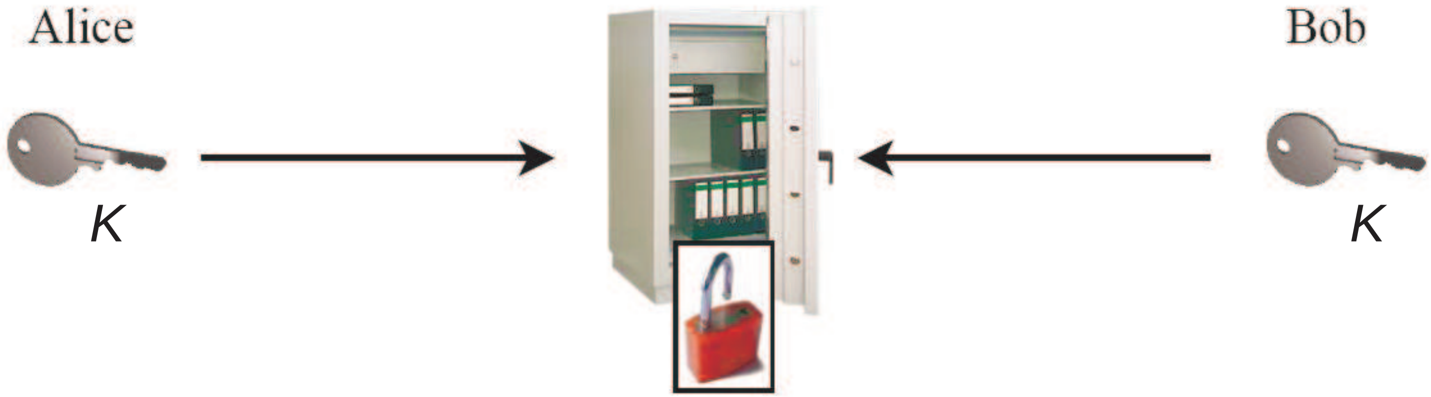
Symmetric Cryptography revisited



Two properties of symmetric (secret-key) crypto-systems:

- The **same secret key *K*** is used for encryption and decryption
- Encryption and Decryption are very similar (or even identical) functions

Symmetric Cryptography: Analogy



Safe with a strong lock, only Alice and Bob have a copy of the key

- Alice encrypts → locks message in the safe with her key
- Bob decrypts → uses his copy of the key to open the safe

Symmetric Cryptography: Shortcomings

- Symmetric algorithms, e.g., AES or 3DES, are very secure, fast & widespread **but**:

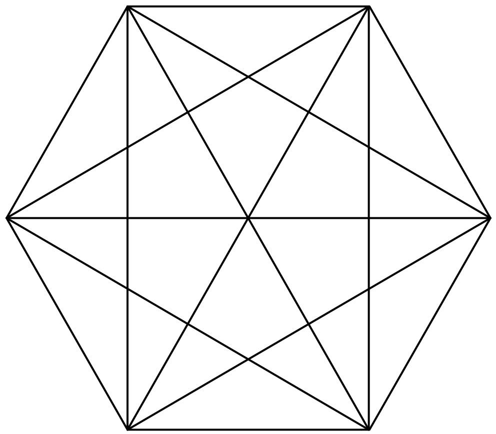
- Key distribution problem: The secret key must be **transported securely**
- Number of keys: In a network, each pair of users requires an individual key

→ *n* users in the network require $\frac{n \cdot (n-1)}{2}$ keys, each user stores *(n-1)* keys

Example:

6 users (nodes)

$\frac{6 \cdot 5}{2} = 15$ keys (edges)



- Alice or Bob can **cheat each other**, because they have identical keys.
  Example: Alice can claim that she never ordered a TV on-line from Bob (he could have fabricated her order). To prevent this: „non-repudiation“

Content of this Chapter

- Symmetric Cryptography Revisited
- **Principles of Asymmetric Cryptography**
- Practical Aspects of Public-Key Cryptography
- Important Public-Key Algorithms
- Essential Number Theory for Public-Key Algorithms

Idea behind Asymmetric Cryptography

New Idea:

Use the „good old mailbox“ principle:

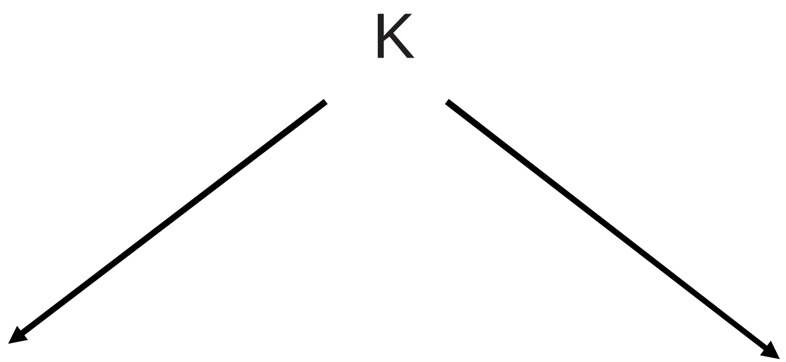
Everyone can drop a letter

But: Only the owner has the correct key to open the box

1976: first publication of such an algorithm by Whitfield Diffie and Martin Hellman,and also by Ralph Merkle.

Asymmetric (Public-Key) Cryptography

Principle: “Split up” the key



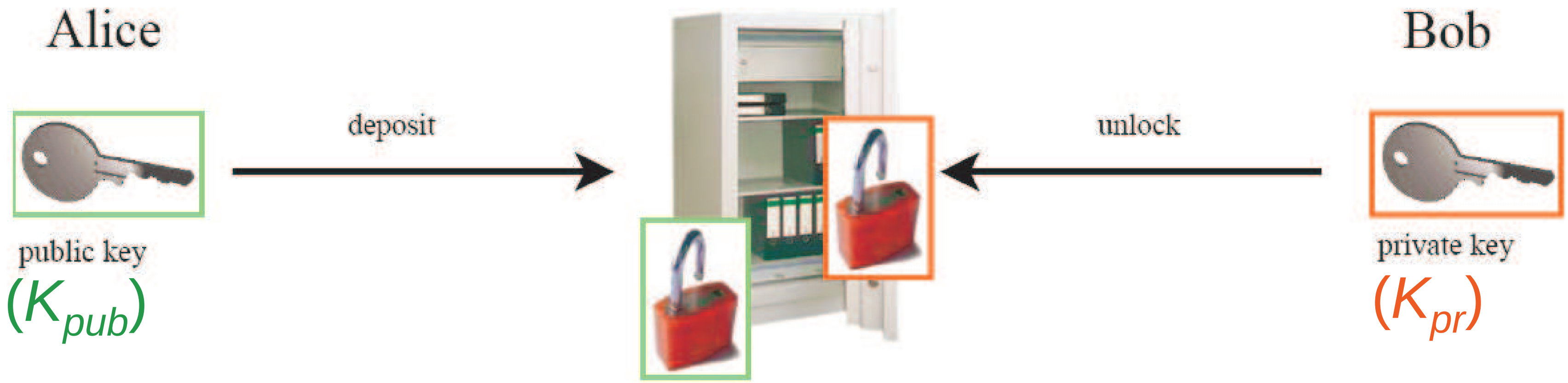
Public Key (K_{pub})
(Encrypt)

Secret Key (K_{pr})
(Decrypt)

→ During the key generation, a key pair K_{pub} and K_{pr} is computed

Asymmetric Cryptography: Analogy

Safe with public lock and private lock:



- Alice deposits (encrypts) a message with the - *not secret* - public key K_{pub}
- Only Bob has the - *secret* - private key K_{pr} to retrieve (decrypt) the message

Content of this Chapter

- Symmetric Cryptography Revisited
- Principles of Asymmetric Cryptography
- **Practical Aspects of Public-Key Cryptography**
- Important Public-Key Algorithms
- Essential Number Theory for Public-Key Algorithms

Basic Protocol for Public-Key Encryption

Alice | **Bob**

Bob: $(K_{pubB}, K_{prB}) = K$

Bob → Alice: K_{pubB}

Alice: x

Alice: $y = e_{K_{pubB}}(x)$

Alice → Bob: y

Bob: $x = d_{K_{prB}}(y)$

→ Key Distribution Problem solved *

*) at least for now; public keys need to be authenticated, cf.Chptr. 13 *of Understanding Cryptogr*.

Security Mechanisms of Public-Key Cryptography

Here are main mechanisms that can be realized with asymmetric cryptography:

- **Key Distribution** (e.g., Diffie-Hellman key exchange, RSA) without a pre-shared secret (key)
- **Nonrepudiation and Digital Signatures** (e.g., RSA, DSA or ECDSA) to provide message integrity
- **Identification**, using challenge-response protocols with digital signatures
- **Encryption** (e.g., RSA / Elgamal)
  Disadvantage: Computationally very intensive
  (1000 times slower than symmetric Algorithms!)

Basic Key Transport Protocol 1/2

In practice: **Hybrid systems**, incorporating asymmetric and symmetric algorithms

1. **Key exchange** (for symmetric schemes) and **digital signatures** are performed with (slow) **asymmetric** algorithms

2. **Encryption** of data is done using (fast) symmetric ciphers, e.g., **block ciphers or stream ciphers**

Basic Key Transport Protocol 2/2

Example: Hybrid protocol with AES as the symmetric cipher

Alice		Bob	
	$\xleftarrow{K_{pubB}}$	$(K_{pubB}, K_{prB}) = K$	**Key Exchange (asymmetric)**
Choose random symmetric key K			
$y_1 = e_{K_{pubB}}(K)$	$\xrightarrow{y_1}$		
		$K = d_{K_{prB}}(y_1)$	
message x			**Data Encryption (symmetric)**
$y_2 = AES_K(x)$	$\xrightarrow{y_2}$	$x = AES^{-1}_K(y_2)$	

Content of this Chapter

- Symmetric Cryptography Revisited
- Principles of Asymmetric Cryptography
- Practical Aspects of Public-Key Cryptography
- **Important Public-Key Algorithms**
- Essential Number Theory for Public-Key Algorithms

How to build Public-Key Algorithms

Asymmetric schemes are based on a **„one-way function“ *f()***:

- Computing $y = f(x)$ is computationally easy
- Computing $x = f^{-1}(y)$ is computationally infeasible

One way functions are based on **mathematically hard problems**.

Three main families:

- **Factoring integers** (RSA, ...):
  Given a composite integer n, find its prime factors
  (Multiply two primes: easy)

- **Discrete Logarithm** (Diffie-Hellman, Elgamal, DSA, …):
  Given a, y and m, find x such that $a^x = y \bmod m$
  (Exponentiation a^x : easy)

- **Elliptic Curves (EC)** (ECDH, ECDSA): Generalization of discrete logarithm

Note: The problems are considered mathematically hard, but no proof exists (so far).

Key Lengths and Security Levels

Symmetric	*ECC*	*RSA, DL*	*Remark*
64 Bit	128 Bit	≈ 700 Bit	Only short term security (a few hours or days)
80 Bit	160 Bit	≈ 1024 Bit	Medium security (except attacks from big governmental institutions etc.)
128 Bit	256 Bit	≈ 3072 Bit	Long term security (without quantum computers)

- The exact complexity of RSA (factoring) and DL (Index-Calculus) is difficult to estimate
- The existence of quantum computers would probably be the end for ECC, RSA & DL (at least 2-3 decades away, and some people doubt that QC will ever exist)

Content of this Chapter

- Symmetric Cryptography Revisited
- Principles of Asymmetric Cryptography
- Practical Aspects of Public-Key Cryptography
- Important Public-Key Algorithms
- **Essential Number Theory for Public-Key Algorithms**

Euclidean Algorithm 1/2

- Compute the **greatest common divisor *gcd* (r_0, r_1)** of two integers r_0 and r_1

- gcd is **easy for small numbers**:
  1. factor r_0 and r_1
  2. gcd = highest common factor

- Example:

  $r_0 = 84 = 2 \cdot 2 \cdot 3 \cdot 7$

  $r_1 = 30 = 2 \cdot 3 \cdot 5$

  → The gcd is the product of all common prime factors:

  $2 \cdot 3 = 6 = gcd\,(30,84)$

- **But:** Factoring is complicated (and often infeasible) for large numbers

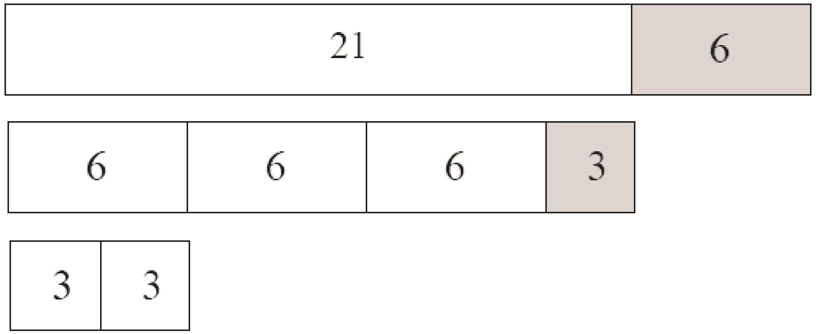
Euclidean Algorithm 2/2

- Observation: ***gcd*** $(r_0, r_1) = gcd(r_0 - r_1, r_1)$

→ Core idea:

  - Reduce the problem of finding the gcd of two given numbers to that of the **gcd of two smaller numbers**
  - Repeat process recursively
  - The final ***gcd*** $(r_i, 0) = r_i$ is the answer to the original problem !

Example: *gcd* (r_0, r_1) for $r_0 = 27$ and $r_1 = 21$



$$\gcd(27, 21) = \gcd(1 \cdot 21 + 6, 21) = \gcd(21, 6)$$

$$\gcd(21, 6) = \gcd(3 \cdot 6 + 3, 6) = \gcd(6, 3)$$

$$\gcd(6, 3) = \gcd(2 \cdot 3 + 0, 3) = \gcd(3, 0) = 3$$

- Note: very efficient method even for long numbers: The complexity grows **linearly** with the number of bits

For the full Euclidean Algorithm see Chapter 6 in *Understanding Cryptography.*

Extended Euclidean Algorithm 1/2

- Extend the Euclidean algorithm to **find modular inverse** of r_1 mod r_0
- EEA computes *s*,*t*, and the gcd : $\gcd(r_0, r_1) = s \cdot r_0 + t \cdot r_1$
- Take the relation **mod r_0**

$$
\begin{aligned}
s \cdot r_0 + t \cdot r_1 &= 1 \\
s \cdot 0 + t \cdot r_1 &\equiv 1 \bmod r_0 \\
r_1 \cdot t &\equiv 1 \bmod r_0
\end{aligned}
$$

→ Compare with the definition of modular inverse**: *t* is the inverse of r_1 mod r_0**

- Note that $gcd\ (r_0, r_1) = 1$ in order for the inverse to exist

- **Recursive formulae** to calculate *s* and *t* in each step

  → „magic table“ for *r*, *s*, *t* and a quotient *q* to derive the inverse with pen and paper

  (cf. Section 6.3.2 in *Understanding Cryptography*)

Extended Euclidean Algorithm 2/2

Example:

- Calculate the modular Inverse of 12 mod 67:
- From magic table follows $-5 \cdot 67 + 28 \cdot 12 = 1$
- Hence **28 is the inverse** of 12 mod 67.

i	q_{i-1}	r_i	s_i	t_i
2	5	7	1	-5
3	1	5	-1	6
4	1	2	2	-11
5	2	1	-5	**28**

- Check: $28 \cdot 12 = 336 \equiv 1 \bmod 67$ ✓

For the full Extended Euclidean Algorithm see Chapter 6 in *Understanding Cryptography.*

Euler‘s Phi Function 1/2

- *New problem, important for public-key systems, e.g., RSA:*
  Given the set of the *m* integers {0, 1, 2, …, *m* -1},
  How many numbers in the set are **relatively prime to *m*** ?

- Answer: **Euler‘s Phi function *Φ(m)***

- **Example** for the sets {0,1,2,3,4,5} (*m*=6), and {0,1,2,3,4} (*m*=5)

$$\gcd(0,6) = 6$$
$$\gcd(1,6) = 1 \leftarrow$$
$$\gcd(2,6) = 2$$
$$\gcd(3,6) = 3$$
$$\gcd(4,6) = 2$$
$$\gcd(5,6) = 1 \leftarrow$$

→ 1 and 5 relatively prime to *m*=6,
hence ***Φ*(6) = 2**

$$\gcd(0,5) = 5$$
$$\gcd(1,5) = 1 \leftarrow$$
$$\gcd(2,5) = 1 \leftarrow$$
$$\gcd(3,5) = 1 \leftarrow$$
$$\gcd(4,5) = 1 \leftarrow$$

→ ***Φ*(5) = 4**

- Testing one gcd per number in the set is **extremely slow for large *m*.**

Euler's Phi Function 2/2

- **If** canonical factorization of m known**:** $m = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_n^{e_n}$
  (where p_i primes and e_i positive integers)
- **then** calculate Phi according to the relation $\Phi(m) = \prod_{i=1}^{n}(p_i^{e_i} - p_i^{e_i - 1})$

- Phi especially easy for $e_i = 1$, e.g., $m = p \cdot q$ → $\Phi(m) = (p-1) \cdot (q-1)$

- **Example** $m = 899 = 29 \cdot 31$:
  $\boldsymbol{\Phi(899)} = (29-1) \cdot (31-1) = 28 \cdot 30$ **= 840**

- **Note:** Finding $\Phi(m)$ is computationally easy **if factorization of *m* is known**
  (otherwise the calculation of $\Phi(m)$ becomes computationally infeasible for large numbers)

Fermat‘s Little Theorem

- Given a **prime *p*** and an **integer *a***: $a^p \equiv a \pmod{p}$
- Can be rewritten as $a^{p-1} \equiv 1 \pmod{p}$

- **Use: Find modular inverse**, if p is prime. Rewrite to $a \cdot a^{p-2} \equiv 1 \pmod{p}$
- Comparing with definition of the modular inverse $a \cdot a^{-1} \equiv 1 \bmod m$

  → $a^{-1} \equiv a^{p-2} \pmod{p}$ is the modular inverse modulo a prime *p*

Example: *a* = 2, *p* = 7

$$a^{p-2} = 2^5 = 32 \equiv 4 \bmod 7$$

$$\text{verify: } 2 \cdot 4 \equiv 1 \bmod 7$$ ✓

- Fermat‘s Little Theorem works only **modulo a prime *p***

Euler‘s Theorem

- Generalization of Fermat‘s little theorem to **any integer modulus**
- Given two **relatively prime integers *a*** and ***m* :** $a^{\Phi(m)} \equiv 1 \ (\bmod \ m)$
- **Example**: *m*=12, *a*=5
  1. Calculate Euler‘s Phi Function

     $$\Phi(12) = \Phi(2^2 \cdot 3) = (2^2 - 2^1)(3^1 - 3^0) = (4-2)(3-1) = 4$$

  2. Verify Euler‘s Theorem

     $$5^{\Phi(12)} = 5^4 = 25^2 = 625 \equiv 1 \bmod \ 12$$

- Fermat‘s little theorem = special case of Euler‘s Theorem
- for a prime ***p***: $\Phi(p) = (p^1 - p^0) = p - 1$

  → Fermat: $a^{\Phi(p)} = a^{p-1} \equiv 1 \ (\bmod \ p)$

Lessons Learned

- Public-key algorithms have **capabilities that symmetric ciphers don't have**, in particular digital signature and key establishment functions.
- Public-key algorithms are **computationally intensive** (a nice way of saying that they are *slow*), and hence are poorly suited for bulk data encryption.
- Only **three families of public-key schemes** are widely used. This is considerably fewer than in the case of symmetric algorithms.
- The **extended Euclidean algorithm** allows us to compute **modular inverses** quickly, which is important for almost all public-key schemes.
- **Euler's phi function** gives us the number of elements smaller than an integer n that are relatively prime to n. This is important for the RSA crypto scheme.