
بسم الله الرحمن الرحيم

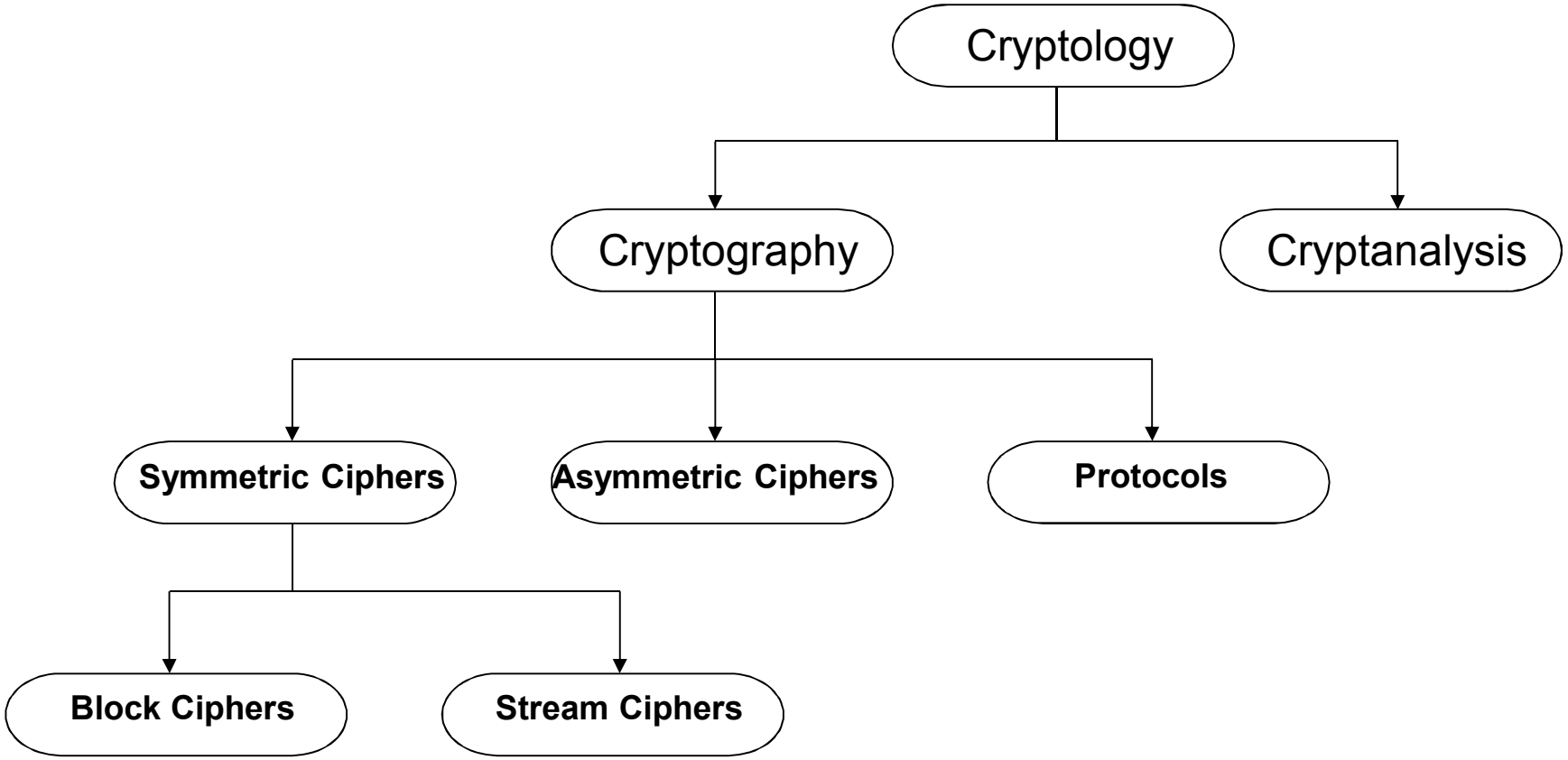
Network Security

Ali Shakiba
Vali-e-Asr University of Rafsanjan
ali.shakiba@vru.ac.ir
www.1ali.ir

Content of this Chapter

- Overview on the field of cryptology
- Basics of symmetric cryptography
- Cryptanalysis
- Substitution Cipher
- Modular arithmetic
- Shift (or Caesar) Cipher and Affine Cipher

Classification of the Field of Cryptology



Some Basic Facts

- **Ancient Crypto:** Early signs of encryption in Eqypt in ca. 2000 B.C.
  Letter-based encryption schemes (e.g., Caesar cipher) popular ever since.

- **Symmetric ciphers:** All encryption schemes from ancient times until 1976 were symmetric ones.

- **Asymmetric ciphers:** In 1976 public-key (or asymmetric) cryptography was openly proposed by Diffie, Hellman and Merkle.

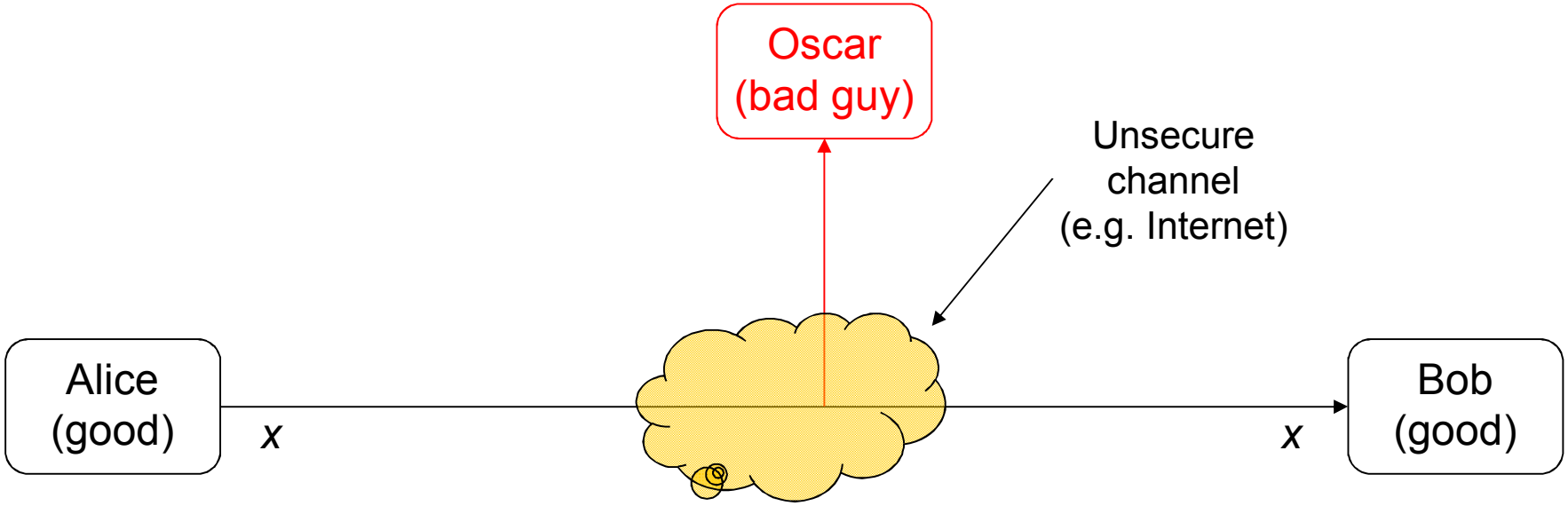
- **Hybrid Schemes:** The majority of today's protocols are hybrid schemes, i.e., the use both
  - symmteric ciphers (e.g., for encryption and message authentication) and
  - asymmetric ciphers (e.g., for key exchange and digital signature).

Content of this Chapter

- Overview on the field of cryptology
- **Basics of symmetric cryptography**
- Cryptanalysis
- Substitution Cipher
- Modular arithmetic
- Shift (or Caesar) Cipher and Affine Cipher

Symmetric Cryptography

- Alternative names: **private-key**, **single-key** or **secret-key** cryptography.

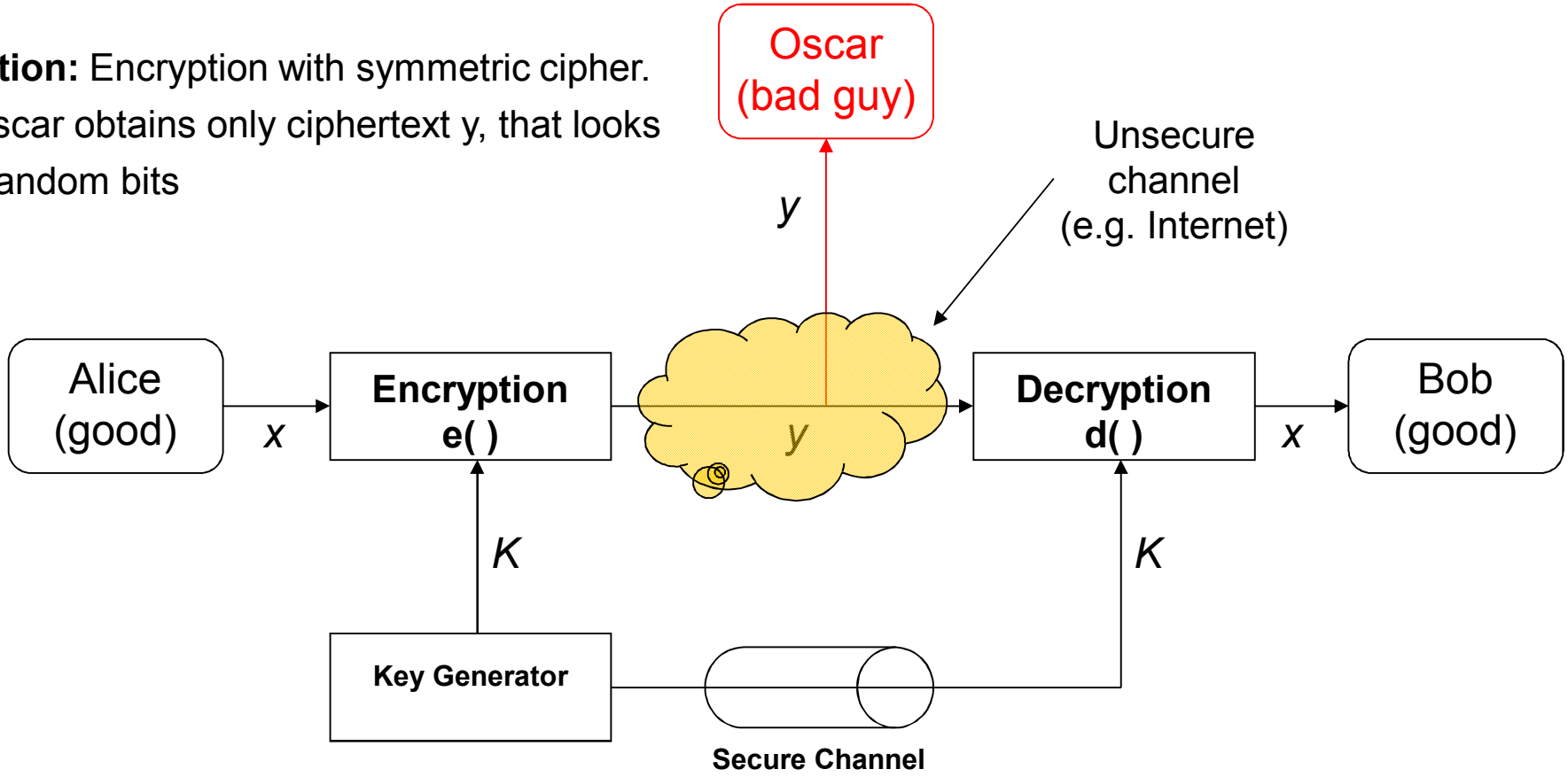


- **Problem Statement:**
  1) Alice and Bob would like to communicate via an unsecure channel (e.g., WLAN or Internet).
  2) A malicious third party Oscar (the bad guy) has channel access but should not be able to understand the communication.

Symmetric Cryptography

Solution: Encryption with symmetric cipher.

⇒ Oscar obtains only ciphertext y, that looks like random bits



- x is the. **plaintext**
- y is the **ciphertext**
- *K* is the **key**
- Set of all keys {*K*1, *K*2, ...,*Kn*} is the **key space**

Symmetric Cryptography

- Encryption equation $y = e_K(x)$
- Decryption equation $x = d_K(y)$

- Encryption and decryption are inverse operations if the same key K is used on both sides:

$$d_K(y) = d_K(e_K(x)) = x$$

- Important: The key must be transmitted via a **secure channel** between Alice and Bob.
- The secure channel can be realized, e.g., by manually installing the key for the Wi-Fi Protected Access (WPA) protocol or a human courier.
- However, the system is only secure if an attacker does not learn the key K!

$\Rightarrow$ **The problem of secure communication is reduced to secure transmission and storage of the key K.**

Content of this Chapter

- Overview on the field of cryptology
- Basics of symmetric cryptography
- **Cryptanalysis**
- Substitution Cipher
- Modular arithmetic
- Shift (or Caesar) Cipher and Affine Cipher

Why do we need Cryptanalysis?

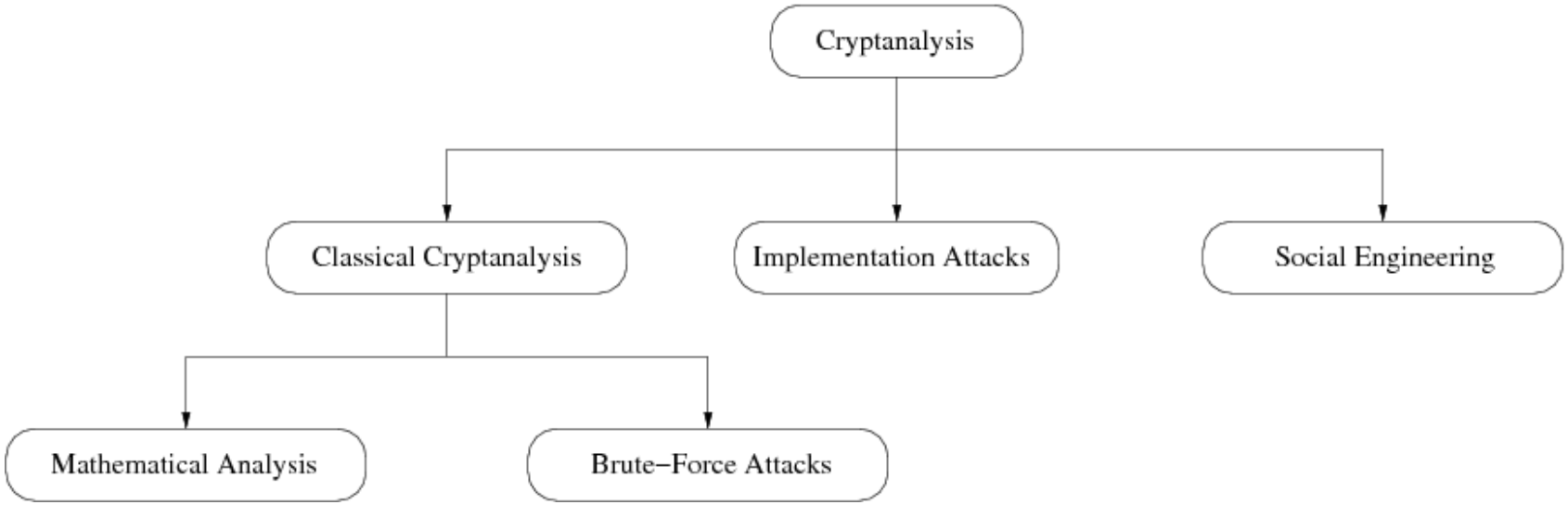
- There is no *mathematical proof of security* for any practial cipher
- The only way to have assurance that a cipher is secure is to try to break it (and fail) !

Kerckhoff's Principle is paramount in modern cryptography:

> A cryptosystem should be secure even if the attacker (Oscar) knows all details about the system, with the exception of the secret key.

- In order to achieve Kerckhoff's Principle in practice:
  Only use widely known ciphers that have been cryptanalyzed for several years by good cryptographers! (*Understanding Cryptography* only treats such ciphers)
- **Remark:** It is tempting to assume that a cipher is „more secure" if its details are kept secret. However, history has shown time and again that secret ciphers can almost always been broken once they have been reversed engineered. (Example: Content Scrambling System (CSS) for DVD content protection.)

Cryptanalysis: Attacking Cryptosystems



- **Classical Attacks**
  - Mathematical Analysis
  - Brute-Force Attack
- **Implementation Attack**: Try to extract key through reverese engineering or power measurement, e.g., for a banking smart card.
- **Social Engineering**: E.g., trick a user into giving up her password

Brute-Force Attack (or Exhaustive Key Search) against Symmetric Ciphers

- Treats the cipher as a black box
- Requires (at least) 1 plaintext-ciphertext pair (x_0, y_0)
- Check all possible keys until condition is fulfilled:

$$d_K(y_0) \stackrel{?}{=} x_0$$

- How many keys to we need ?

Key length in bit	Key space	Security life time (assuming brute-force as best possible attack)
64	2^{64}	**Short term** (few days or less)
128	2^{128}	**Long-term** (several decades in the absence of quantum computers)
256	2^{256}	**Long-term** (also resistant against quantum computers – note that QC do not exist at the moment and might never exist)

Important: An adversary only needs to succeed with **one** attack. Thus, a long key space does not help if other attacks (e.g., social engineering) are possible..

Content of this Chapter

- Overview on the field of cryptology
- Basics of symmetric cryptography
- Cryptanalysis
- **Substitution Cipher**
- Modular arithmetic
- Shift (or Caesar) Cipher and Affine Cipher

Substitution Cipher

- Historical cipher
- Great tool for understanding brute-force vs. analytical attacks
- Encrypts letters rather than bits (like all ciphers until after WW II)

Idea: replace each plaintext letter by a fixed other letter.

Plaintext		Ciphertext
A	→	k
B	→	d
C	→	w
	....	

for instance, `ABBA` would be encrypted as `kddk`

- Example (ciphertext):

```
iq ifcc vqqr fb rdq vfllcq na rdq cfjwhwz hr bnnb  hcc
                hwwhbsqvqbre hwq vhlq
```

- How secure is the Substitution Cipher? Let‘s look at attacks…

Attacks against the Substitution Cipher

1. **Attack: Exhaustive Key Search (Brute-Force Attack)**

- Simply try every possible subsititution table until an intelligent plaintext appears (note that each substitution table is a key)..
- How many substitution tables (= keys) are there?

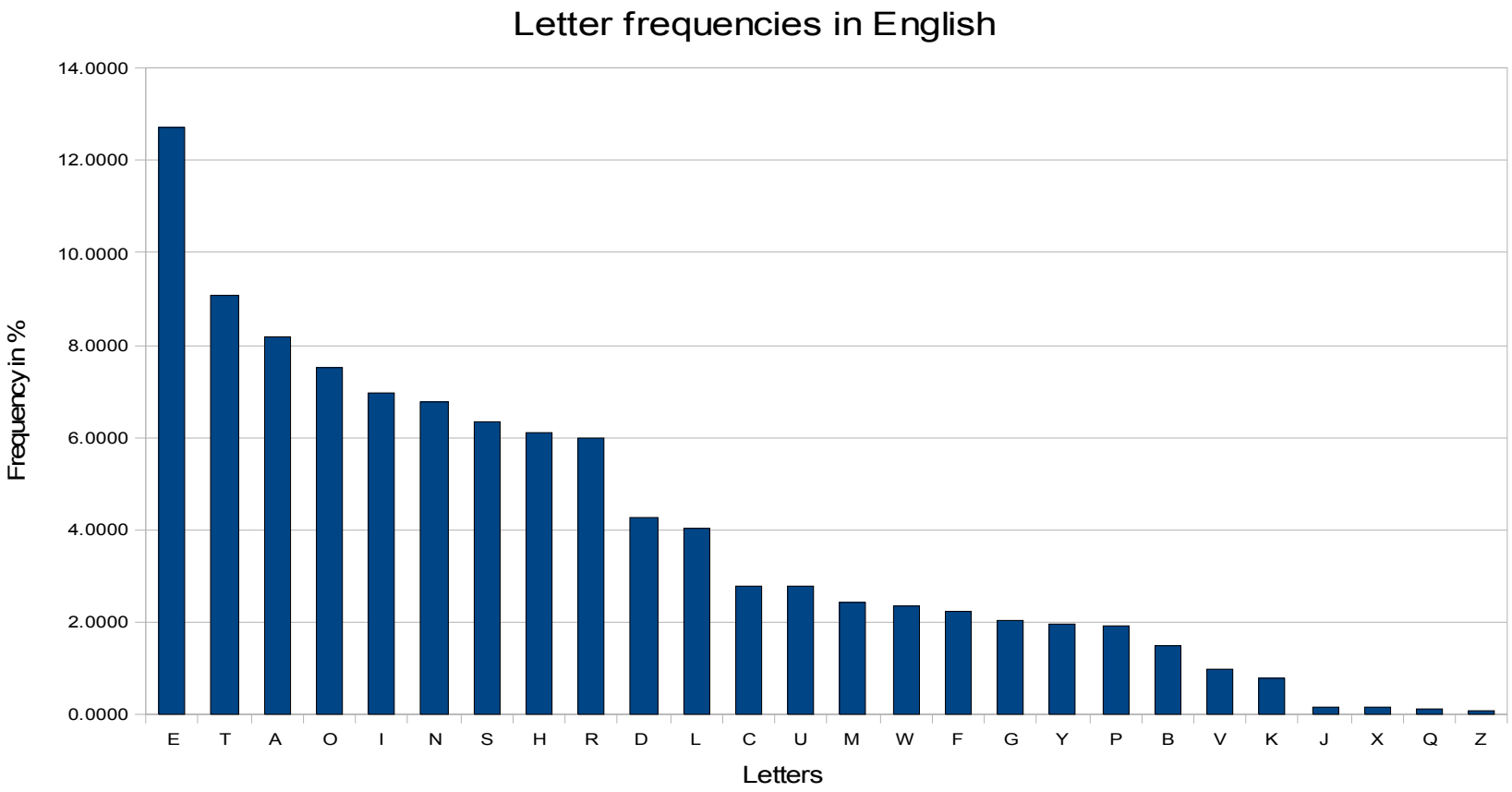
$$26 \times 25 \times \ldots \times 3 \times 2 \times 1 = 26! \approx 2^{88}$$

Search through 2^{88} keys is completely infeasible with today's computers! (cf. earlier table on key lengths)

- Q: Can we now conclude that the substitution cipher is secure since a brute-forece attack is not feasible?
- A: No! We have to protect against **all** possible attacks…

2. Attack: Letter Frequency Analysis (Mathematical Analysis Attack)

- Letters have very different frequencies in the English language
- Moreover: the frequency of plaintext letters is preserved in the ciphertext.
- For instanc, „e“ is the most common letter in English; almost 13% of all letters in a typical English text are „e“.
- The next most common one is „t“ with about 9%.



Breaking the Substitution Cipher with Letter Frequency Attack

- Let's retun to our example and identify the most frequent letter:

```
iq ifcc vqqr fb rdq vfllcq na rdq cfjwhwz hr bnnb hcc
hwwhbsqvqbre hwq vhlq
```

- We replace the ciphertext letter `q` by `E` and obtain:

```
iE ifcc vEEr fb rdE vfllcE na rdE cfjwhwz hr bnnb  hcc
hwwhbsEvEbre hwE vhlE
```

- By further guessing based on the frequency of the remaining letters we obtain the plaintext:

```
WE WILL MEET IN THE MIDDLE OF THE LIBRARY AT NOON ALL
ARRANGEMENTS ARE MADE
```

Breaking the Substitution Cipher with Letter Frequency Attack

- In practice, not only frequencies of individual letters can be used for an attack, but also the frequency of letter pairs (i.e., „th“ is very common in English), letter triples, etc.
- cf. Problem 1.1 in *Understanding Cryptography* for a longer ciphertext you can try to break!

Important lesson: Even though the substitution cipher has a sufficiently large key space of appr. 2^{88}, it can easily be defeated with analytical methods. This is an excellent example that an encryption scheme must withstand all types of attacks.

Content of this Chapter

- Overview on the field of cryptology
- Basics of symmetric cryptography
- Attacking crypto schemes
- Substitution Cipher
- **Modular arithmetic**
- Shift (or Caesar) Cipher and Affine Cipher

Short Introduction to Modular Arithmetic

Why do we need to study modular arithmetic?

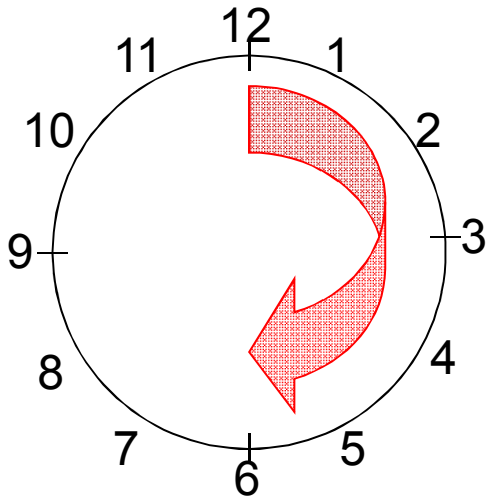
- Extremely important for asymmetric cryptography (RSA, elliptic curves etc.)
- Some historical ciphers can be elegantly described with modular arithmetic (cf. Caesar and affine cipher later on).

Short Introduction to Modular Arithmetic

Generally speaking, most cryptosytems are based on **sets of numbers** that are

1. **discrete** (sets with integers are particularly useful)
2. **finite** (i.e., if we only compute with a finiely many numbers)

Seems too abstract? --- Let's look at a finite set with discrete numbers we are quite familiar with: a clock.



Interestingly, even though the numbers are incremented every hour we never leave the set of integers:

1, 2, 3, … 11, 12, 1, 2, 3, … 11, 12, 1, 2, 3, …:

Short Introduction to Modular Arithmetic

- We develop now an arithmetic system which allows us to **compute** in finite sets of integers like the 12 integers we find on a clock (1,2,3, … ,12).
- It is crucial to have an operation which „keeps the numbers within limits“, i.e., after addition and multiplication they should never leave the set (i.e., never larger than 12).

Definition: Modulus Operation

Let *a, r, m* be integers and $m > 0$. We write

$$a \equiv r \bmod m$$

if *(r-a) is divisable by m.*

- “*m*” is called the **modulus**
- “*r*” is called the **remainder**

Examples for modular reduction.

- Let a= 12 and m= 9 : $12 \equiv 3 \bmod 9$
- Let a= 37 and m= 9: $34 \equiv 7 \bmod 9$
- Let a= -7 and m= 9: $-7 \equiv 2 \bmod 9$

(you should check whether the condition „*m* divides (*r-a*)“holds in each of the 3 cases)

Properties of Modular Arithmetic (1)

- **The remainder is not unique**

It is somewhat surprising that for every given modulus *m* and number *a*, there are (infinitely) many valid remainders.

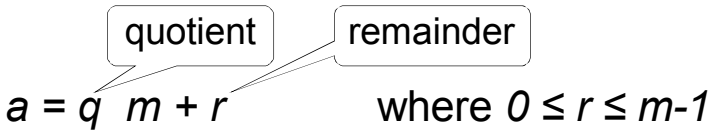
Example:

- $12 \equiv 3 \bmod 9$ → 3 is a valid remainder since 9 divides (3-12)
- $12 \equiv 21 \bmod 9$ → 21 is a valid remainder since 9 divides (21-12)
- $12 \equiv -6 \bmod 9$ → -6 is a valid remainder since 9 divides (-6-12)

Properties of Modular Arithmetic (2)

- **Which remainder do we choose?**

By convention, we usually agree on the **smallest positive integer *r*** as remainder. This integer can be computed as



- Example: *a=12* and *m= 9*

$$12 = 1 \times 9 + 3 \quad \rightarrow \quad r = 3$$

Remark: This is just a convention. Algorithmically we are free to choose any other valid remainder to compute our crypto functions.

Properties of Modular Arithmetic (3)

- **How do we perform modular division?**

First, note that rather than performing a division, we prefer to multiply by the inverse. Ex:

$$b / a \equiv b \times a^{-1} \bmod m$$

The inverse a^{-1} of a number a is defined such that:

$$a\, a^{-1} \equiv 1 \bmod m$$

Ex: What is $5 / 7 \bmod 9$?

The inverse of $7 \bmod 9$ is 4 since $7 \times 4 \equiv 28 \equiv 1 \bmod 9$, hence:

$$5 / 7 \equiv 5 \times 4 = 20 \equiv 2 \bmod 9$$

- **How is the inverse compute?**

The inverse of a number $a \bmod m$ only exists if and only if:

$$\gcd(a, m) = 1$$

(note that in the example above gcd(5, 9) = 1, so that the inverse of 5 exists modulo 9)

For now, the best way of computing the inverse is to use exhaustive search. In Chapter 6 of *Understanding Cryptography* we will learn the powerful Euclidean Algorithm which actually computes an inverse for a given number and modulus.

Properties of Modular Arithmetic (4)

- **Modular reduction can be performed at any point during a calculation**

Let's look first at an example. We want to compute $3^8 \bmod 7$ (note that exponentiation is extremely important in public-key cryptography).

1. Approach: Exponentiation followed by modular reduction

$$3^8 = 6561 \equiv \mathbf{2} \bmod 7$$

Note that we have the intermediate result 6561 even though we know that the final result can't be larger than 6.

2. Approach: Exponentiation with intermediate modular reduction

$$3^8 = 3^4 \, 3^4 = 81 \times 81$$

At this point we reduce the intermediate results 81 modulo 7:

$$3^8 = 81 \times 81 \equiv 4 \times 4 \bmod 7$$

$$4 \times 4 = 16 \equiv \mathbf{2} \bmod 7$$

Note that we can perform all these multiplications without pocket calculator, whereas mentally computing $3^8 = 6561$ is a bit challenging for most of us.

General rule: For most algorithms it is advantageous to reduce intermediate results as soon as possible.

An Algebraic View on Modulo Arithmetic: The Ring Z_m (1)

We can view modular arithmetic in terms of sets and operations in the set. By doing arithmetic modulo m we obtain **the integer ring Z_m** .with the following properties:

- **Closure**: We can add and multiply any two numbers and the result is always in the ring.
- Addition and multiplication are **associative**, i.e., for all $a,b,c \in Z_m$

  $a + (b + c) = (a + b) + c$

  $a \times (b \times c) = (a \times b) \times c$

  and addition is **commutative**: $a + b = b + a$
- The **distributive law** holds: $a \times (b+c) = (a \times b) + (a \times c)$ for all $a,b,c \in Z_m$
- There is the **neutral element *0* with respect to addition**, i.e., for all $a \in Z_m$

  $a + 0 \equiv a \bmod m$
- For all $a \in Z_m$, there is always an **additive inverse element $-a$** such that

  $a + (-a) \equiv 0 \bmod m$
- There is the **neutral element 1 with respect to multiplication**, i.e., for all $a \in Z_m$

  $a \times 1 \equiv a \bmod m$
- The **multiplicative inverse a^{-1}**

  $a \times a^{-1} \equiv 1 \bmod m$

  exists only for some, but not for all, elements in Z_m.

An Algebraic View on Modulo Arithmetic: The Ring $Z_m(2)$

Roughly speaking, a ring is a structure in which we can always add, subtract and multiply, but we can only divide by certain elements (namely by those for which a multiplicative inverse exists).

- We recall from above that an element $a \varepsilon Z_m$ has a multiplicative inverse only if:

  gcd *(a, m) = 1*

  We say that *a* is **coprime** or **relatively prime** to *m*.

- Ex: We consider the ring $Z_9 = \{0,1,2,3,4,5,6,7,8\}$

  The elements 0, 3, and 6 do not have inverses since they are not coprime to 9.

  The inverses of the other elements 1, 2, 4, 5, 7, and 8 are:

  $1^{-1} \equiv 1 \bmod 9$ $\quad 2^{-1} \equiv 5 \bmod 9$ $\quad 4^{-1} \equiv 7 \bmod 9$

  $5^{-1} \equiv 2 \bmod 9$ $\quad 7^{-1} \equiv 4 \bmod 9$ $\quad 8^{-1} \equiv 8 \bmod 9$

Content of this Chapter

- Overview on the field of cryptology
- Basics of symmetric cryptography
- Attacking crypto schemes
- Substitution Cipher
- Modular arithmetic
- **Shift (or Caesar) Cipher and Affine Cipher**

Shift (or Caesar) Cipher (1)

- Ancient cipher, allegedly used by Julius Caesar
- Replaces each plaintext letter by another one.
- Replacement rule is very simple: Take letter that follows after *k* positions in the alphabet

Needs mapping from letters → numbers:

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

- Example for $k = 7$

Plaintext = `ATTACK` = 0, 19, 19, 0, 2, 10

Ciphertext = `haahr` = 7, 0, 0, 7, 17

Note that the letters ”wrap around” at the end of the alphabet, which can be mathematically be expressed as reduction modulo 26, e.g., $19 + 7 = 26 \equiv 0 \bmod 26$

Shift (or Caesar) Cipher (2)

- Elegant mathematical description of the cipher.

Let k, x, y ε {0,1, …, 25}

- Encryption: $y = e_k(x) \equiv x + k \bmod 26$
- Decryption: $x = d_k(x) \equiv y - k \bmod 26$

- Q; Is the shift cipher secure?
- A: No! several attacks are possible, including:
  - Exhaustive key search (key space is only 26!)
  - Letter frequency analysis, similar to attack against substitution cipher

Affine Cipher (1)

- Extension of the shift cipher: rather than just adding the key to the plaintext, we also multiply by the key
- We use for this a key consisting of two parts: k = (a, b)

Let k, x, y ε {0,1, …, 25}

- Encryption: $y = e_k(x) \equiv a\,x + b \bmod 26$
- Decryption: $x = d_k(x) \equiv a^{-1}(y - b) \bmod 26$

- Since the inverse of *a* is needed for inversion, we can only use values for *a* for which:

$$\gcd(a, 26) = 1$$

  There are 12 values for *a* that fulfill this condition.
- From this follows that the key space is only 12 x 26 = 312 (cf. Sec 1.4 in *Understanding Cryptography*)
- Again, several attacks are possible, including:
  - Exhaustive key search and letter frequency analysis, similar to the attack against the substitution cipher

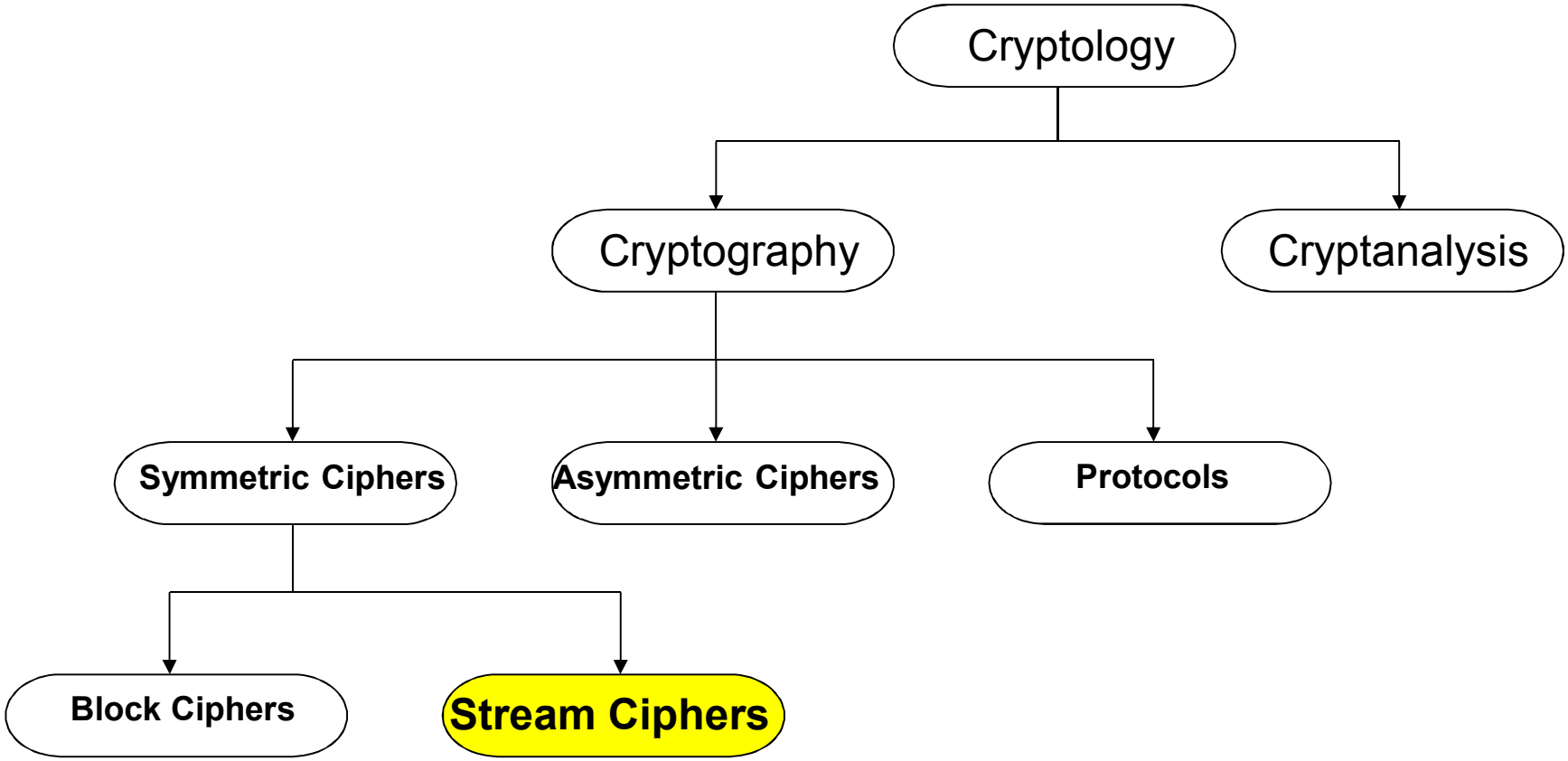
Lessons Learned

- Never ever develop your own crypto algorithm unless you have a team of experienced cryptanalysts checking your design.
- Do not use unproven crypto algorithms or unproven protocols.
- Attackers always look for the weakest point of a cryptosystem. For instance, a large key space by itself is no guarantee for a cipher being secure; the cipher might still be vulnerable against analytical attacks.
- Key lengths for symmetric algorithms in order to thwart exhaustive key-search attacks:
  - 64 bit: insecure except for data with extremely short-term value
  - 128 bit: long-term security of several decades, unless quantum computers become available (quantum computers do not exist and perhaps never will)
  - 256 bit: as above, but probably secure against attacks by quantum computers.
- Modular arithmetic is a tool for expressing historical encryption schemes, such as the affine cipher, in a mathematically elegant way.

Content of this Chapter

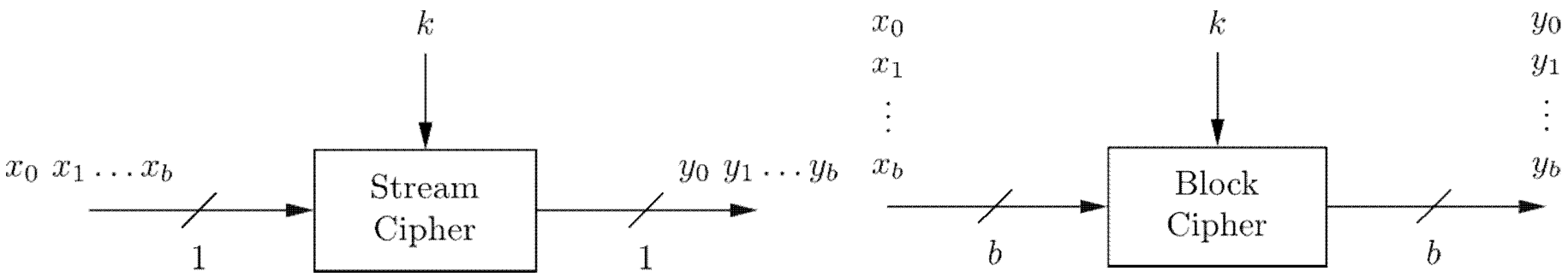
- **Intro to stream ciphers**
- Random number generators (RNGs)
- One-Time Pad (OTP)
- Linear feedback shift registers (LFSRs)
- Trivium: a modern stream cipher

Stream Ciphers in the Field of Cryptology



Stream Ciphers were invented in 1917 by Gilbert Vernam

Stream Cipher vs. Block Cipher

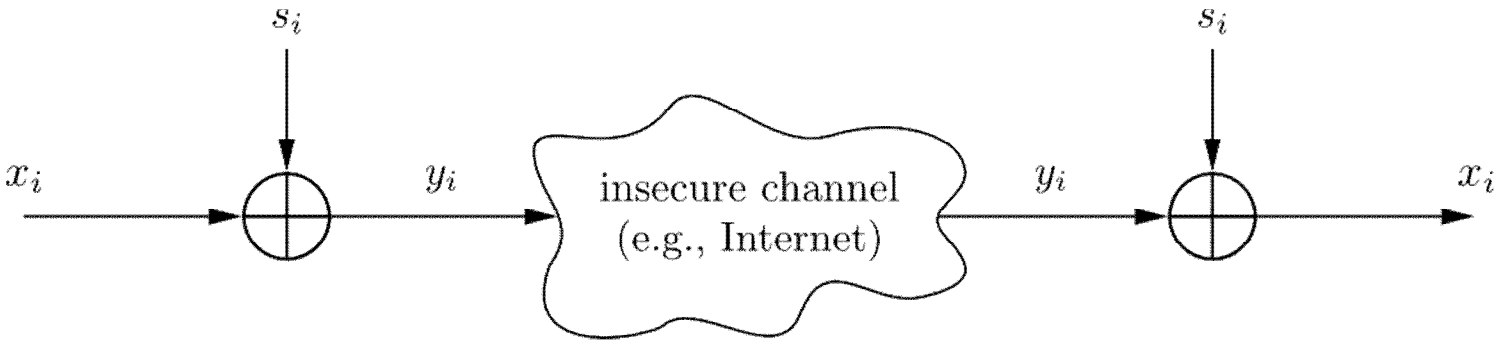


- **Stream Ciphers**
  - Encrypt bits individually
  - Usually small and fast → common in embedded devices (e.g., A5/1 for GSM phones)

- **Block Ciphers:**
  - Always encrypt a full block (several bits)
  - Are common for Internet applications

Encryption and Decryption with Stream Ciphers

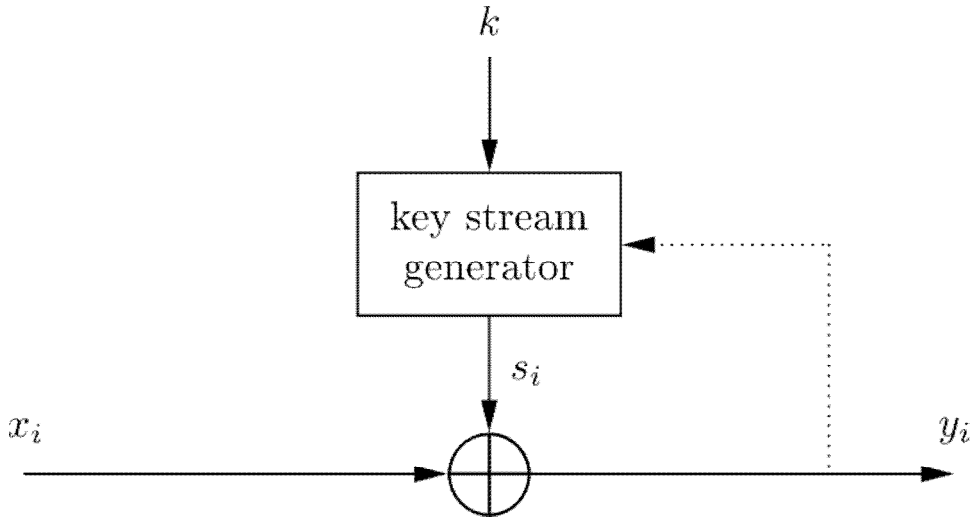
Plaintext x_i, ciphertext y_i and key stream s_i consist of individual bits



- Encryption and decryption are simple additions modulo 2 (aka XOR)
- Encryption and decryption are the same functions

- **Encryption:** $y_i = e_{si}(x_i) = x_i + s_i \bmod 2 \qquad x_i, y_i, s_i \in \{0,1\}$
- **Decryption:** $x_i = e_{si}(y_i) = y_i + s_i \bmod 2$

Synchronous vs. Asynchronous Stream Cipher



- Security of stream cipher depends entirely on the key stream s_i :
  - Should be **random** , i.e., $Pr(s_i = 0) = Pr(s_i = 1) = 0.5$
  - Must be **reproducible** by sender and receiver

- **Synchronous Stream Cipher**
  - Key stream depend only on the key (and possibly an initialization vector IV)

- **Asynchronous Stream Ciphers**
  - Key stream depends also on the ciphertext (dotted feedback enabled)

Why is Modulo 2 Addition a Good Encryption Function?

- Modulo 2 addition is equivalent to XOR operation

- For perfectly random key stream s_i, each ciphertext output bit has a 50% chance to be 0 or 1

  → Good statistic property for ciphertext

- Inverting XOR is simple, since it is the same XOR operation

x_i	s_i	y_i
0	0	0
0	1	1
1	0	1
1	1	0

Stream Cipher: Throughput

Performance comparison of symmetric ciphers (Pentium4):

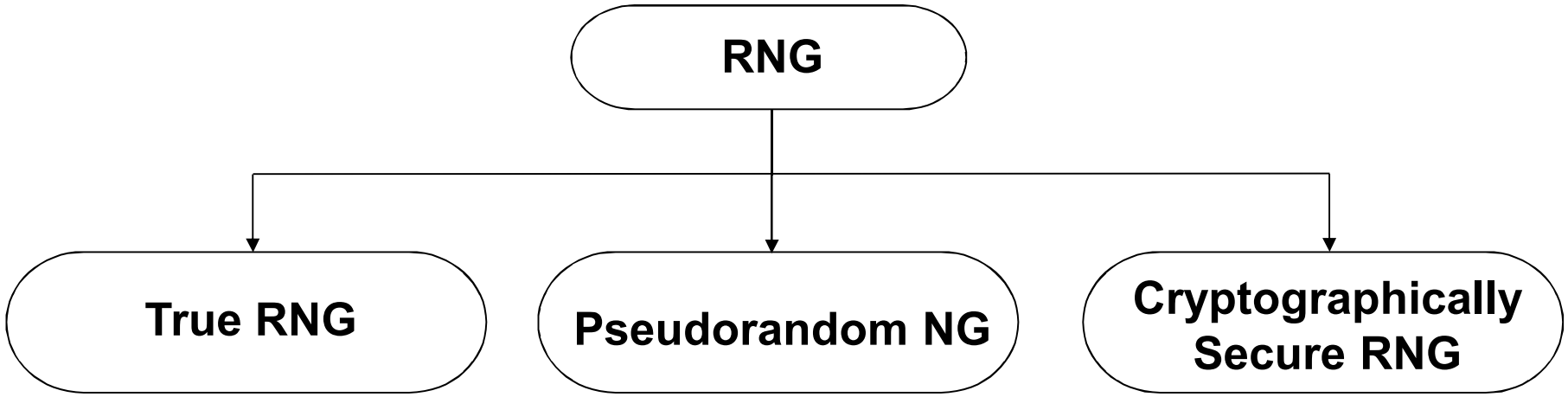
Cipher	Key length	Mbit/s
DES	56	36.95
3DES	112	13.32
AES	128	51.19
RC4 (stream cipher)	(choosable)	211.34

Source: Zhao et al., Anatomy and Performance of SSL Processing, ISPASS 2005

Content of this Chapter

- Intro to stream ciphers
- **Random number generators (RNGs)**
- One-Time Pad (OTP)
- Linear feedback shift registers (LFSRs)
- Trivium: a modern stream cipher

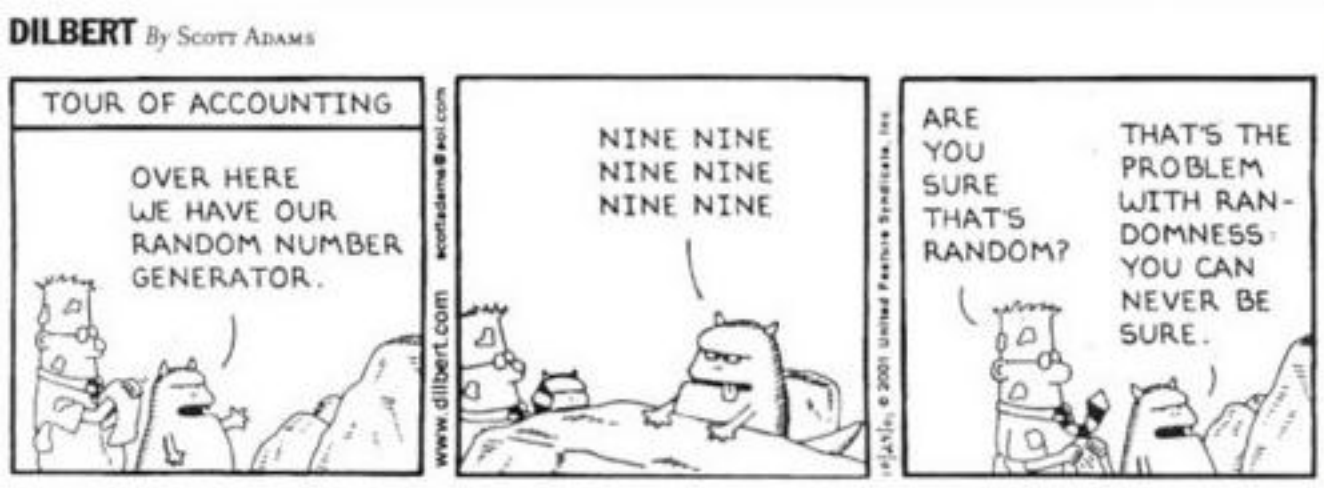
- Random number generators (RNGs)



True Random Number Generators (TRNGs)

- Based on physical random processes: coin flipping, dice rolling, semiconductor noise, radioactive decay, mouse movement, clock jitter of digital circuits
- Output stream s_i should have good statistical properties:
  $\Pr(s_i = 0) = \Pr(s_i = 1) = 50\%$ (often achieved by post-processing)
- Output can neither be predicted nor be reproduced

Typically used for generation of keys, nonces (used only-once values) and for many other purposes



Pseudorandom Number Generator (PRNG)

- Generate sequences from initial seed value
- Typically, output stream has good statistical properties
- Output can be reproduced and can be predicted

Often computed in a recursive way:

$$s_0 = seed$$

$$s_{i+1} = f(s_i, s_{i-1}, \ldots, s_{i-t})$$

Example: *rand() function* in ANSI C:

$$s_0 = 12345$$

$$s_{i+1} = 1103515245 s_i + 12345 \bmod 2^{31}$$

Most PRNGs have bad cryptographic properties!

Cryptanalyzing a Simple PRNG

Simple PRNG: **Linear Congruential Generator**

$$S_0 = seed$$

$$S_{i+1} = AS_i + B \bmod m$$

Assume

- unknown A, B *and* S_0 as key
- Size of A, B and S_i to be 100 bit
- 300 bit of output are known, i.e. S_1, S_2 and S_3

Solving

$$S_2 = AS_1 + B \bmod m$$

$$S_3 = AS_2 + B \bmod m$$

...directly reveals A and B. All S_i can be computed easily!

Bad cryptographic properties due to the linearity of most PRNGs

Cryptographically Secure Pseudorandom Number Generator (CSPRNG)

- Special PRNG with additional property:
  - Output must be **unpredictable**

More precisely: Given n consecutive bits of output s_i, the following output bits s_{n+1} cannot be predicted (in polynomial time).

- Needed in cryptography, in particular for stream ciphers

- Remark: There are almost no other applications that need unpredictability, whereas many, many (technical) systems need PRNGs.

Content of this Chapter

- Intro to stream ciphers
- Random number generators (RNGs)
- **One-Time Pad (OTP)**
- Linear feedback shift registers (LFSRs)
- Trivium: a modern stream cipher

One-Time Pad (OTP)

Unconditionally secure cryptosystem:

- A cryptosystem is unconditionally secure if it cannot be broken even with *infinite* computational resources

One-Time Pad

- A cryptosystem developed by Mauborgne that is based on Vernam's stream cipher:
- Properties:

  Let the plaintext, ciphertext and key consist of individual bits
  $x_i, y_i, k_i \in \{0,1\}$.

Encryption:	$e_{k_i}(x_i) = x_i \oplus k_i$.
Decryption:	$d_{k_i}(y_i) = y_i \oplus k_i$

OTP is unconditionally secure if and only if the key k_i is used once!

One-Time Pad (OTP)

Unconditionally secure cryptosystem:

$$y_0 = x_0 \oplus k_0$$
$$y_1 = x_1 \oplus k_1$$
$$:$$

Every equation is a linear equation with two unknowns

$\Rightarrow$ for every y_i are $x_i = 0$ and $x_i = 1$ equiprobable!

$\Rightarrow$This is true iff k_0, k_1, ... are independent, i.e., all k_i have to be generated truly random

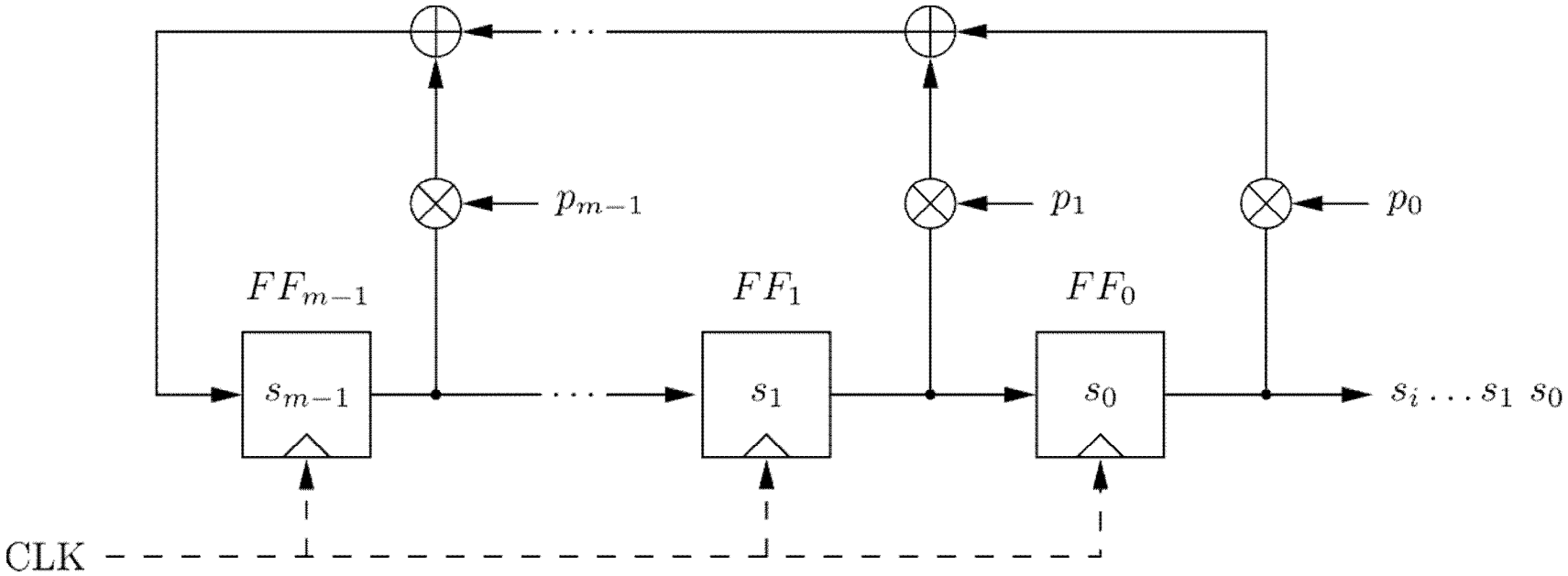
$\Rightarrow$ It can be shown that this systems can *provably* not be solved.

Disadvantage: For almost all applications the OTP is **impractical** since the key must be as long as the message! (Imagine you have to encrypt a 1GByte email attachment.)

Content of this Chapter

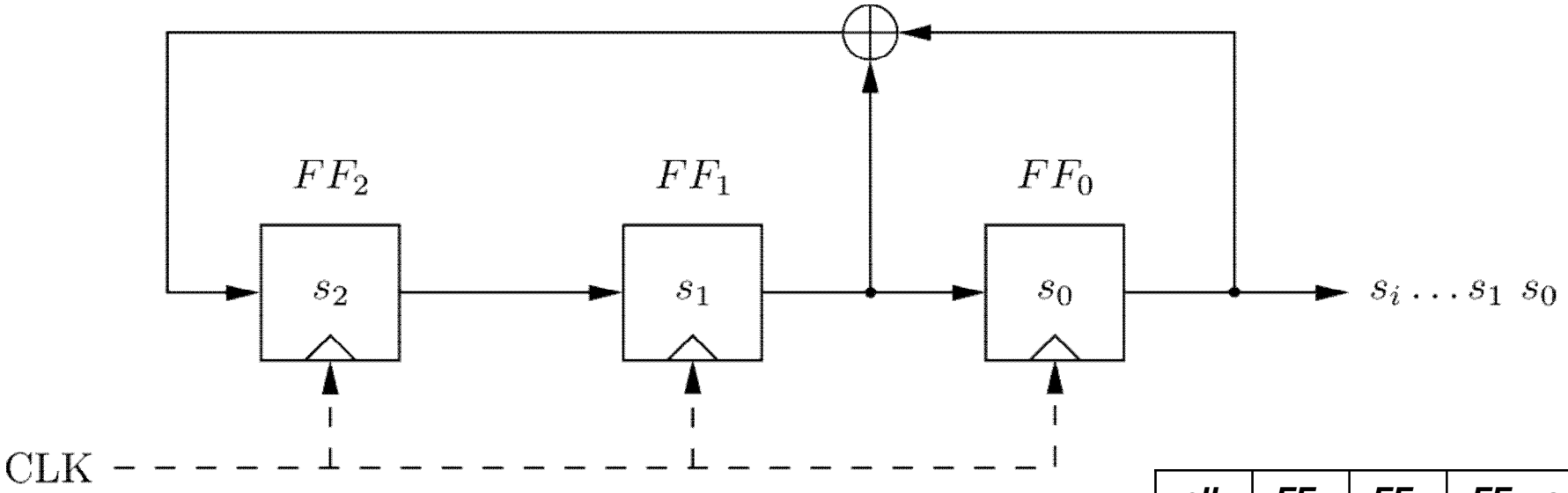
- Intro to stream ciphers
- Random number generators (RNGs)
- One-Time Pad (OTP)
- **Linear feedback shift registers (LFSRs)**
- Trivium: a modern stream cipher

Linear Feedback Shift Registers (LFSRs)



- Concatenated *flip-flops (FF)*, i.e., a shift register together with a feedback path
- Feedback computes fresh input by XOR of certain state bits
- *Degree m* given by number of storage elements
- If $p_i = 1$, the feedback connection is present ("closed switch), otherwise there is not feedback from this flip-flop ("open switch")
- Output sequence repeats periodically
- Maximum output length: 2^m-1

Linear Feedback Shift Registers (LFSRs): Example with m=3



- LFSR output described by recursive equation:

$$s_{i+3} = s_{i+1} + s_i \bmod 2$$

- Maximum output length (of 2^3-1=7) achieved only for certain feedback configurations, .e.g., the one shown here.

clk	FF_2	FF_1	$FF_0=s_i$
0	1	0	0
1	0	1	0
2	1	0	1
3	1	1	0
4	1	1	1
5	0	1	1
6	0	0	1
7	1	0	0
8	0	1	0

Security of LFSRs

LFSRs typically described by polynomials:

$$P(x) = x^m + p_{l-1}x^{m-1} + \ldots + p_1 x + p_0$$

- Single LFSRs generate highly predictable output
- If $2m$ output bits of an LFSR of degree m are known, the feedback coefficients p_i of the LFSR can be found by solving a system of linear equations*

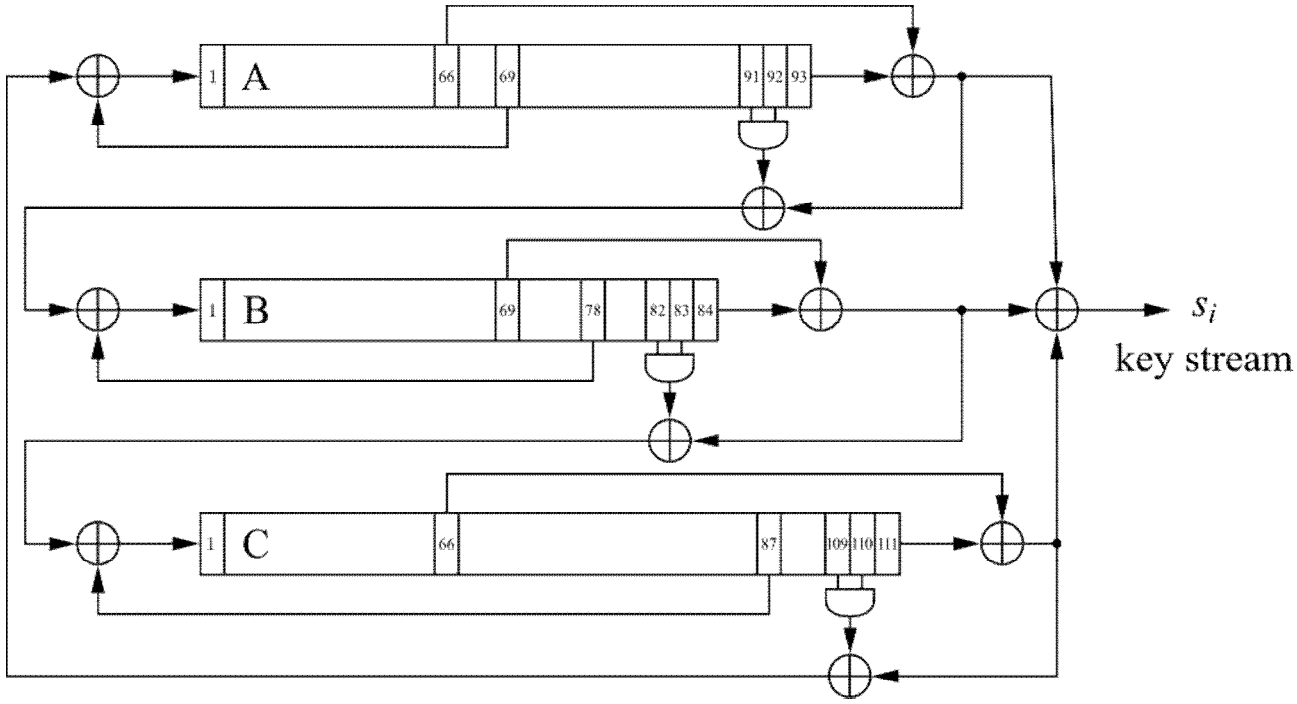
- Because of this many stream ciphers use **combinations** of LFSRs

*See Chapter 2 of *Understanding Cryptography* for further details.

Content of this Chapter

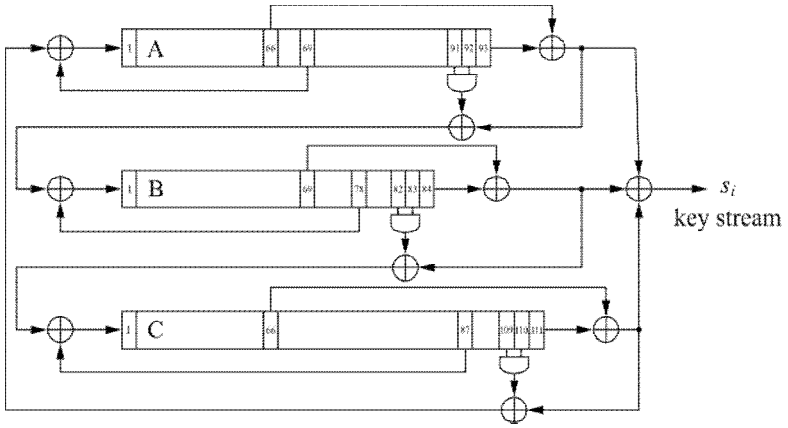
- Intro to stream ciphers
- Random number generators (RNGs)
- One-Time Pad (OTP)
- Linear feedback shift registers (LFSRs)
- **Trivium: a modern stream cipher**

A Modern Stream Cipher - Trivium



- Three *nonlinear* LFSRs (NLFSR) of length 93, 84, 111
- XOR-Sum of all three NLFSR outputs generates key stream s_i
- Small in Hardware:
  - Total register count: 288
  - Non-linearity: 3 AND-Gates
  - 7 XOR-Gates (4 with three inputs)

Trivium



Initialization:

- Load 80-bit IV into A
- Load 80-bit key into B
- Set c_{109} , c_{110} , c_{111} =1, all other bits 0

Warm-Up:

- Clock cipher 4 x 288 = 1152 times without generating output

Encryption:

- XOR-Sum of all three NLFSR outputs generates key stream s_i

Design can be parallelized to produce up to 64 bits of output per clock cycle

	Register length	Feedback bit	Feedforward bit	AND inputs
A	93	69	66	91, 92
B	84	78	69	82, 83
C	111	87	66	109, 110

Lessons Learned

- Stream ciphers are less popular than block ciphers in most domains such as Internet security. There are exceptions, for instance, the popular stream cipher RC4.

- Stream ciphers sometimes require fewer resources, e.g., code size or chip area, for implementation than block ciphers, and they are attractive for use in constrained environments such as cell phones.

- The requirements for a *cryptographically secure pseudorandom number generator* are far more demanding than the requirements for pseudorandom number generators used in other applications such as testing or simulation

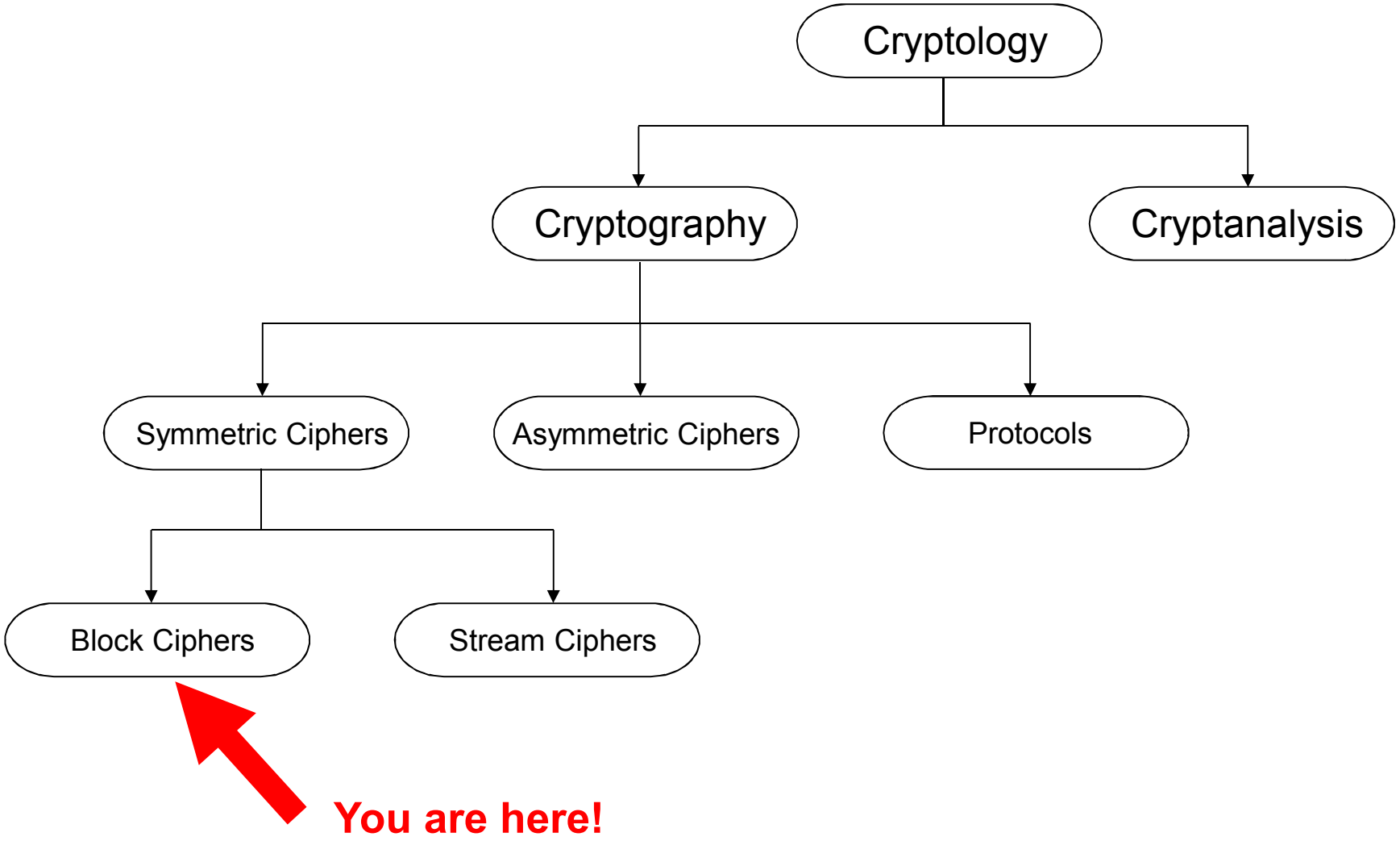
- The One-Time Pad is a provable secure symmetric cipher. However, it is highly impractical for most applications because the key length has to equal the message length.

- Single LFSRs make poor stream ciphers despite their good statistical properties. However, careful combinations of several LFSR can yield strong ciphers.

Content of this Chapter

- **Introduction to DES**
- Overview of the DES Algorithm
- Internal Structure of DES
- Decryption
- Security of DES

Classification of DES in the Field of Cryptology



DES Facts

- Data Encryption Standard (DES) encrypts **blocks of size 64 bit**.
- Developed by **IBM** based on the cipher *Lucifer* under influence of the *National Security Agency* (NSA), the design criteria for DES have not been published
- **Standardized 1977** by the **National Bureau of Standards** (NBS)
  today called *National Institute of Standards and Technology* (NIST)
- Most popular **block cipher** for most of the last 30 years.
- By far best studied symmetric algorithm.
- Nowadays considered insecure due to the small **key length of 56 bit.**
- **But: 3DES yields very secure cipher**, still widely used today.
- Replaced by the *Advanced Encryption Standard* (**AES**) in 2000

- For a more detailed history see Chapter 3.1 in *Understanding Cryptography*

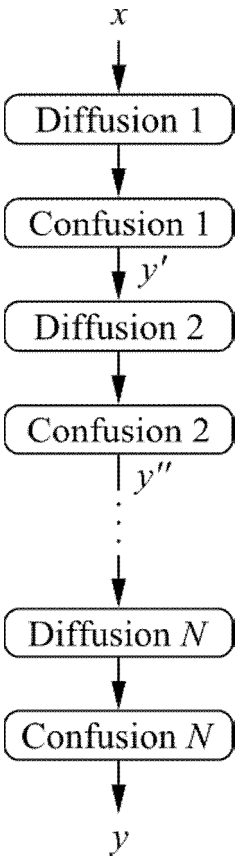
Block Cipher Primitives: Confusion and Diffusion

- Claude Shannon: There are two primitive operations with which strong encryption algorithms can be built:
  1. **Confusion:** An encryption operation where the **relationship between key and ciphertext is obscured**.
     Today, a common element for achieving confusion is **substitution**, which is found in both AES and DES.

  2. **Diffusion:** An encryption operation where the **influence of one plaintext symbol is spread over many ciphertext symbols** with the goal of hiding statistical properties of the plaintext.
     A simple diffusion element is the **bit permutation**, which is frequently used within DES.

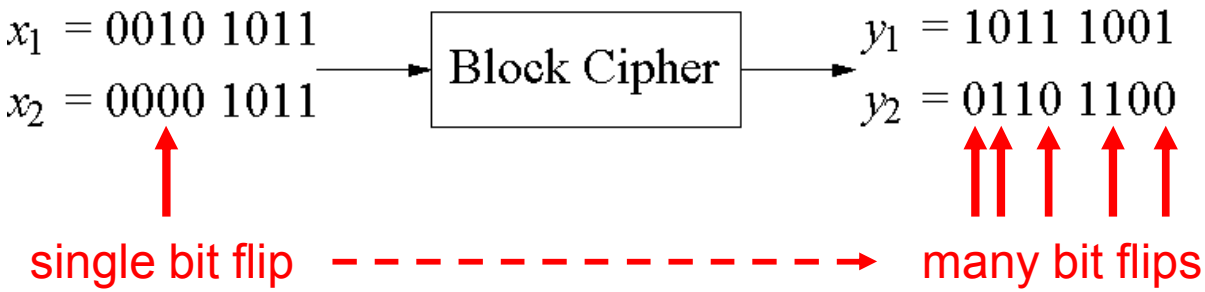
- Both operations by themselves cannot provide security. The idea is to concatenate confusion and diffusion elements to build so called *product ciphers.*

Product Ciphers



- Most of today's block ciphers are *product ciphers* as they consist of rounds which are applied repeatedly to the data.
- Can reach excellent diffusion: **changing of one bit of plaintext results** *on average* in the **change of half the output bits**.

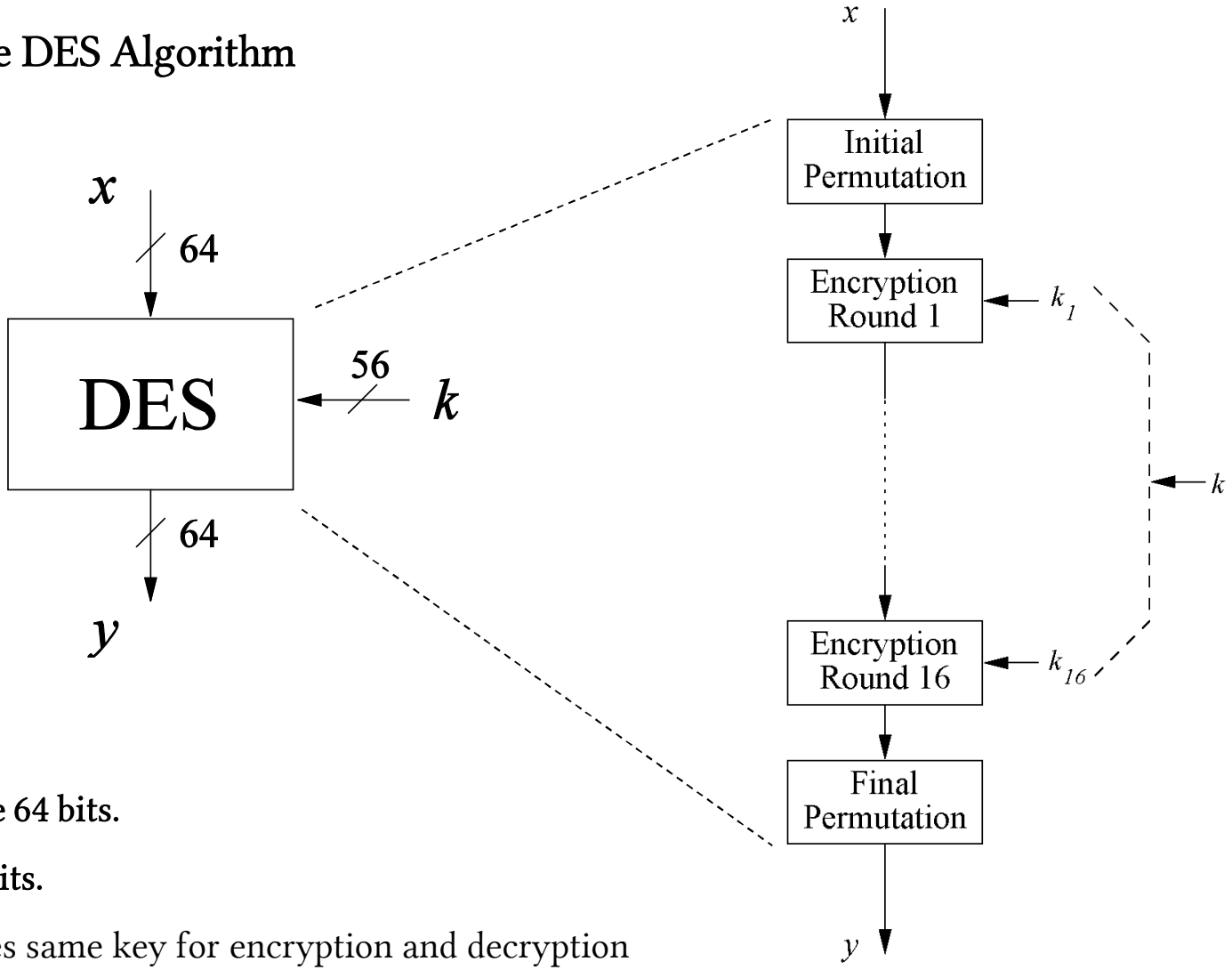
Example:



Content of this Chapter

- Introduction to DES
- **Overview of the DES Algorithm**
- Internal Structure of DES
- Decryption
- Security of DES

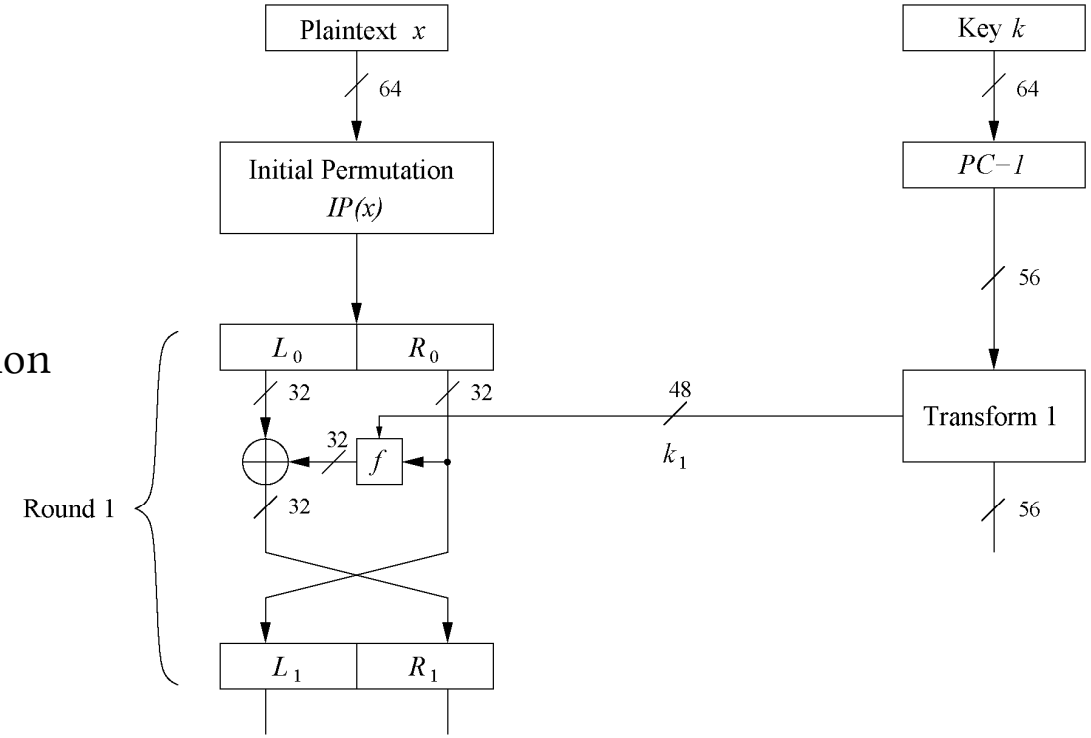
Overview of the DES Algorithm



- Encrypts blocks of size 64 bits.
- Uses a key of size 56 bits.
- Symmetric cipher: uses same key for encryption and decryption
- Uses 16 rounds which all perform the identical operation
- Different subkey in each round derived from main key

The DES Feistel Network (1)

- DES structure is a *Feistel network*
- Advantage: encryption and decryption differ only in keyschedule

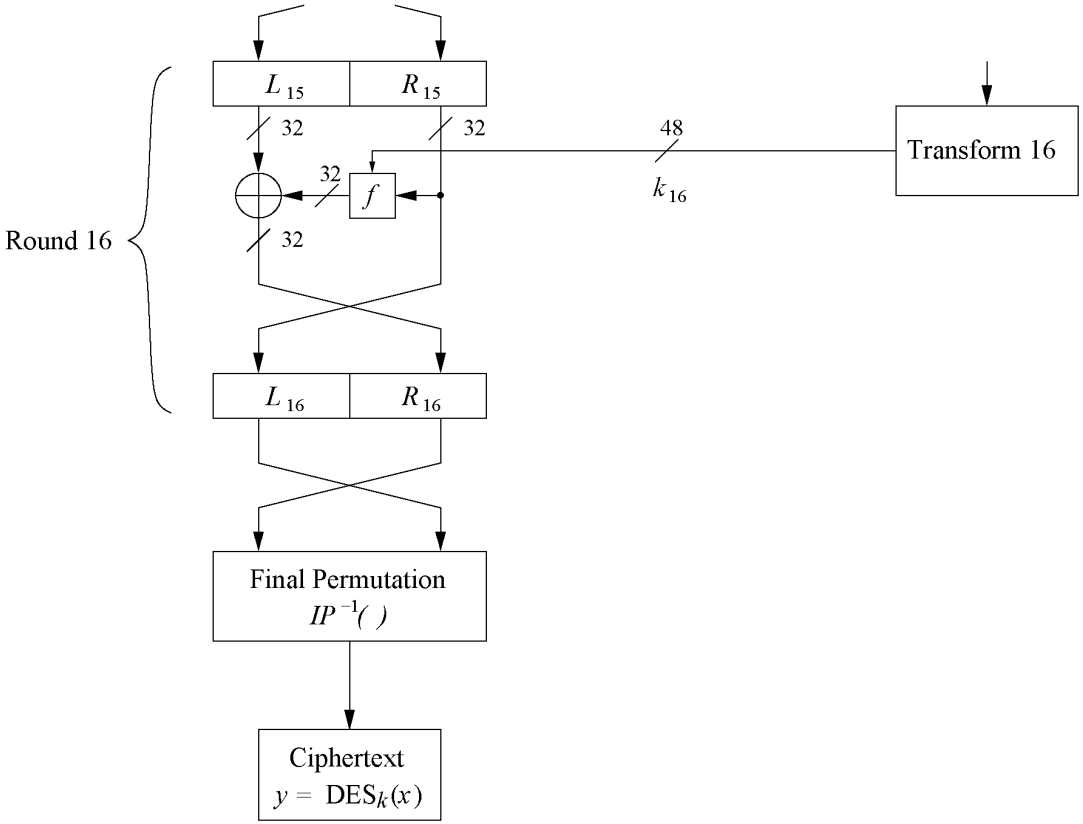


- Bitwise initial permutation, then 16 rounds
  1. Plaintext is split into 32-bit halves L_i and R_i
  2. R_i is fed into the function f, the output of which is then XORed with L_i
  3. Left and right half are swapped
- Rounds can be expressed as:

$$L_i = R_{i-1},$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, k_i)$$

The DES Feistel Network (2)

- L and R swapped again at the end of the cipher, i.e., after round 16 followed by a final permutation



Content of this Chapter

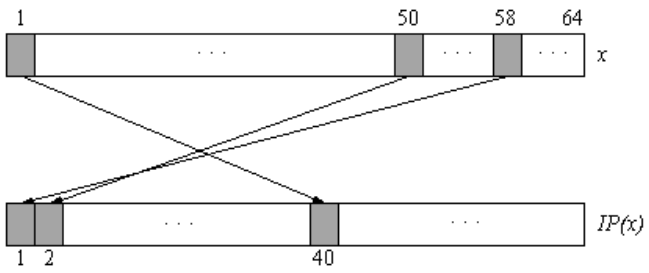
- Introduction to DES
- Overview of the DES Algorithm
- **Internal Structure of DES**
- Decryption
- Security of DES

Initial and Final Permutation

- Bitwise Permutations.
- Inverse operations.
- Described by tables IP and IP^{-1}.

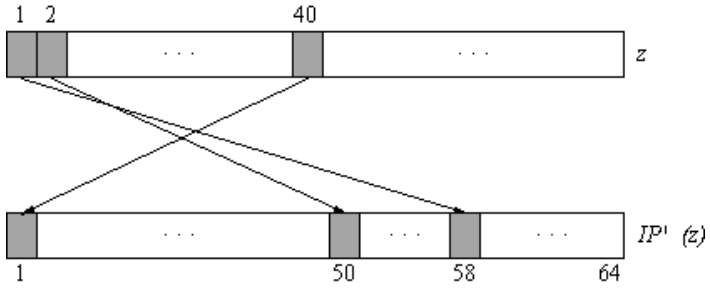
Initial Permutation

IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7



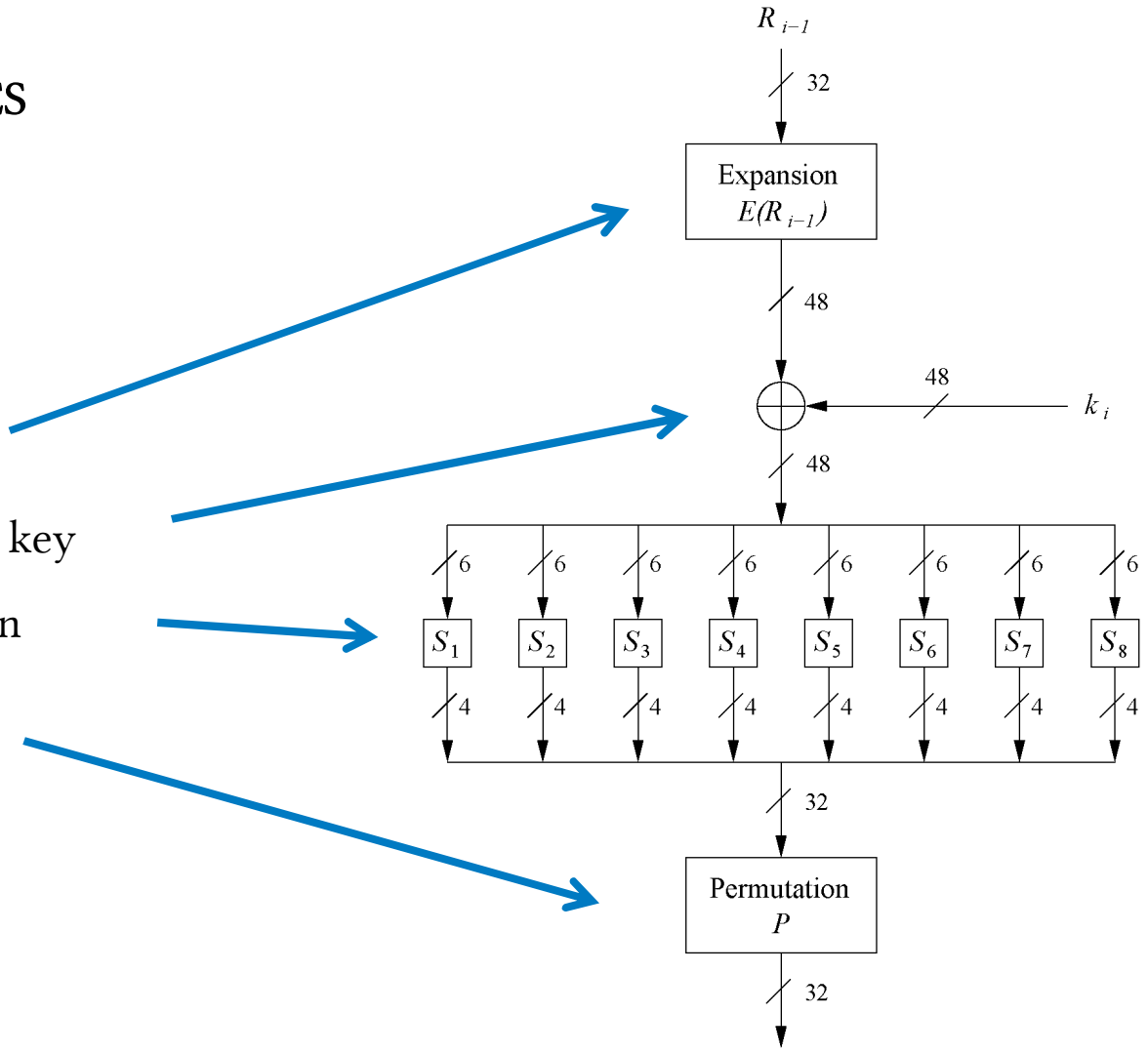
Final Permutation

IP^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25



The f-Function

- main operation of DES
- *f*-Function inputs:
  R_{i-1} and round key k_i
- 4 Steps:
  1. Expansion E
  2. XOR with round key
  3. S-box substitution
  4. Permutation

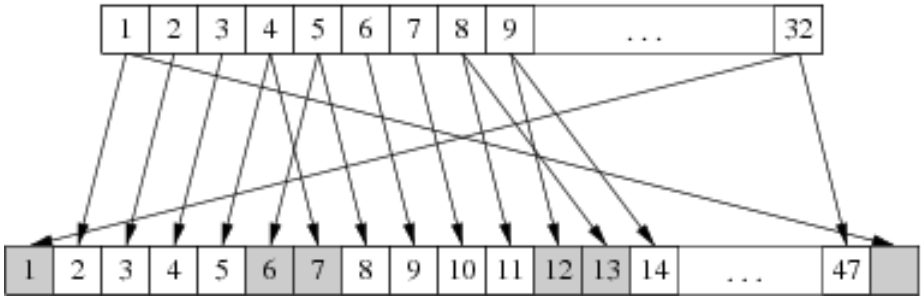


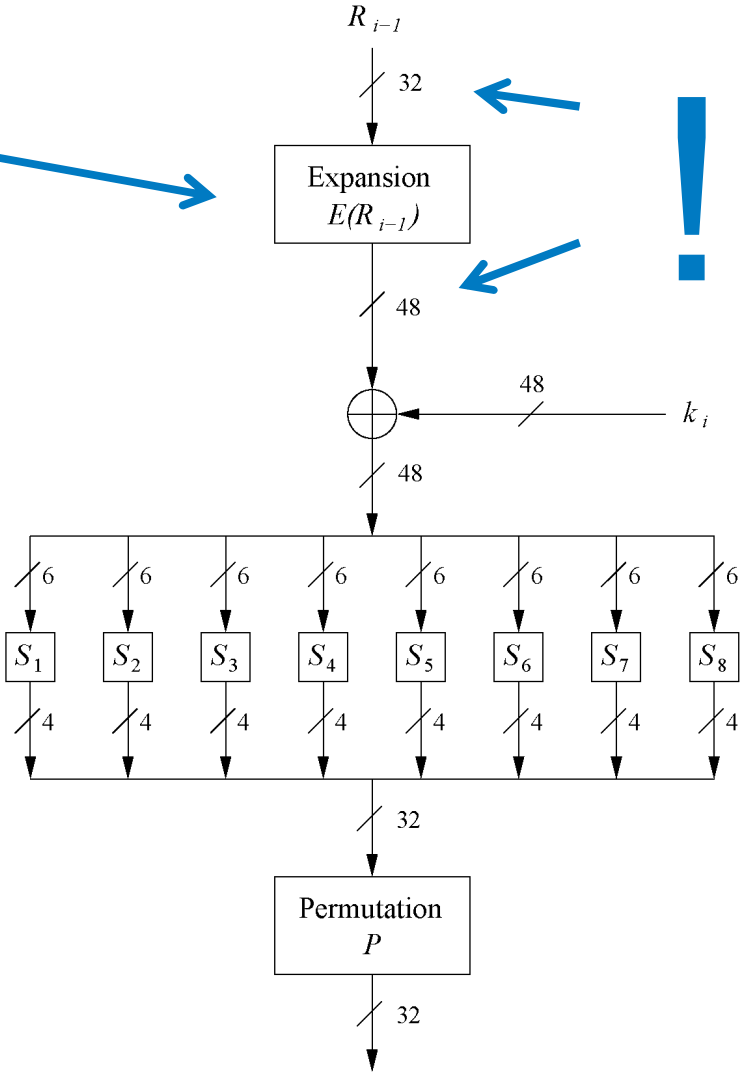
The Expansion Function E

1. Expansion E

- main purpose: increases diffusion

E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

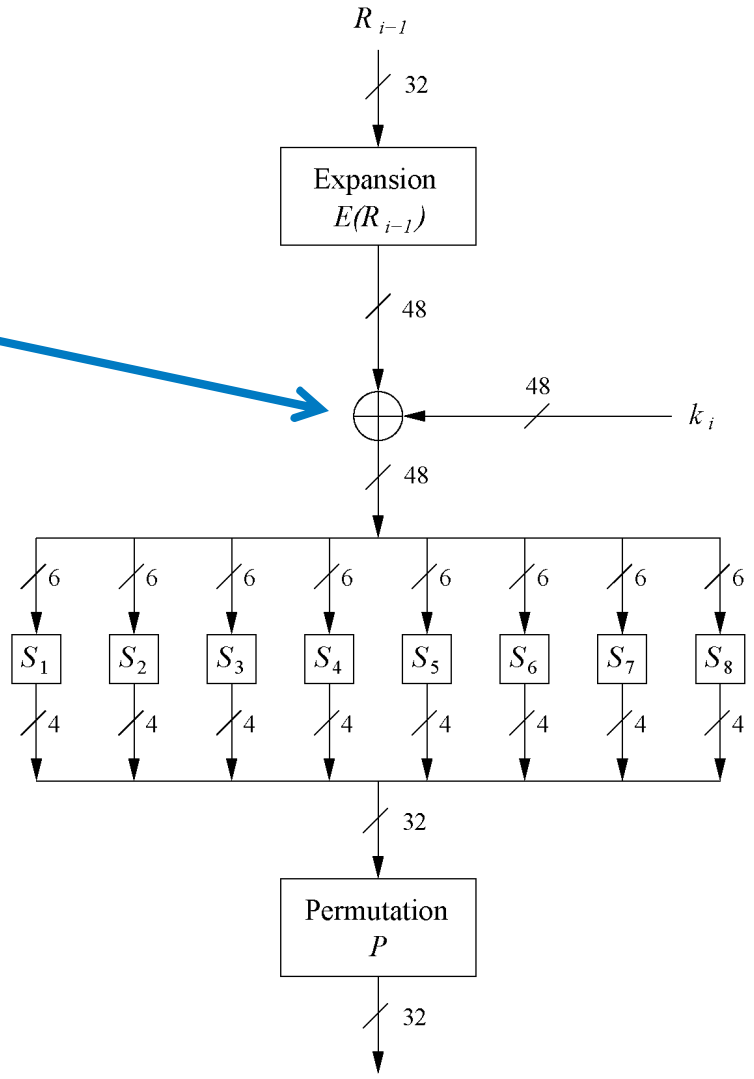




Add Round Key

2. XOR Round Key

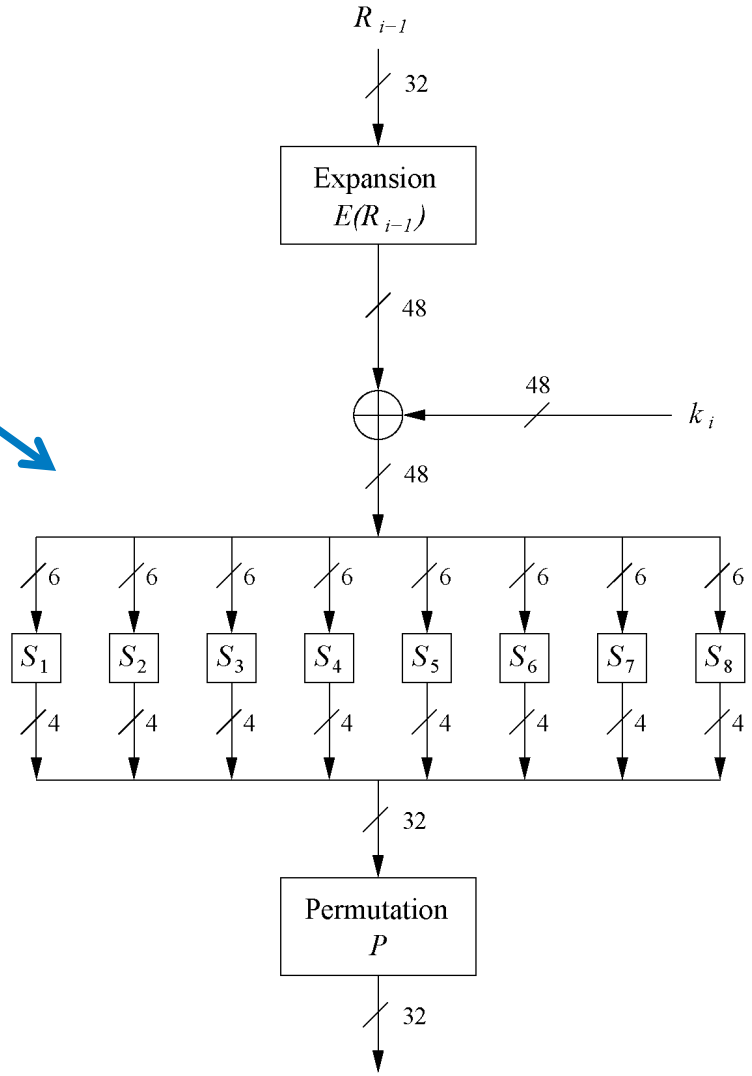
- Bitwise XOR of the round key and the output of the expansion function E
- Round keys are derived from the main key in the DES keyschedule (in a few slides)

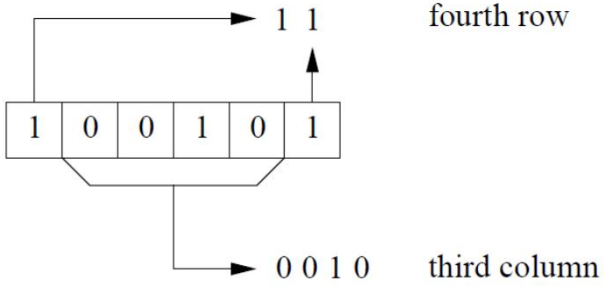


The DES S-Boxes

3. S-Box substitution

- Eight substitution tables.
- 6 bits of input, 4 bits of output.
- Non-linear and resistant to differential cryptanalysis.
- Crucial element for DES security!
- Find all S-Box tables and S-Box design criteria in *Understanding Cryptography* Chapter 3.





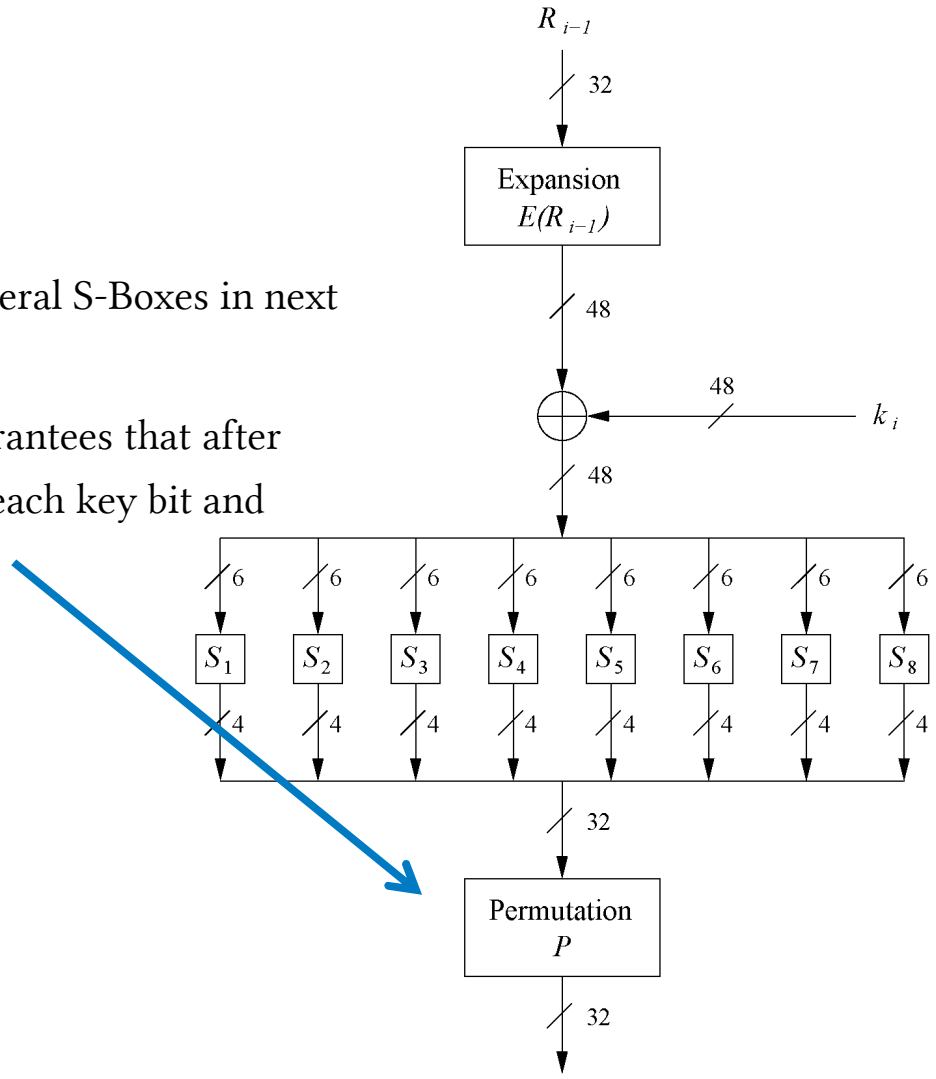
S_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

The Permutation P

4. Permutation P

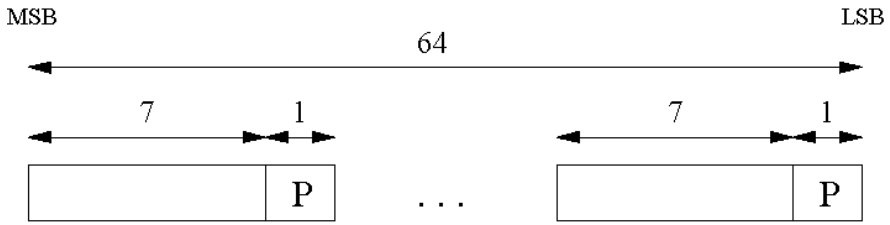
- Bitwise permutation.
- Introduces diffusion.
- Output bits of one S-Box effect several S-Boxes in next round
- Diffusion by E, S-Boxes and P guarantees that after Round 5 every bit is a function of each key bit and each plaintext bit.

P							
16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25



Key Schedule (1)

- Derives 16 round keys (or *subkeys*) k_i of 48 bits each from the original 56 bit key.
- The input key size of the DES is 64 bit: **56 bit key** and 8 bit parity:



P = parity bit

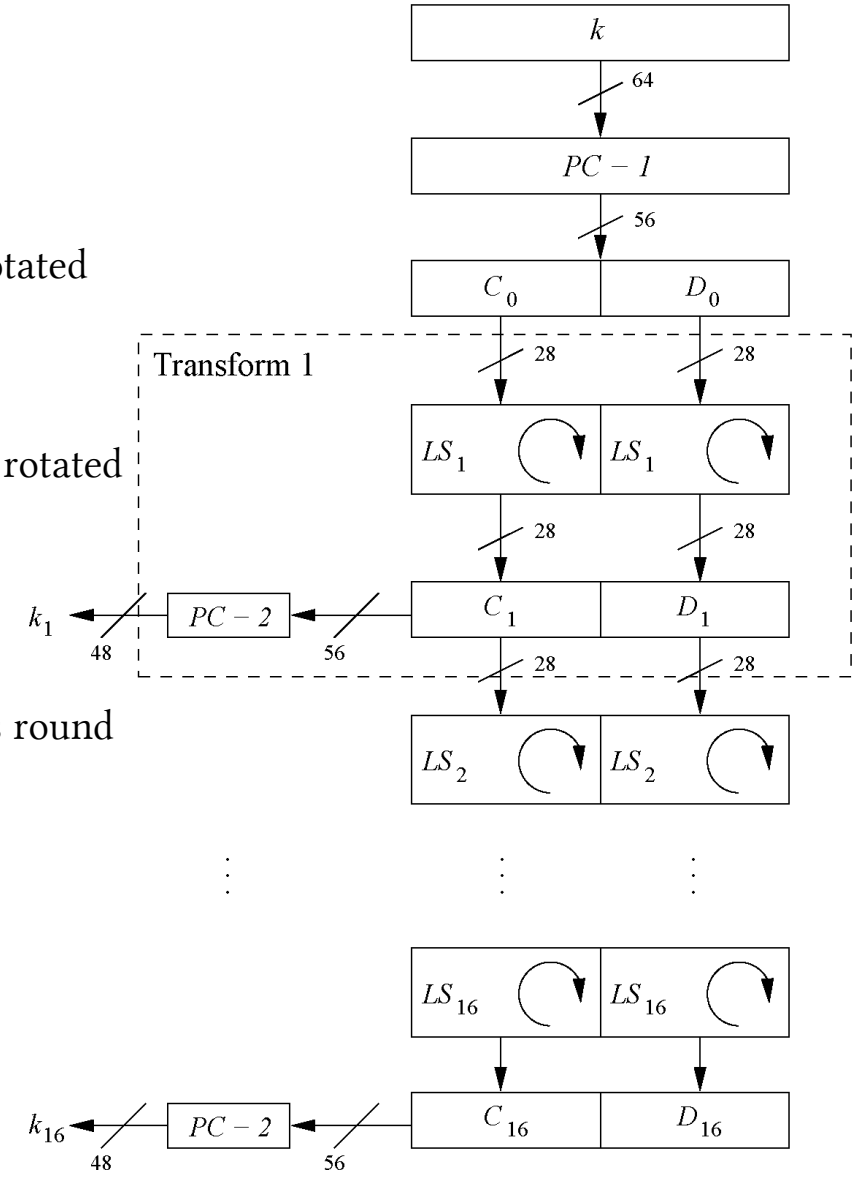
- **Parity bits are removed** in a first **permuted choice** *PC-1*:
(note that the bits 8, 16, 24, 32, 40, 48, 56 and 64 are not used at all)

PC – 1							
57	49	41	33	25	17	9	1
58	50	42	34	26	18	10	2
59	51	43	35	27	19	11	3
60	52	44	36	63	55	47	39
31	23	15	7	62	54	46	38
30	22	14	6	61	53	45	37
29	21	13	5	28	20	12	4

Key Schedule (2)

- Split key into 28-bit halves C_0 and D_0.
- In **rounds i = 1, 2, 9 ,16,** the two halves are each rotated left by **one bit**.
- In **all other rounds** where the two halves are each rotated left by **two bits**.
- *In each round i permuted choice* ***PC-2*** selects a permuted subset of 48 bits of C_i and D_i as round key k_i, i.e. **each k_i is a permutation of k!**

$PC-2$							
14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32



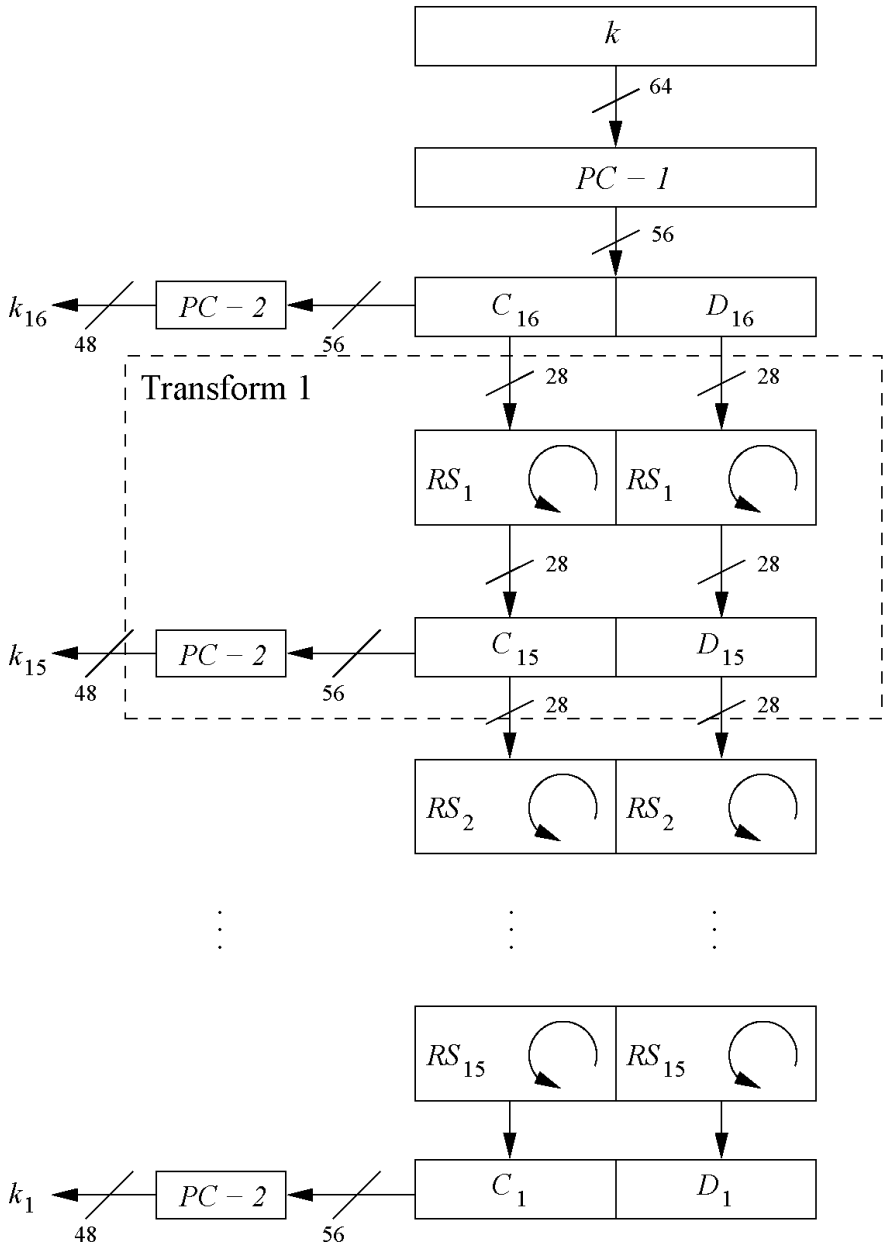
- **Note:** The total number of rotations:
  4 x 1 + 12 x 2 = 28 $\Rightarrow D_0 = D_{16}$ and $C_0 = C_{16}$!

Content of this Chapter

- Introduction to DES
- Overview of the DES Algorithm
- Internal Structure of DES
- **Decryption**
- Security of DES

Decryption

- In **Feistel ciphers** only the keyschedule has to be modified for decryption.
- Generate the same 16 round keys in reverse order.
  (for a detailed discussion on why this works see *Understanding Crptography* Chapter 3)
- **Reversed key schedule:**
  As D_0=D_{16} and C_0=C_{16} the first round key can be generated by applying *PC-2* right after *PC-1* (no rotation here!).
  All other rotations of C and D can be reversed to reproduce the other round keys resulting in:
  - No rotation in round 1.
  - One bit rotation **to the right** in rounds 2, 9 and 16.
  - Two bit rotations **to the right** in all other rounds.



Content of this Chapter

- Introduction to DES
- Overview of the DES Algorithm
- Internal Structure of DES
- Decryption
- **Security of DES**

Security of DES

- **After proposal of DES two major criticisms arose:**
  1. Key space is too small (2^{56} keys)
  2. S-box design criteria have been kept secret: Are there any hidden analytical attacks (*backdoors*), only known to the NSA?

- **Analytical Attacks:** DES is highly resistent to both *differential* and *linear cryptanalysis*, which have been published years later than the DES. This means IBM and NSA had been aware of these attacks for 15 years!
  So far there is no known analytical attack which breaks DES in realistic scenarios.

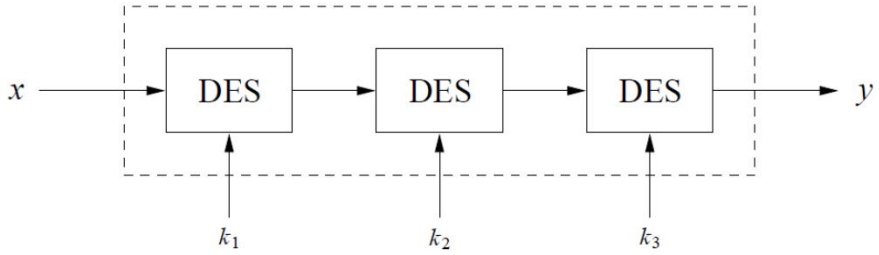
- **Exhaustive key search:** For a given pair of plaintext-ciphertext (x, y) test all 2^{56} keys until the condition $DES_k^{-1}(x)=y$ is fulfilled.
  $\Rightarrow$ Relatively easy given today's computer technology!

History of Attacks on DES

Triple DES – 3DES

- Triple encryption using DES is often used in practice to extend the effective key length of DES to 112. For more info on multiple encryption and effective key lengths see Chapter 5 of *Understanding Cryptography.*

$$y = DES_{k_3}(DES_{k_2}(DES_{k_1}(x)))$$



- Alternative version of *3DES:* $y = DES_{k_3}(DES^{-1}_{k_2}(DES_{k_1}(x)))$.

  Advantage: choosing k_1=k_2=k_3 performs single DES encryption.
- No practical attack known today.
- Used in many legacy applications, i.e., in banking systems.

Alternatives to DES

Algorithm	I/O Bit	key lengths	remarks
AES / Rijndael	128	128/192/256	DES "replacement", worldwide used standard
Triple DES	64	112 (effective)	conservative choice
Mars	128	128/192/256	AES finalist
RC6	128	128/192/256	AES finalist
Serpent	128	128/192/256	AES finalist
Twofish	128	128/192/256	AES finalist
IDEA	64	128	(Patented till 2011)

Lessons Learned

- DES was the dominant symmetric encryption algorithm from the mid-1970s to the mid-1990s. Since 56-bit keys are no longer secure, the Advanced Encryption Standard (AES) was created.
- Standard DES with 56-bit key length can be broken relatively easily nowadays through an exhaustive key search.
- DES is quite robust against known analytical attacks: In practice it is very difficult to break the cipher with differential or linear cryptanalysis.
- By encrypting with DES three times in a row, triple DES (3DES) is created, against which no practical attack is currently known.
- The "default" symmetric cipher is nowadays often AES. In addition, the other four AES finalist ciphers all seem very secure and efficient.

Content of this Chapter

- **Overview of the AES algorithm**
- Internal structure of AES
  - Byte Substitution layer
  - Diffusion layer
  - Key Addition layer
  - Key schedule
- Decryption
- Practical issues

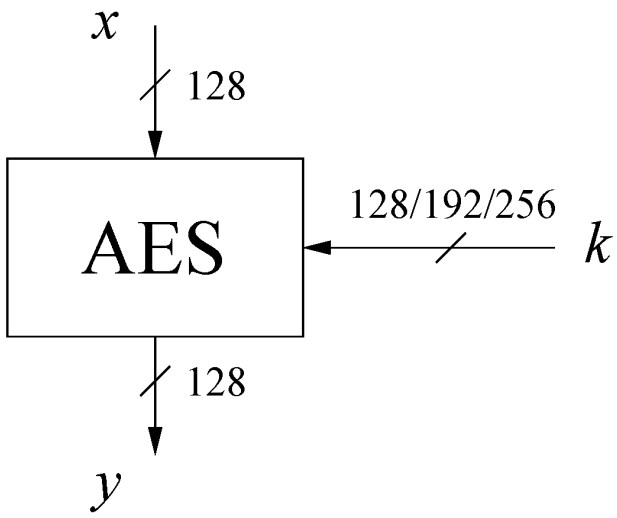
Some Basic Facts

- AES is the most widely used symmetric cipher today
- The algorithm for AES was chosen by the US *National Institute of Standards and Technology* (NIST) in a multi-year selection process
- The requirements for all AES candidate submissions were:
  - Block cipher with **128-bit block size**
  - **Three supported key lengths**: 128, 192 and 256 bit
  - Security relative to other submitted algorithms
  - **Efficiency** in software and hardware

Chronology of the AES Selection

- The need for a new block cipher announced by NIST in January, 1997
- 15 candidates algorithms accepted in August, 1998
- 5 finalists announced in August, 1999:
  - *Mars* – IBM Corporation
  - *RC6* – *RSA* Laboratories
  - *Rijndael* – J. Daemen & V. Rijmen
  - *Serpent* – Eli Biham et al.
  - *Twofish* – B. Schneier et al.
- In October 2000, *Rijndael* was chosen as the AES
- AES was formally approved as a US federal standard in November 2001

AES: Overview

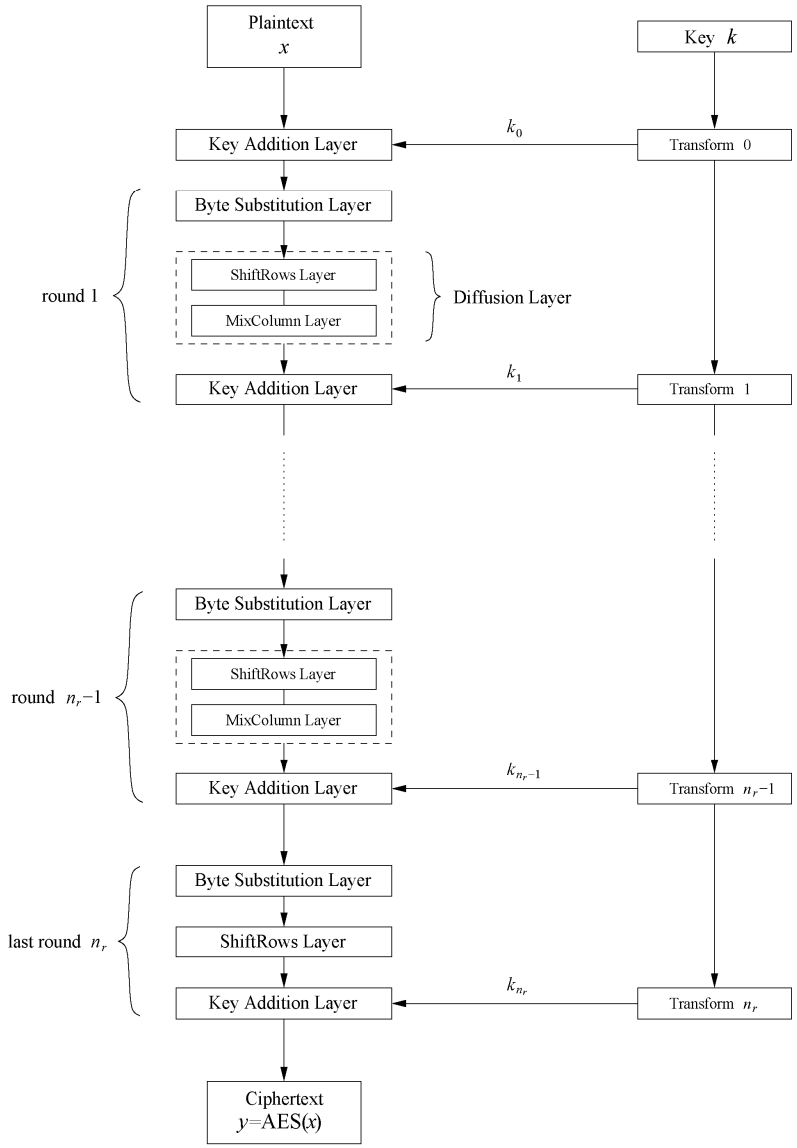


The number of rounds depends on the chosen key length:

Key length (bits)	Number of rounds
128	10
192	12
256	14

AES: Overview

- Iterated cipher with 10/12/14 rounds
- Each round consists of “Layers”



Content of this Chapter

- Overview of the AES algorithm
- **Internal structure of AES**
  - Byte Substitution layer
  - Diffusion layer
  - Key Addition layer
  - Key schedule
- Decryption
- Practical issues

Internal Structure of AES

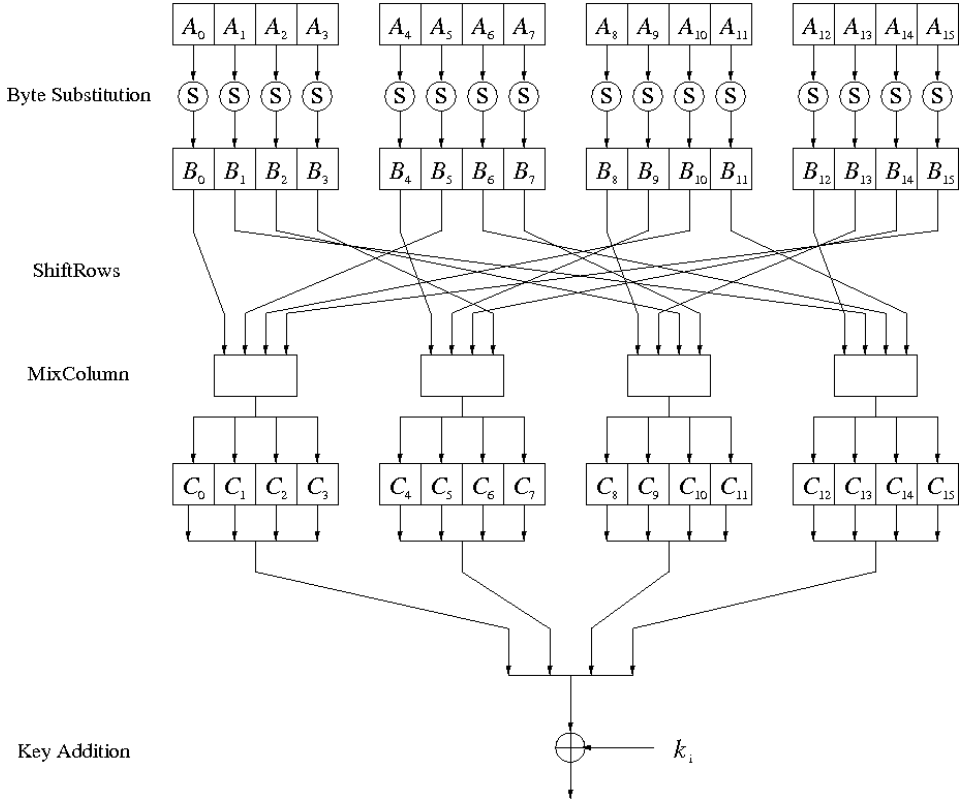
- AES is a byte-oriented cipher
- The state A (i.e., the 128-bit data path) can be arranged in a 4x4 matrix:

A_0	A_4	A_8	A_{12}
A_1	A_5	A_9	A_{13}
A_2	A_6	A_{10}	A_{14}
A_3	A_7	A_{11}	A_{15}

with A_0,..., A_{15} denoting the 16-byte input of AES

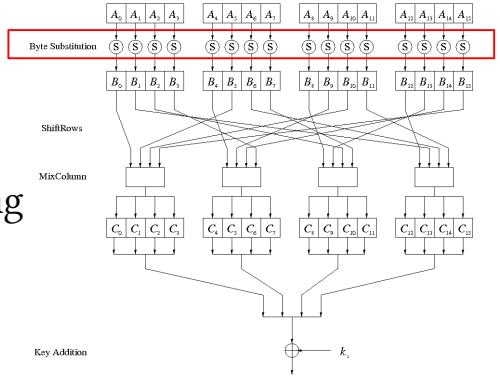
Internal Structure of AES

- Round function for rounds 1,2,…,n_{r-1}:



- Note: In the last round, the MixColumn tansformation is omitted

Byte Substitution Layer



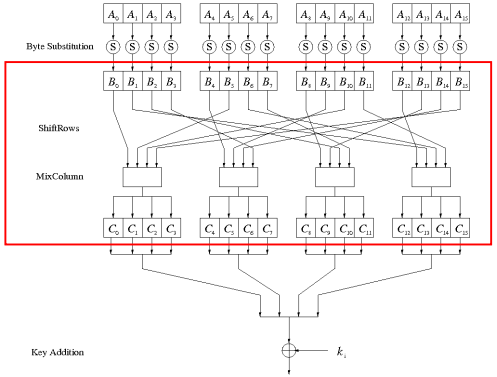
- The Byte Substitution layer consists of 16 **S-Boxes** with the following properties:

  The S-Boxes are

  - **identical**
  - the only **nonlinear** elements of AES, i.e.,
    ByteSub(A_i) + ByteSub(A_j) ≠ ByteSub(A_i + A_j), for i,j = 0,...,15
  - **bijective**, i.e., there exists a one-to-one mapping of input and output bytes
    ⇒ S-Box can be uniquely reversed

- In software implementations, the S-Box is usually realized as a lookup table

Diffusion Layer



The Diffusion layer

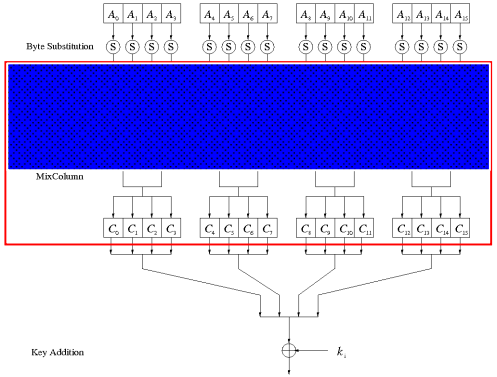
- provides diffusion over all input state bits

- consists of two sublayers:
  - **ShiftRows Sublayer**: Permutation of the data on a byte level
  - **MixColumn Sublayer**: Matrix operation which combines ("mixes") blocks of four bytes

- performs a linear operation on state matrices A, B, i.e.,

$$\text{DIFF}(A) + \text{DIFF}(B) = \text{DIFF}(A + B)$$

ShiftRows Sublayer



- Rows of the state matrix are shifted cyclically:

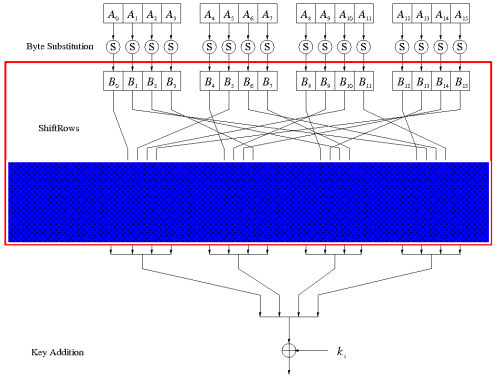
Input matrix

B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}

Output matrix

B_0	B_4	B_8	B_{12}	no shift
B_5	B_9	B_{13}	B_1	← one position left shift
B_{10}	B_{14}	B_2	B_6	← two positions left shift
B_{15}	B_3	B_7	B_{11}	← three positions left shift

MixColumn Sublayer



- Linear transformation which mixes each column of the state matrix

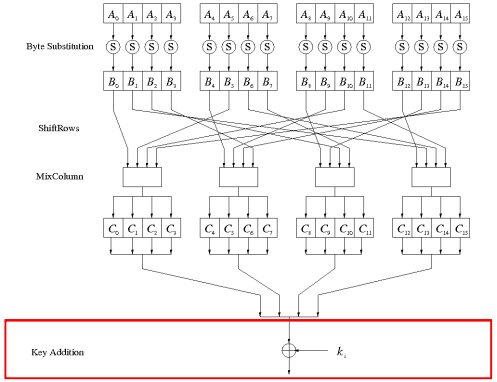
- Each 4-byte column is considered as a vector and multiplied by a fixed 4x4 matrix, e.g.,

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}$$

  where 01, 02 and 03 are given in hexadecimal notation

- All arithmetic is done in the Galois field $GF(2^8)$ (for more information see Chapter 4.3 in *Understanding Cryptography*)

Key Addition Layer



- Inputs:
  - 16-byte state matrix C
  - 16-byte subkey k_i
- Output: $C \oplus k_i$
- The subkeys are generated in the key schedule

Key Schedule

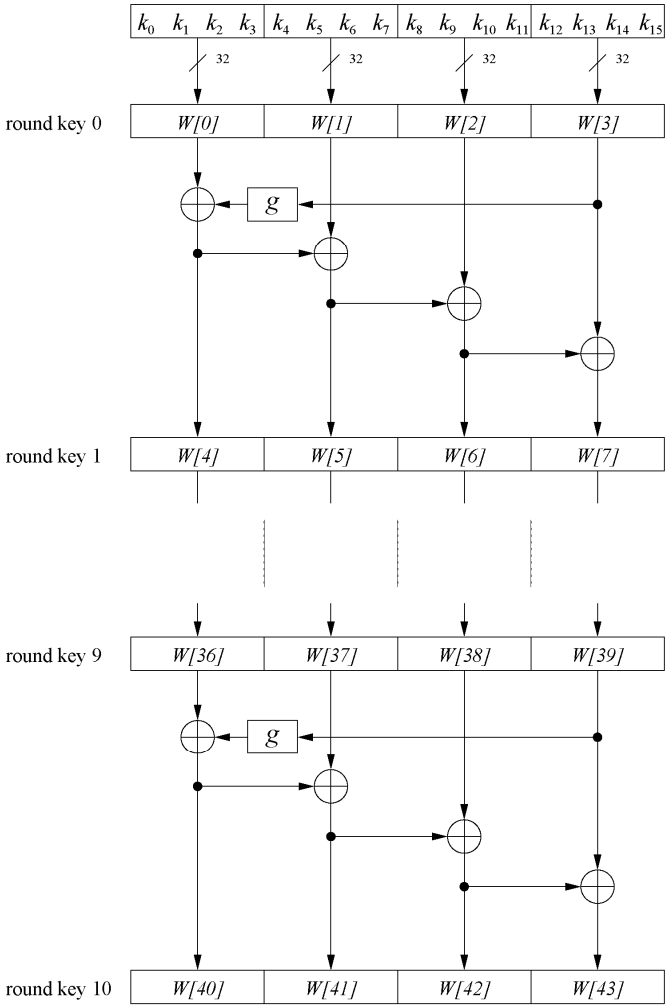
- Subkeys are derived recursively from the original 128/192/256-bit input key
- Each round has 1 subkey, plus 1 subkey at the beginning of AES

Key length (bits)	Number of subkeys
128	11
192	13
256	15

- Key whitening: Subkey is used both at the input and output of AES
  $\Rightarrow$ # subkeys = # rounds + 1
- There are different key schedules for the different key sizes

Key Schedule

Example: Key schedule for 128-bit key AES



- Word-oriented: 1 word = 32 bits
- 11 subkeys are stored in *W[0]…W[3], W[4]…W[7], … , W[40]…W[43]*
- First subkey *W[0]…W[3]* is the original AES key

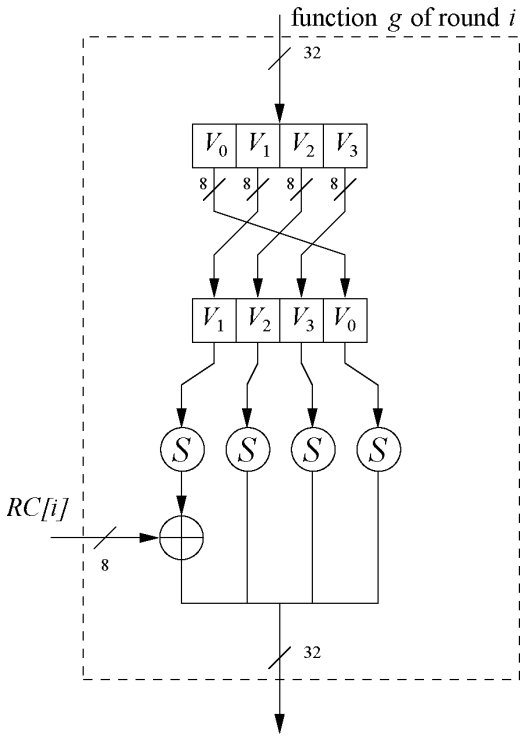
Key Schedule

- Function g rotates its four input bytes and performs a bytewise S-Box substitution
  $\Rightarrow$ nonlinearity

- The round coefficient RC is only added to the leftmost byte and varies from round to round:

$$RC[1] = x^0 = (00000001)_2$$
$$RC[2] = x^1 = (00000010)_2$$
$$RC[3] = x^2 = (00000100)_2$$
$$\ldots$$
$$RC[10] = x^9 = (00110110)_2$$

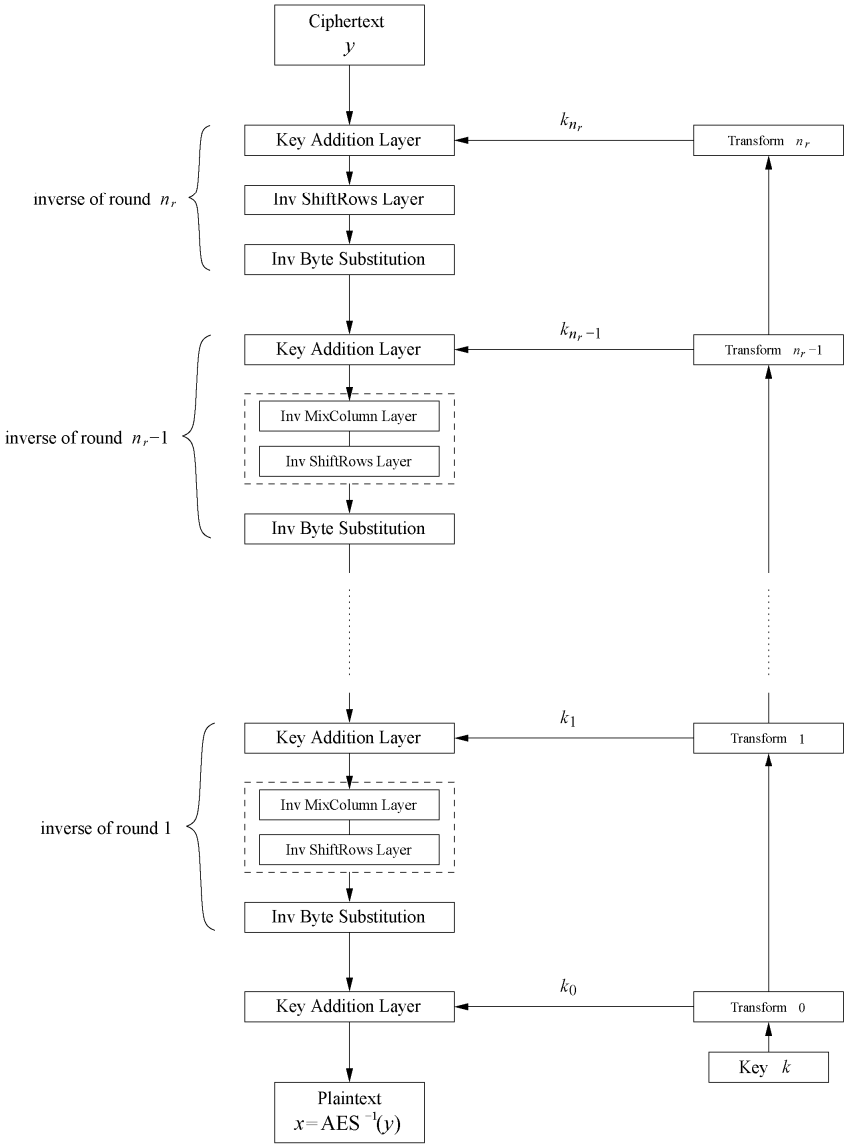
- x^i represents an element in a Galois field
  (again, cf. Chapter 4.3 of *Understanding Cryptography*)



Content of this Chapter

- Overview of the AES algorithm
- Internal structure of AES
  - Byte Substitution layer
  - Diffusion layer
  - Key Addition layer
  - Key schedule
- **Decryption**
- Practical issues

Decryption



- AES is not based on a Feistel network

$\Rightarrow$ All layers must be inverted for decryption:

- MixColumn layer $\longrightarrow$ **Inv MixColumn layer**
- ShiftRows layer$\longrightarrow$ **Inv ShiftRows layer**
- Byte Substitution layer $\longrightarrow$ **Inv Byte Substitution layer**
- Key Addition layer is its own inverse

Decryption

- **Inv MixColumn layer**:
  - To reverse the MixColumn operation, each column of the state matrix C must be multiplied with the **inverse of the 4x4 matrix**, e.g.,

$$\begin{pmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \cdot \begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix}$$

    where 09, $0B$, $0D$ and $0E$ are given in hexadecimal notation

- Again, all arithmetic is done in the Galois field $GF(2^8)$ (for more information see Chapter 4.3 in *Understanding Cryptography*)

Decryption

- **Inv ShiftRows layer**:
  - All rows of the state matrix B are shifted to the opposite direction:

Input matrix

B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}

Output matrix

B_0	B_4	B_8	B_{12}	no shift
B_{13}	B_1	B_5	B_9	→ one position right shift
B_{10}	B_{14}	B_2	B_6	→ two positions right shift
B_7	B_{11}	B_{15}	B_3	→ three positions right shift

Decryption

- **Inv Byte Substitution layer**:
  - Since the S-Box is bijective, it is possible to construct an inverse, such that

$$A_i = S^{-1}(B_i) = S^{-1}(S(A_i))$$

    $\Rightarrow$ The inverse S-Box is used for decryption. It is usually realized as a lookup table

- **Decryption key schedule**:
  - Subkeys are needed in reversed order (compared to encryption)
  - In practice, for encryption and decryption, the same key schedule is used. This requires that all subkeys must be computed before the encryption of the first block can begin

Content of this Chapter

- Overview of the AES algorithm
- Internal structure of AES
  - Byte Substitution layer
  - Diffusion layer
  - Key Addition layer
  - Key schedule
- Decryption
- **Practical issues**

Implementation in Software

- One requirement of AES was the possibility of an efficient software implementation
- Straightforward implementation is well suited for 8-bit processors (e.g., smart cards), but inefficient on 32-bit or 64-bit processors
- A more sophisticated approach: Merge all round functions (except the key addition) into one table look-up
  - This results in four tables with 256 entries, where each entry is 32 bits wide
  - One round can be computed with 16 table look-ups
- Typical SW speeds are more than 1.6 Gbit/s on modern 64-bit processors

Security

- **Brute-force attack:** Due to the key length of 128, 192 or 256 bits, a brute-force attack is not possible
- **Analytical attacks:** There is no analytical attack known that is better than brute-force
- **Side-channel attacks:**
  - Several side-channel attacks have been published
  - Note that side-channel attacks do not attack the underlying algorithm but the implementation of it

Lessons Learned

- AES is a modern block cipher which supports three key lengths of 128, 192 and 256 bit. It provides excellent long-term security against brute-force attacks.
- AES has been studied intensively since the late 1990s and no attacks have been found that are better than brute-force.
- AES is not based on Feistel networks. Its basic operations use Galois field arithmetic and provide strong diffusion and confusion.
- AES is part of numerous open standards such as IPsec or TLS, in addition to being the mandatory encryption algorithm for US government applications. It seems likely that the cipher will be the dominant encryption algorithm for many years to come.
- AES is efficient in software and hardware.

Content of this Chapter

- Encryption with Block Ciphers: Modes of Operation
  - Electronic Code Book mode (ECB)
  - Cipher Block Chaining mode (CBC)
  - Output Feedback mode (OFB)
  - Cipher Feedback mode (CFB)
  - Counter mode (CTR)
  - Galois Counter Mode (GCM)
- Exhaustive Key Search Revisited
- Increasing the Security of Block Ciphers

Block Ciphers

- A block cipher is much more than just an encryption algorithm, it can be used ...
  - to build different types of block-based encryption schemes
  - to realize stream ciphers
  - to construct hash functions
  - to make message authentication codes
  - to build key establishment protocols
  - to make a pseudo-random number generator
  - ...

- The security of block ciphers also can be increased by
  - key whitening
  - multiple encryption

Content of this Chapter

- **Encryption with Block Ciphers: Modes of Operation**
  - Electronic Code Book mode (ECB)
  - Cipher Block Chaining mode (CBC)
  - Output Feedback mode (OFB)
  - Cipher Feedback mode (CFB)
  - Counter mode (CTR)
  - Galois Counter Mode (GCM)
- Exhaustive Key Search Revisited
- Increasing the Security of Block Ciphers

Encryption with Block Ciphers

- There are several ways of encrypting long plaintexts, e.g., an e-mail or a computer file, with a block cipher ("modes of operation")
  - Electronic Code Book mode (ECB)
  - Cipher Block Chaining mode (CBC)
  - Output Feedback mode (OFB)
  - Cipher Feedback mode (CFB)
  - Counter mode (CTR)
  - Galois Counter Mode (GCM)

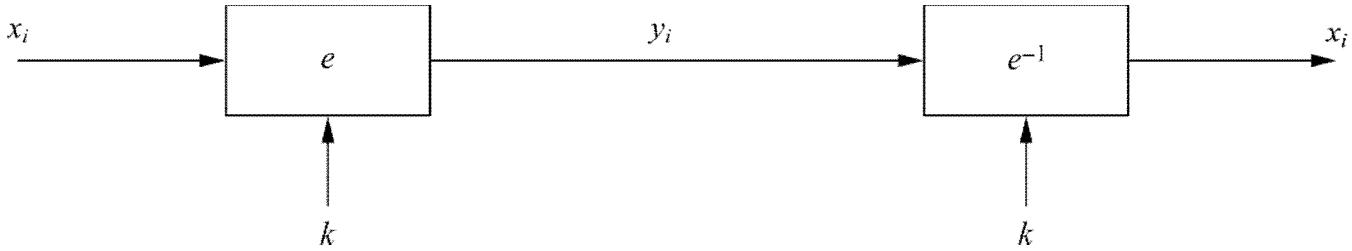
- All of the 6 modes have one goal:
  - In addition to confidentiality, they provide authenticity and integrity:
    - Is the message really coming from the original sender? (authenticity)
    - Was the ciphertext altered during transmission? (integrity)

Content of this Chapter

- Encryption with Block Ciphers: Modes of Operation
  - Electronic Code Book mode (ECB)
  - Cipher Block Chaining mode (CBC)
  - Output Feedback mode (OFB)
  - Cipher Feedback mode (CFB)
  - Counter mode (CTR)
  - Galois Counter Mode (GCM)
- Exhaustive Key Search Revisited
- Increasing the Security of Block Ciphers

Electronic Code Book mode (ECB)

- $e_k(x_i)$ denote the encryption of a b-bit plaintext block x_i with key k
- $e_k^{-1}(y_i)$ denote the decryption of b-bit ciphertext block y_i with key k
- Messages which exceed b bits are partitioned into b-bit blocks
- Each Block is encrypted separately



Encryption: $y_i = e_k(x_i), \quad i \geq 1$

Decryption: $x_i = e_k^{-1}(y_i) = e_k^{-1}(e_k(x_i)), \quad i \geq 1$

ECB: advantages/disadvantages

- Advantages
  - no block synchronization between sender and receiver is required
  - bit errors caused by noisy channels only affect the corresponding block but not succeeding blocks
  - Block cipher operating can be parallelized
    - advantage for high-speed implementations

- Disadvantages
  - ECB encrypts highly deterministically
    - identical plaintexts result in identical ciphertexts
    - an attacker recognizes if the same message has been sent twice
    - plaintext blocks are encrypted independently of previous blocks
      - an attacker may reorder ciphertext blocks which results in valid plaintext

Substitution Attack on ECB

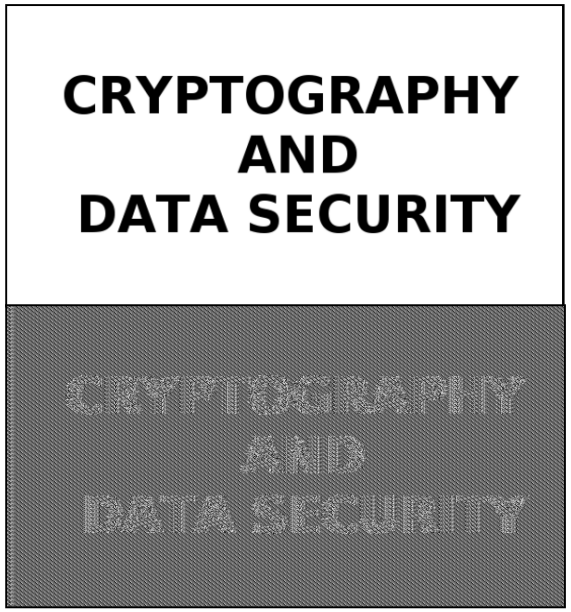
- Once a particular plaintext to ciphertext block mapping $x_i \longrightarrow y_i$ is known, a sequence of ciphertext blocks can easily be manipulated
- Suppose an *electronic bank transfer*

Block #	1	2	3	4	5
	Sending Bank A	Sending Account #	Receiving Bank B	Receiving Account #	Amount $

  - the encryption key between the two banks does not change too frequently
  - The attacker sends $1.00 transfers from his account at bank A to his account at bank B repeatedly
    - He can check for ciphertext blocks that repeat, and he stores blocks 1,3 and 4 of these transfers
  - He now simply replaces block 4 of other transfers with the block 4 that he stored before
    - *all transfers* from some account of bank A to some account of bank B are redirected to go into the attacker's B account!

Example of encrypting bitmaps in ECB mode

- Identical plaintexts are mapped to identical ciphertexts



- Statistical properties in the plaintext are preserved in the ciphertext

Content of this Chapter

- **Encryption with Block Ciphers: Modes of Operation**
  - Electronic Code Book mode (ECB)
  - **Cipher Block Chaining mode (CBC)**
  - Output Feedback mode (OFB)
  - Cipher Feedback mode (CFB)
  - Counter mode (CTR)
  - Galois Counter Mode (GCM)
- Exhaustive Key Search Revisited
- Increasing the Security of Block Ciphers

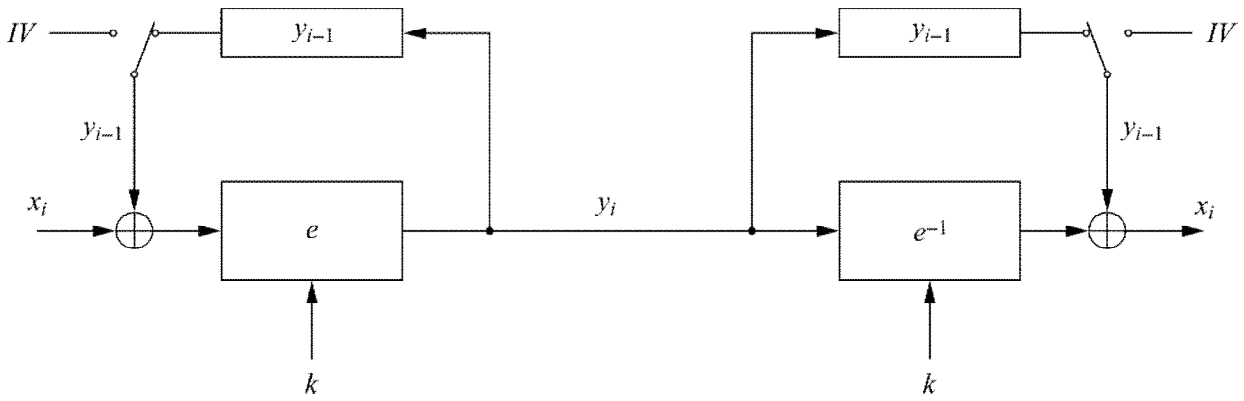
Cipher Block Chaining mode (CBC)

- There are two main ideas behind the CBC mode:
  - The encryption of all blocks are "chained together"
    - ciphertext y_i depends not only on block x_i but on all previous plaintext blocks as well
  - The encryption is randomized by using an initialization vector (IV)

Encryption (first block):	$y_1 = e_k(x_1 \oplus \text{IV})$
Encryption (general block):	$y_i = e_k(x_i \oplus y_{i-1}), \quad i \geq 2$
Decryption (first block):	$x_1 = e_k^{-1}(y_1) \oplus \text{IV}$
Decryption (general block) :	$x_i = e_k^{-1}(y_i) \oplus y_{i-1}, \quad i \geq 2$

Cipher Block Chaining mode (CBC)

- For the first plaintext block x_1 there is no previous ciphertext
  - an IV is added to the first plaintext to make each CBC encryption nondeterministic
  - the first ciphertext y_1 depends on plaintext x_1 and the IV
- The second ciphertext y_2 depends on the IV, x_1 *and* x_2
- The third ciphertext y_3 depends on the IV and x_1, x_2 *and* x_3, and so on



Substitution Attack on CBC

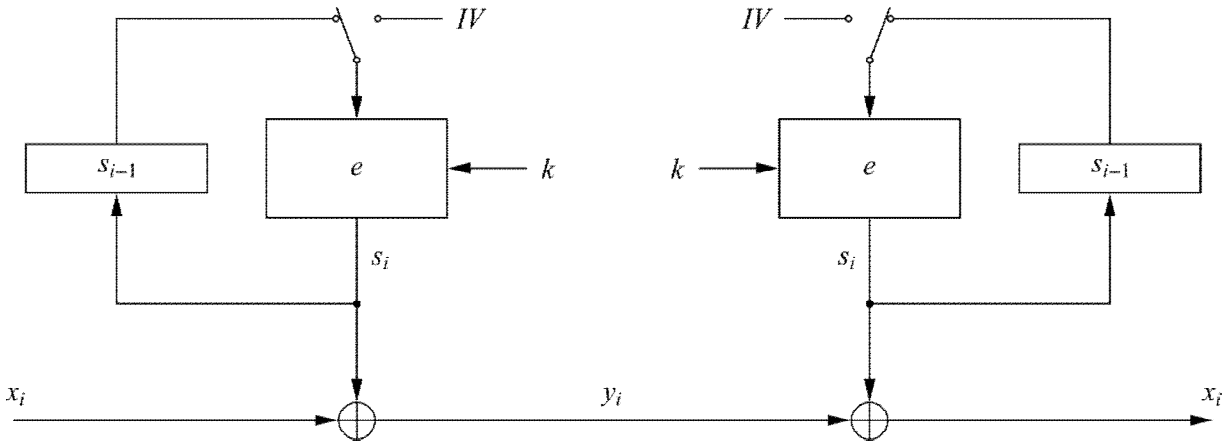
- Suppose the last example (*electronic bank transfer*)
- If the IV is properly chosen for every wire transfer, the attack will not work at all
- If the IV is kept the same for several transfers, the attacker would recognize the transfers from his account at bank A to back B
- If we choose a new IV every time we encrypt, the CBC mode becomes a probabilistic encryption scheme, i.e., two encryptions of the same plaintext look entirely different
- It is not needed to keep the IV *secret*!
- Typically, the IV should be a non-secret nonce (value used only once)

Content of this Chapter

- **Encryption with Block Ciphers: Modes of Operation**
  - Electronic Code Book mode (ECB)
  - Cipher Block Chaining mode (CBC)
  - **Output Feedback mode (OFB)**
  - Cipher Feedback mode (CFB)
  - Counter mode (CTR)
  - Galois Counter Mode (GCM)
- Exhaustive Key Search Revisited
- Increasing the Security of Block Ciphers

Output Feedback mode (OFB)

- It is used to build a *synchronous* **stream cipher** from a block cipher
- The key stream is not generated bitwise but instead in a blockwise fashion
- The output of the cipher gives us key stream bits S_i with which we can encrypt plaintext bits using the XOR operation



Encryption (first block): $s_1 = e_k(\mathrm{IV})$ *and* $y_1 = s_1 \oplus x_1$

Encryption (general block): $s_i = e_k(s_{i-1})$ *and* $y_i = s_i \oplus x_i$, $i \geq 2$

Decryption (first block): $s_1 = e_k(\mathrm{IV})$ *and* $x_1 = s_1 \oplus y_1$

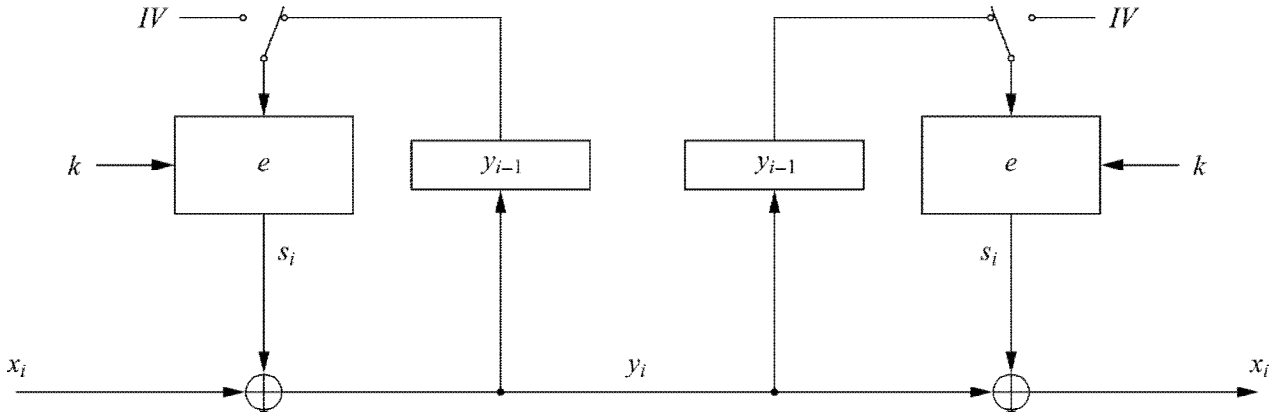
Decryption (general block) : $s_i = e_k(s_{i-1})$ *and* $x_i = s_i \oplus y_i$, $i \geq 2$

Content of this Chapter

- **Encryption with Block Ciphers: Modes of Operation**
  - Electronic Code Book mode (ECB)
  - Cipher Block Chaining mode (CBC)
  - Output Feedback mode (OFB)
  - **Cipher Feedback mode (CFB)**
  - Counter mode (CTR)
  - Galois Counter Mode (GCM)
- Exhaustive Key Search Revisited
- Increasing the Security of Block Ciphers

Cipher Feedback mode (CFB)

- It uses a block cipher as a building block for an asynchronous **stream cipher** (similar to the OFB mode), more accurate name: "Ciphertext Feedback Mode"
- The key stream S_i is generated in a blockwise fashion and is also a function of the ciphertext
- As a result of the use of an IV, the CFB encryption is also nondeterministic



Encryption (first block): $y_1 = e_k(\text{IV}) \oplus x_1$
Encryption (general block): $y_i = e_k(y_{i-1}) \oplus x_i, \quad i \geq 2$
Decryption (first block): $x_1 = e_k(\text{IV}) \oplus y_1$
Decryption (general block) : $x_i = e_k(y_{i-1}) \oplus y_i, \quad i \geq 2$

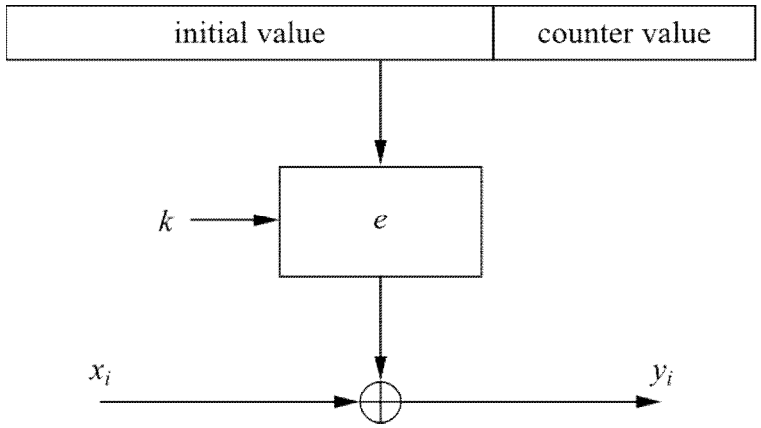
- It can be used in situations where short plaintext blocks are to be encrypted

Content of this Chapter

- **Encryption with Block Ciphers: Modes of Operation**
  - Electronic Code Book mode (ECB)
  - Cipher Block Chaining mode (CBC)
  - Output Feedback mode (OFB)
  - Cipher Feedback mode (CFB)
  - **Counter mode (CTR)**
  - Galois Counter Mode (GCM)
- Exhaustive Key Search Revisited
- Increasing the Security of Block Ciphers

Counter mode (CTR)

- It uses a block cipher as a **stream cipher** (like the OFB and CFB modes)
- The key stream is computed in a blockwise fashion
- The input to the block cipher is a counter which assumes a different value every time the block cipher computes a new key stream block



- Unlike CFB and OFB modes, the CTR mode can be parallelized since the 2nd encryption can begin before the 1st one has finished
  - Desirable for high-speed implementations, e.g., in network routers

$$\textbf{\textit{Encryption}}: \quad y_i = e_k(\text{IV} \,||\, \text{CTR}_i) \oplus x_i, \quad i \geq 1$$
$$\textbf{\textit{Decryption}}: \quad x_i = e_k(\text{IV} \,||\, \text{CTR}_i) \oplus y_i, \quad i \geq 1$$

Content of this Chapter

- **Encryption with Block Ciphers: Modes of Operation**
  - Electronic Code Book mode (ECB)
  - Cipher Block Chaining mode (CBC)
  - Output Feedback mode (OFB)
  - Cipher Feedback mode (CFB)
  - Counter mode (CTR)
  - **Galois Counter Mode (GCM)**
- Exhaustive Key Search Revisited
- Increasing the Security of Block Ciphers

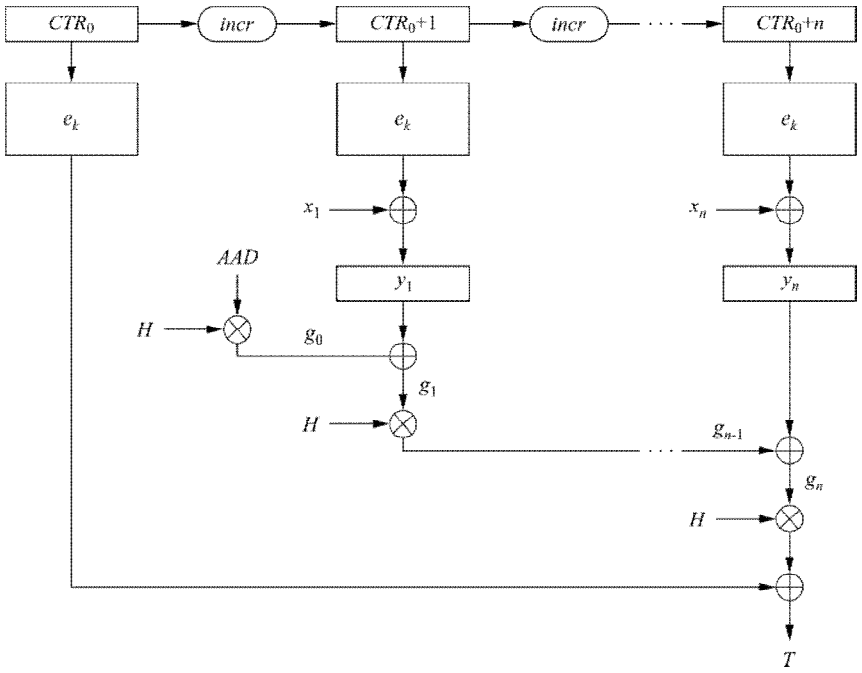
Galois Counter Mode (GCM)

- It also computes a *message authentication code* (MAC), i.e., a cryptographic checksum is computed for a message (for more information see Chapter 12 in *Understanding Cryptography*)
- By making use of GCM, two additional services are provided:
  - Message Authentication
    - the receiver can make sure that the message was really created by the original sender
  - Message Integrity
    - the receiver can make sure that nobody tampered with the ciphertext during transmission

Galois Counter Mode (GCM)

- For encryption
  - An initial counter is derived from an IV and a serial number
  - The initial counter value is incremented then encrypted and XORed with the first plaintext block
  - For subsequent plaintexts, the counter is incremented and then encrypted
- For authentication
  - A chained Galois field multiplication is performed (for more information Galois field see Chapter 4.3 in *Understanding Cryptography*)
  - For every plaintext an intermediate authentication parameter g_i is derived
    - g_i is computed as the XOR of the current ciphertext and the last g_{i-1}, and multiplied by the constant H
      - H is generated by encryption of the zero input with the block cipher
  - All multiplications are in the 128-bit Galois field $\mathrm{GF}(2^{128})$

Galois Counter Mode (GCM)



Encryption:

a. Derive a counter value CTR_0 from the IV and compute $\mathrm{CTR}_1 = \mathrm{CTR}_0 + 1$

b. Compute ciphertext: $y_i = e_k(\mathrm{CTR}_i) \oplus x_i, \quad i \geq 1$

Authentication:

a. Generate authentication subkey $H = e_k(0)$

b. Compute $g_0 = AAD \times H$ (Galois field multiplication)

c. Compute $g_i = (g_{i-1} \oplus y_i) \times H, \; 1 \leq i \leq n$ (Galois field multiplication)

d. Final authentication tag: $T = (g_n \times H) \oplus e_k(\mathrm{CTR}_0)$

Content of this Chapter

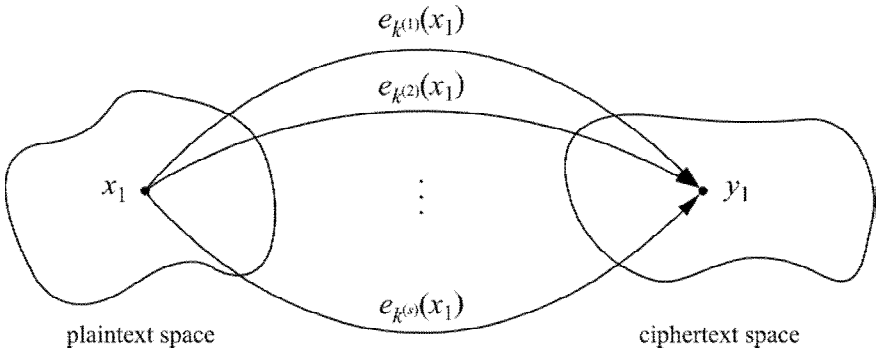
- Encryption with Block Ciphers: Modes of Operation
- **Exhaustive Key Search Revisited**
- Increasing the Security of Block Ciphers

Exhaustive Key Search Revisited

- A simple exhaustive search for a DES key knowing one pair (x_1, y_1):

$$DES_k^{(i)}(x_1) \stackrel{?}{=} y_1, \quad i = 0,1,\ldots,2^{56}-1$$

- However, for most other block ciphers a key search is somewhat more complicated
- A brute-force attack can produce *false positive* results
  - keys k_i that are found are not the one used for the encryption



  - The likelihood of this is related to the relative size of the key space and the plaintext space
  - A brute-force attack is still *possible*, but several pairs of plaintext–ciphertext are needed

An Exhaustive Key Search Example

- Assume a cipher with a block width of 64 bit and a key size of 80 bit
- If we encrypt x_1 under all possible 2^{80} keys, we obtain 2^{80} ciphertexts
  - However, there exist only 2^{64} different ones
- If we run through all keys for a given plaintext–ciphertext pair, we find on average $2^{80}/2^{64} = 2^{16}$ keys that perform the mapping $e_k(x_1) = y_1$

> Given a block cipher with a key length of k bits and block size of n bits, as well as t plaintext–ciphertext pairs $(x_1, y_1), \ldots, (x_t, y_t)$, the expected number of *false* keys which encrypt all plaintexts to the corresponding ciphertexts is:
>
> $$2^{k-tn}$$

- In this example assuming two plaintext-ciphertext pairs, the likelihood is

$$2^{80-2\cdot 64} = 2^{-48}$$

  - for almost all practical purposes two plaintext-ciphertext pairs are sufficient

Content of this Chapter

- Encryption with Block Ciphers: Modes of Operation
- Exhaustive Key Search Revisited
- **Increasing the Security of Block Ciphers**
  - Double Encryption and Meet-in-the-Middle Attack
  - Triple Encryption
  - Key Whitening

Increasing the Security of Block Ciphers

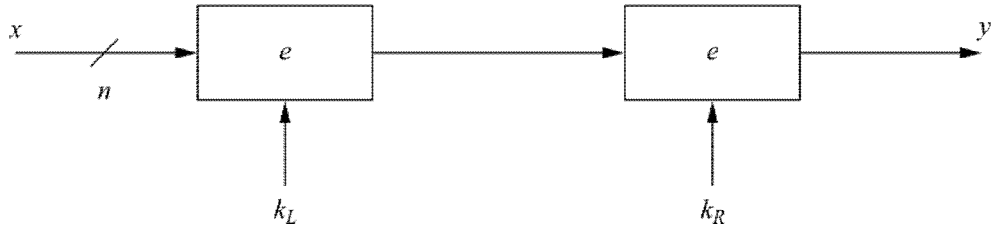
- In some situations we wish to increase the security of block ciphers, e.g., if a cipher such as DES is available in hardware or software for legacy reasons in a given application
- Two approaches are possible
  - Multiple encryption
    - theoretically much more secure, but **sometimes** in practice increases the security very little
  - Key whitening

Content of this Chapter

- Encryption with Block Ciphers: Modes of Operation
- Exhaustive Key Search Revisited
- **Increasing the Security of Block Ciphers**
  - **Double Encryption and Meet-in-the-Middle Attack**
  - Triple Encryption
  - Key Whitening

Double Encryption

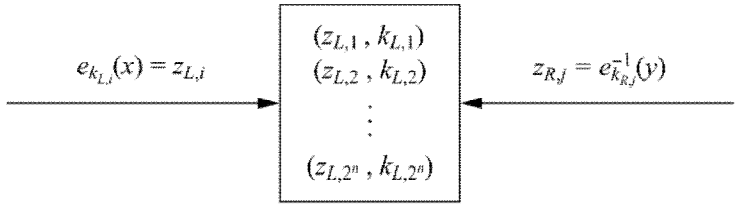
- A plaintext x is first encrypted with a key k_L, and the resulting ciphertext is encrypted again using a second key k_R



- Assuming a key length of k bits, an exhaustive key search would require $2^k \cdot 2^k = 2^{2k}$ encryptions or decryptions

Meet-in-the-Middle Attack

- A Meet-in-the-Middle attack requires $2^k+2^k = 2^{k+1}$ operations!



  - **Phase I**: for the given (x_1, y_1) the **left** encryption is brute-forced for all $k_{L,i}$, i=1,2, ..., 2^k and a lookup table with 2^k entry (each $n+k$ bits wide) is computed
    - the lookup table should be ordered by the result of the encryption ($z_{L,i}$)
  - **Phase II**: the **right** encryption is brute-forced (using decryption) and for each $z_{R,i}$ it is checked whether $z_{R,i}$ is equal to any $z_{L,i}$ value in the table of the first phase

- Computational Complexity

number of encryptions and decryptions = $2^k + 2^k = 2^{k+1}$
number of storage locations = 2^k

- **Double encryption is not much more secure then single encryption!**

Content of this Chapter

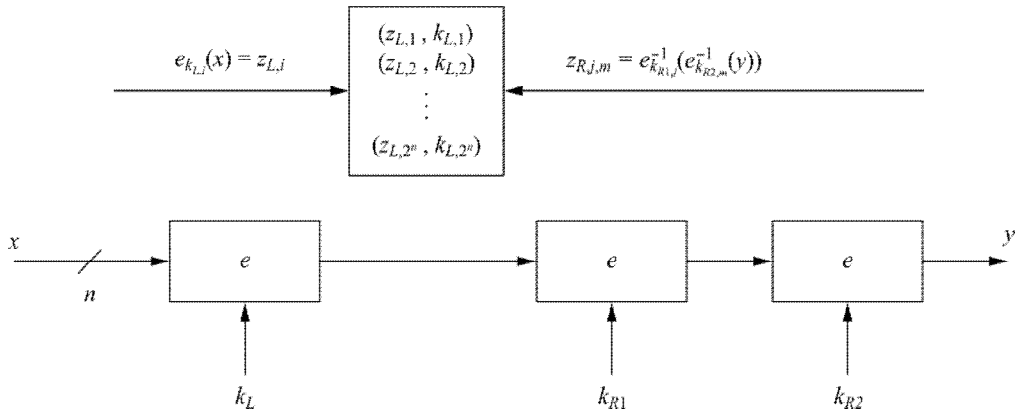
- Encryption with Block Ciphers: Modes of Operation
- Exhaustive Key Search Revisited
- **Increasing the Security of Block Ciphers**
  - Double Encryption and Meet-in-the-Middle Attack
  - **Triple Encryption**
  - Key Whitening

Triple Encryption

- The encryption of a block three times $y = e_{k3}(e_{k2}(e_{k1}(x)))$
- In practice a variant scheme is often used EDE (encryption-decryption-encryption)

$$y = e_{k3}(e^{-1}_{k2}(e_{k1}(x)))$$

  - Advantage: choosing $k1=k2=k3$ performs single DES encryption
- Still we can perform a meet-in-the middle attack, and it reduces the *effective key length* of triple encryption from $3K$ to $2K$!
  - The attacker must run 2^{112} tests in the case of 3DES



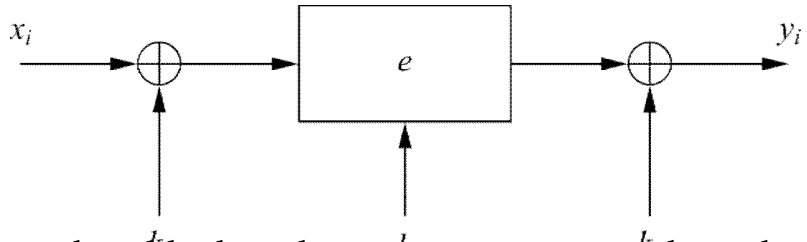
- **Triple encryption effectively doubles the key length**

Content of this Chapter

- Encryption with Block Ciphers: Modes of Operation
- Exhaustive Key Search Revisited
- **Increasing the Security of Block Ciphers**
  - Double Encryption and Meet-in-the-Middle Attack
  - Triple Encryption
  - **Key Whitening**

Key Whitening

- Makes block ciphers such as DES much more resistant against brute-force attacks
- In addition to the regular cipher key k, two whitening keys k_1 and k_2 are used to XOR-mask the plaintext and ciphertext



- It does not strengthen block ciphers against most analytical attacks such as linear and differential cryptanalysis
- It is not a "cure" for inherently weak ciphers
- The additional computational load is negligible
- Its main application is ciphers that are relatively strong against analytical attacks but possess too short a key space especially DES
  - a variant of DES which uses key whitening is called DESX

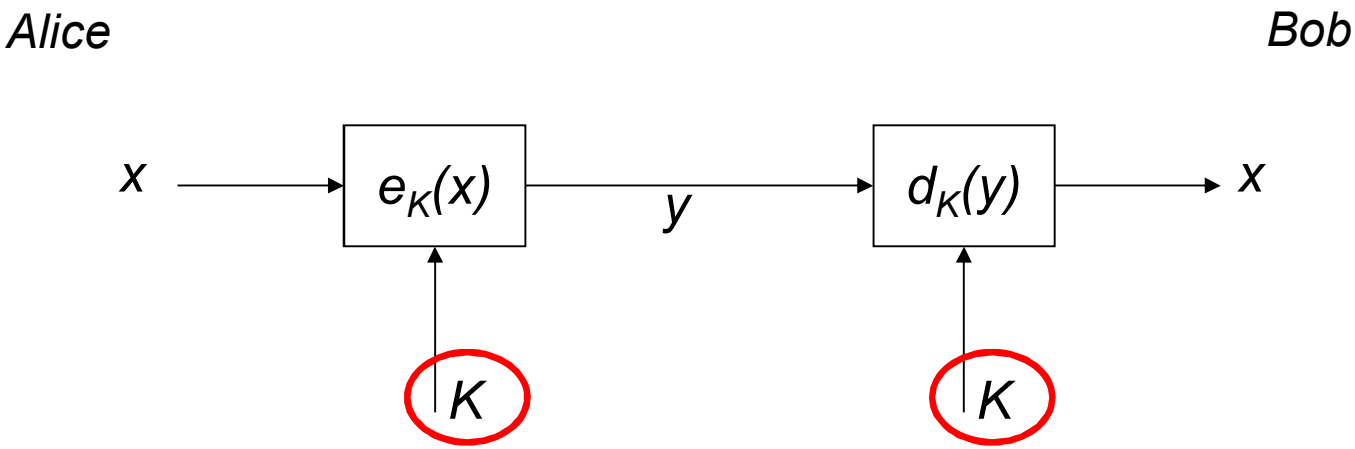
Lessons Learned

- There are many different ways to encrypt with a block cipher. Each mode of operation has some advantages and disadvantages
- Several modes turn a block cipher into a stream cipher
- There are modes that perform encryption together together with authentication, i.e., a cryptographic checksum protects against message manipulation
- The straightforward ECB mode has security weaknesses, independent of the underlying block cipher
- The counter mode allows parallelization of encryption and is thus suited for high speed implementations
- Double encryption with a given block cipher only marginally improves the resistance against brute-force attacks
- Triple encryption with a given block cipher roughly *doubles* the key length
- Triple DES (3DES) has an effective key length of 112 bits
- Key whitening enlarges the DES key length without much computational overhead.

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**:

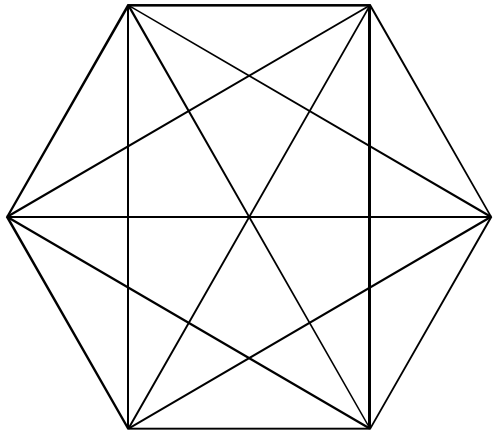
- Key distribution problem: The secret key must be **transported securely**
- Number of keys: In a network, each pair of users requires an individual key

→ *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:

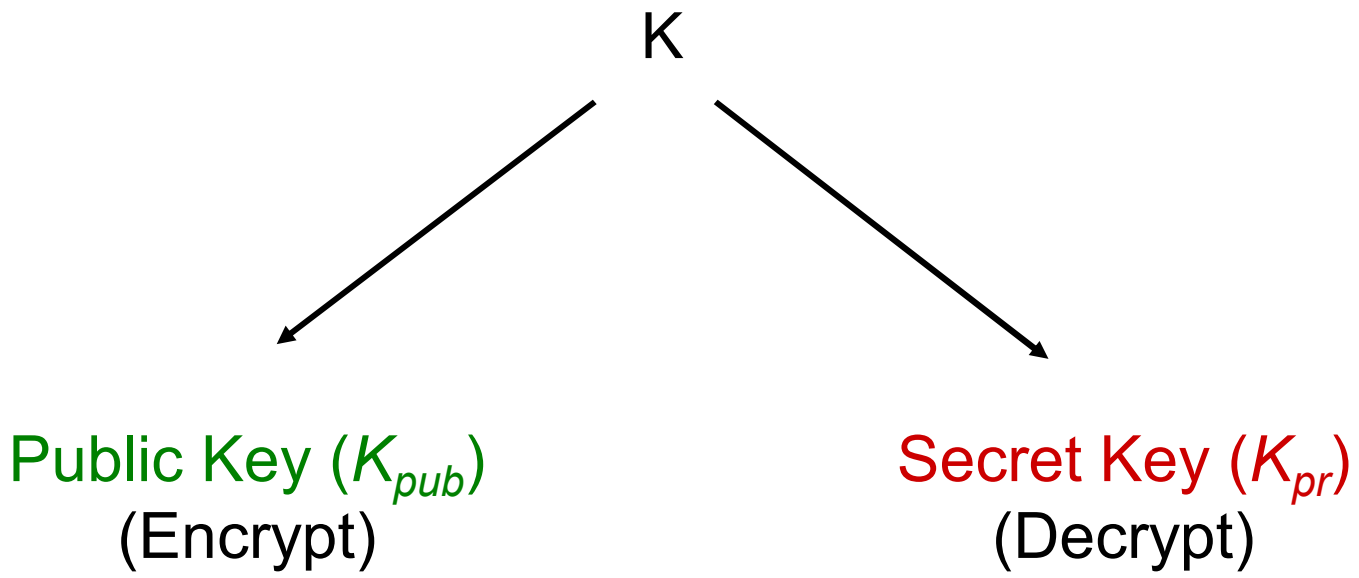
Everyone can drop a letter

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



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

Asymmetric Cryptography: Analogy

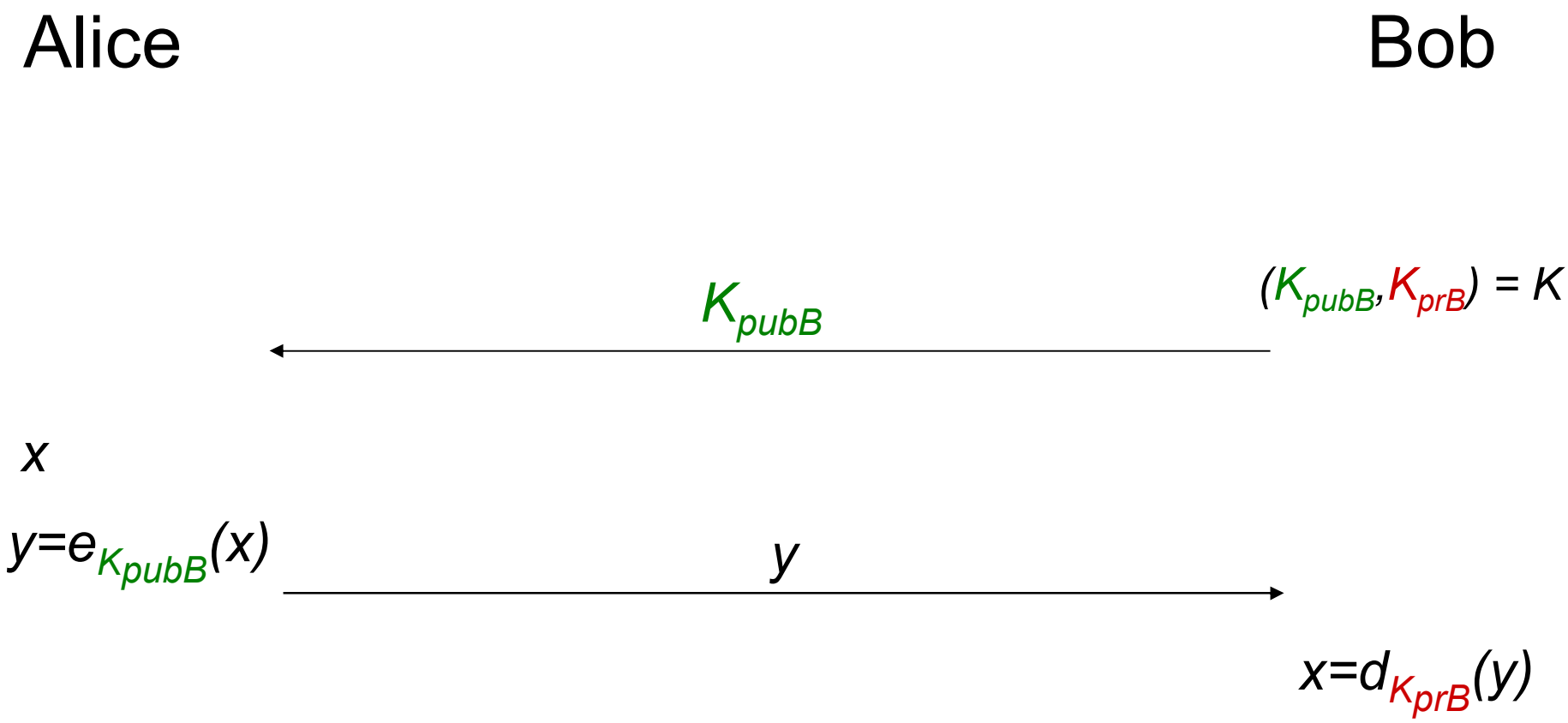
Safe with public lock and private lock:

- 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



→ 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

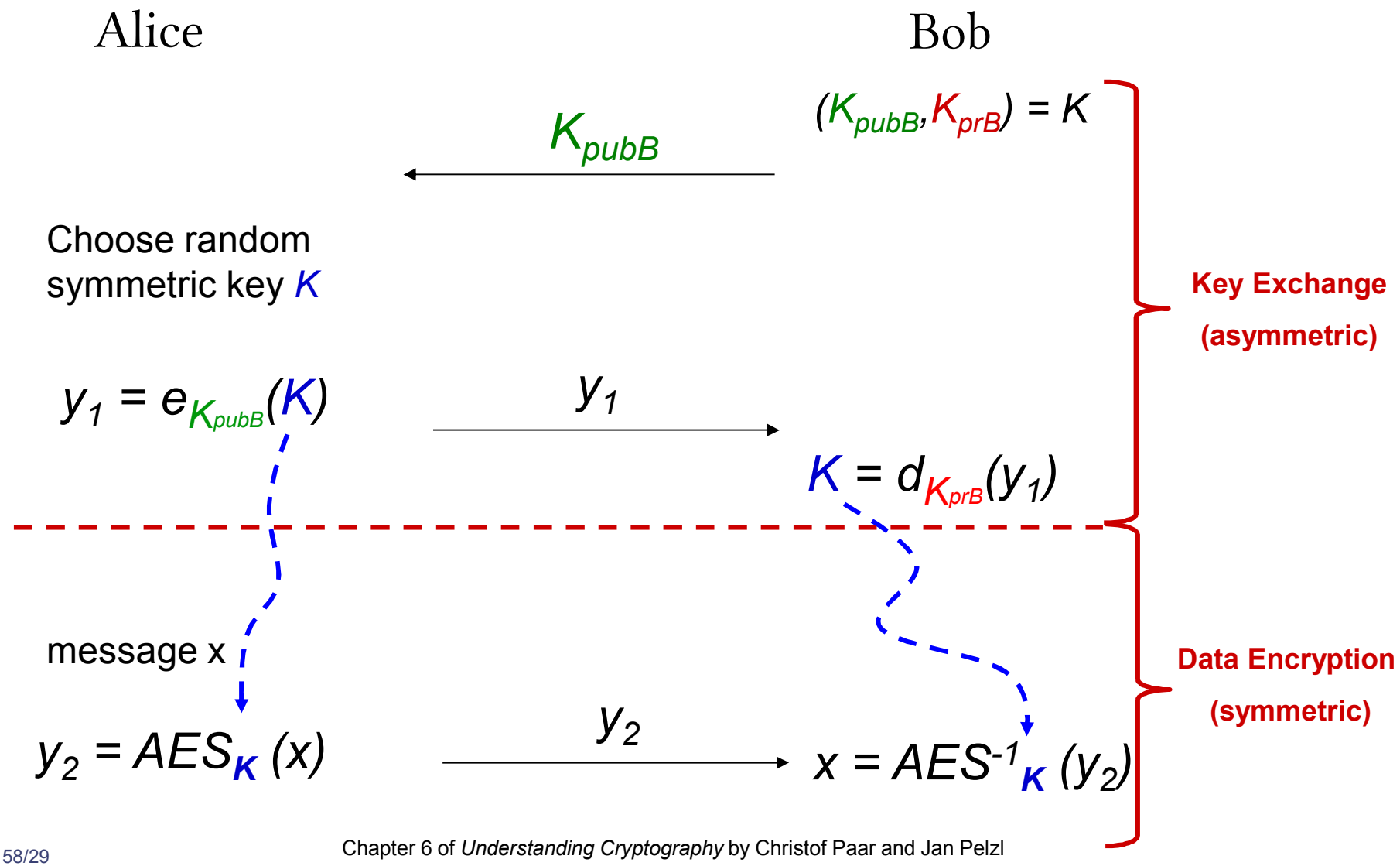
In practice: **Hybrid systems**, incorporating asymmetric and symmetric algorithms

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



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

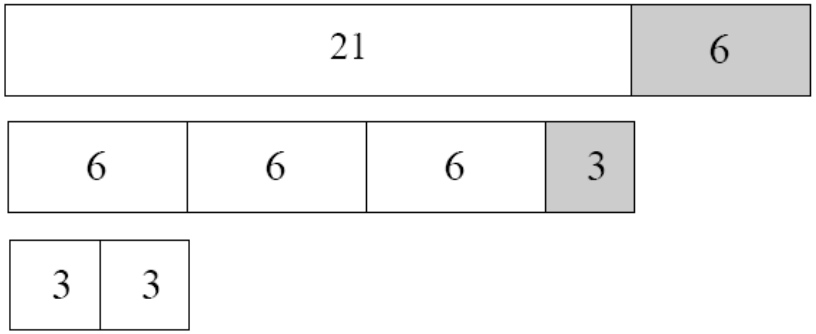
Euclidean Algorithm 2/2

- 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 $r_1 \bmod r_0$
- EEA computes s,t, and the gcd :
- Take the relation **mod r_0**

$$\gcd(r_0, r_1) = s \cdot r_0 + t \cdot r_1$$

$$s \cdot r_0 + t \cdot r_1 = 1$$

$$s \cdot 0 + t \cdot r_1 \equiv 1 \bmod r_0$$

$$r_1 \cdot t \equiv 1 \bmod r_0$$

→ Compare with the definition of modular inverse: **t is the inverse of $r_1 \bmod r_0$**

- 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 $\Phi(m)$**

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

$$\begin{aligned}
&\gcd(0,6) = 6\\
&\gcd(1,6) = 1 \leftarrow\\
&\gcd(2,6) = 2\\
&\gcd(3,6) = 3\\
&\gcd(4,6) = 2\\
&\gcd(5,6) = 1 \leftarrow
\end{aligned}$$

$$\begin{aligned}
&\gcd(0,5) = 5\\
&\gcd(1,5) = 1 \leftarrow\\
&\gcd(2,5) = 1 \leftarrow\\
&\gcd(3,5) = 1 \leftarrow\\
&\gcd(4,5) = 1 \leftarrow
\end{aligned}$$

→ 1 and 5 relatively prime to m=6,
hence $\Phi(6) = 2$

→ $\Phi(5) = 4$

- 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:
  (where p_i primes and e_i positive integers)

$$m = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_n^{e_n}$$

- **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., $m = p \cdot q$ → $\Phi(m) = (p\text{-}1) \cdot (q\text{-}1)$

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

- **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: $\boxed{a^p \equiv a \pmod{p}}$
- 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: $a = 2$, $p = 7$

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

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

- 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*** : $$a^{\Phi(m)} \equiv 1 \pmod{m}$$
- **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: $$a^{\Phi(p)} = a^{p-1} \equiv 1 \pmod{p}$$

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.

Content of this Chapter

- **The RSA Cryptosystem**
- Implementation aspects
- Finding Large Primes
- Attacks and Countermeasures
- Lessons Learned

The RSA Cryptosystem

- Martin Hellman and Whitfield Diffie published their landmark public-key paper in 1976
- Ronald Rivest, Adi Shamir and Leonard Adleman proposed the asymmetric RSA cryptosystem in1977
- Until now, RSA is the most widely use asymmetric cryptosystem although elliptic curve cryptography (ECC) becomes increasingly popular
- RSA is mainly used for two applications
  - Transport of (i.e., symmetric) keys (cf. Chptr 13 of *Understanding Cryptography*)
  - Digital signatures (cf. Chptr 10 of *Understanding Cryptography*)

Encryption and Decryption

- RSA operations are done over the integer ring Z_n (i.e., arithmetic modulo n), where $n = p * q$, with p, q being large primes
- Encryption and decryption are simply exponentiations in the ring

Definition

Given the public key $(n,e) = k_{pub}$ and the private key $d = k_{pr}$ we write

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

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

where $x, y \in Z_n$.

We call $e_{k_{pub}}()$ the encryption and $d_{k_{pr}}()$ the decryption operation.

- In practice x, y, n and d are very long integer numbers (≥ 1024 bits)
- The security of the scheme relies on the fact that it is hard to derive the „private exponent“ d given the public-key (n, e)

Key Generation

- Like all asymmetric schemes, RSA has set-up phase during which the private and public keys are computed

Algorithm: RSA Key Generation

Output: public key: $k_{pub} = (n, e)$ and private key $k_{pr} = d$

1. Choose two large primes p, q
2. Compute $n = p * q$
3. Compute $\Phi(n) = (p-1) * (q-1)$
4. Select the public exponent $e \in \{1, 2, \ldots, \Phi(n)-1\}$ such that $\gcd(e, \Phi(n)) = 1$
5. Compute the private key d such that $d * e \equiv 1 \bmod \Phi(n)$
6. **RETURN** $k_{pub} = (n, e)$, $k_{pr} = d$

Remarks:

- Choosing two large, distinct primes p, q (in Step 1) is non-trivial
- $\gcd(e, \Phi(n)) = 1$ ensures that e has an inverse and, thus, that there is always a private key d

Example: RSA with small numbers

ALICE

Message ***x = 4***

BOB

1. Choose $p = 3$ and $q = 11$
2. Compute $n = p * q = 33$
3. $\Phi(n) = (3-1) * (11-1) = 20$
4. Choose $e = 3$
5. $d \equiv e^{-1} \equiv 7 \bmod 20$

$K_{pub} = (33,3)$ ←

$y = x^e \equiv 4^3 \equiv 31 \bmod 33$

y = 31 →

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

Content of this Chapter

- The RSA Cryptosystem
- **Implementation aspects**
- Finding Large Primes
- Attacks and Countermeasures
- Lessons Learned

Implementation aspects

- The RSA cryptosystem uses only one arithmetic operation (modular exponentiation) which makes it conceptually a simple asymmetric scheme
- Even though conceptually simple, due to the use of very long numbers, RSA is orders of magnitude slower than symmetric schemes, e.g., DES, AES
- When implementing RSA (esp. on a constrained device such as smartcards or cell phones) close attention has to be paid to the correct choice of arithmetic algorithms
- The square-and-multiply algorithm allows fast exponentiation, even with very long numbers…

Square-and-Multiply

- **Basic principle**: Scan exponent bits from left to right and square/multiply operand accordingly

Algorithm: Square-and-Multiply for $x^H \bmod n$

Input: Exponent H, base element x, Modulus n

Output: $y = x^H \bmod n$

1. Determine binary representation $H = (h_t, h_{t-1}, \ldots, h_0)_2$
2. **FOR** $i = t-1$ **TO** 0
3. $y = y^2 \bmod n$
4. **IF** $h_i = 1$ **THEN**
5. $y = y * x \bmod n$
6. **RETURN** y

- Rule: Square in every iteration (Step 3) and multiply current result by x if the exponent bit $h_i = 1$ *(Step 5)*
- Modulo reduction after each step keeps the operand y small

Example: Square-and-Multiply

- Computes x^{26} without modulo reduction
- Binary representation of exponent: $26 = (1,1,0,1,0)_2 = (h_4,h_3,h_2,h_1,h_0)_2$

Step		Binary exponent	Op	Comment
1	$x = x^1$	$(1)_2$		Initial setting, h_4 processed
1a	$(x^1)^2 = x^2$	$(10)_2$	SQ	Processing h_3
1b	$x^2 * x = x^3$	$(11)_2$	MUL	$h_3 = 1$
2a	$(x^3)^2 = x^6$	$(110)_2$	SQ	Processing h_2
2b	-	$(110)_2$	-	$h_0 = 0$
3a	$(x^6)^2 = x^{12}$	$(1100)_2$	SQ	Processing h_1
3b	$x^{12} * x = x^{13}$	$(1101)_2$	MUL	$h_1=1$
4a	$(x^{13})^2 = x^{26}$	$(11010)_2$	SQ	Processing h_0
4b	-	$(11010)_2$	-	$h_0 = 0$

- Observe how the exponent evolves into $x^{26} = x^{11010}$

Complexity of Square-and-Multiply Alg.

- The square-and-multiply algorithm has a logarithmic complexity, i.e., its run time is proportional to the bit length (rather than the absolute value) of the exponent
- Given an exponent with t+1 bits

  $H = (h_t, h_{t-1}, \ldots, h_0)_2$

  with $h_t = 1$, we need the following operations
  - *# Squarings* = t
  - Average # multiplications = $0.5\ t$
  - Total complexity: *#SQ + #MUL* = $1.5\ t$
- Exponents are often randomly chosen, so $1.5\ t$ is a good estimate for the average number of operations
- Note that each squaring and each multiplication is an operation with very long numbers, e.g., 2048 bit integers.

Speed-Up Techniques

- Modular exponentiation is computationally intensive
- Even with the square-and-multiply algorithm, RSA can be quite slow on constrained devices such as smart cards
- Some important tricks:
    - Short public exponent *e*
    - Chinese Remainder Theorem (CRT)
    - Exponentiation with pre-computation *(not covered here)*

Fast encryption with small public exponent

- Choosing a small public exponent *e* does not weaken the security of RSA
- A small public exponent improves the speed of the RSA encryption significantly

Public Key	***e* as binary string**	**#MUL + #SQ**
$2^1+1 = 3$	$(11)_2$	1 + 1 = 2
$2^4+1 = 17$	$(1\ 0001)_2$	4 + 1 = 5
$2^{16} + 1$	$(1\ 0000\ 0000\ 0000\ 0001)_2$	16 + 1 = 17

- This is a commonly used trick (e.g., SSL/TLS, etc.) and makes RSA the fastest asymmetric scheme with regard to encryption!

Fast decryption with CRT

- Choosing a small private key *d* results in security weaknesses!
  - In fact, d must have at least *0.3t* bits, where *t* is the bit length of the modulus *n*
- However, the Chinese Remainder Theorem (CRT) can be used to (somewhat) accelerate exponentiation with the private key *d*
- Based on the CRT we can replace the computation of

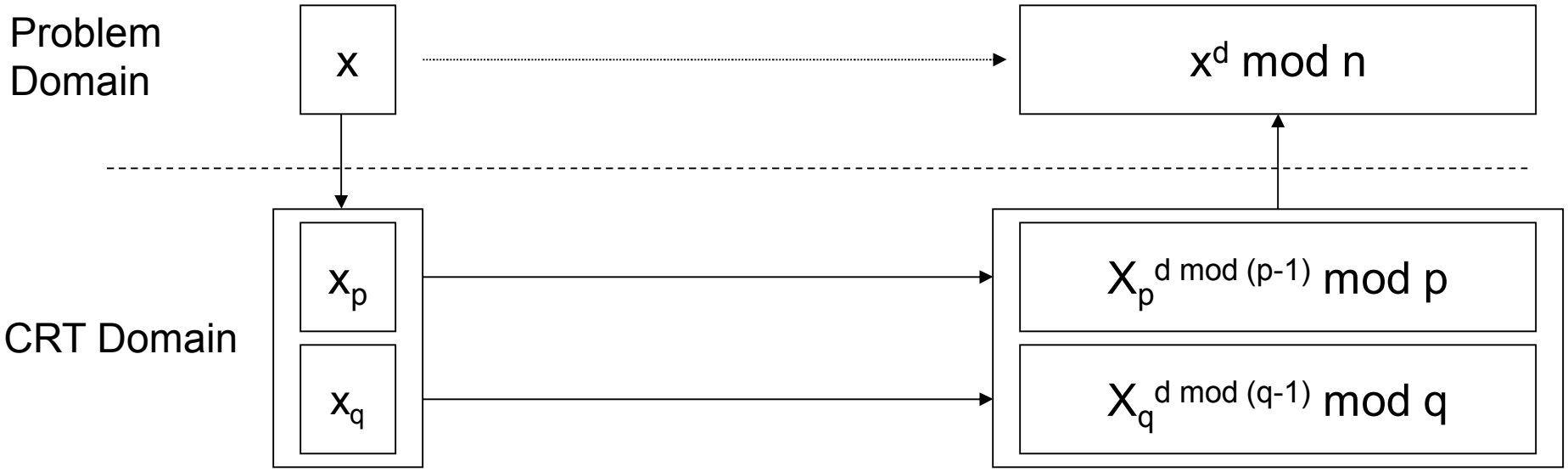
$$x^{d \bmod \Phi(n)} \bmod n$$

by two computations

$$x^{d \bmod (p-1)} \bmod p \quad \text{and} \quad x^{d \bmod (q-1)} \bmod q$$

where *q* and *p* are „small“ compared to *n*

Basic principle of CRT-based exponentiation



- CRT involves three distinct steps

  (1) Transformation of operand into the CRT domain

  (2) Modular exponentiation in the CRT domain

  (3) Inverse transformation into the problem domain

- These steps are equivalent to one modular exponentiation in the problem domain

CRT: Step 1 – Transformation

- Transformation into the CRT domain requires the knowledge of p and q
- p and q are only known to the owner of the private key, hence CRT cannot be applied to speed up encryption
- The transformation computes (x_p, x_q) which is the representation of x in the CRT domain. They can be found easily by computing

$$x_p \equiv x \bmod p \quad \text{and} \quad x_q \equiv x \bmod q$$

CRT: Step 2 – Exponentiation

- Given d_p and d_q such that

$$d_p \equiv d \bmod (p-1) \quad \text{and} \quad d_q \equiv d \bmod (q-1)$$

one exponentiation in the problem domain requires two exponentiations in the CRT domain

$$y_p \equiv x_p^{dp} \bmod p \quad \text{and} \quad y_q \equiv x_q^{dq} \bmod q$$

- In practice, p and q are chosen to have half the bit length of n, i.e., $|p| \approx |q| \approx |n|/2$

CRT: Step 3 – Inverse Transformation

- Inverse transformation requires modular inversion twice, which is computationally expensive

$$c_p \equiv q^{-1} \bmod p \qquad \text{and} \qquad c_q \equiv p^{-1} \bmod q$$

- Inverse transformation assembles y_p, y_q to the final result $y \bmod n$ in the problem domain

$$y \equiv [\, q * c_p \,] * y_p + [\, p * c_q \,] * y_q \bmod n$$

- The primes p and q typically change infrequently, therefore the cost of inversion can be neglected because the two expresssions

$$[\, q * c_p \,] \text{ and } [\, p * c_q \,]$$

can be precomputed and stored

Complexity of CRT

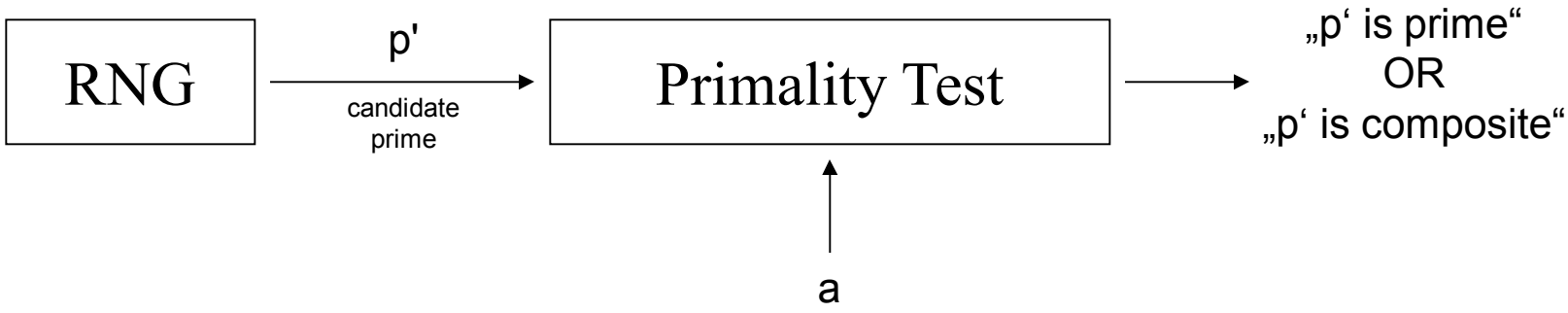
- We ignore the transformation and inverse transformation steps since their costs can be neglected under reasonable assumptions
- Assuming that n has $t+1$ bits, both p and q are about $t/2$ bits long
- The complexity is determined by the two exponentiations in the CRT domain. The operands are only $t/2$ bits long. For the exponentiations we use the square-and-multiply algorithm:
  - # squarings (one exp.): $\#SQ = 0.5\ t$
  - # aver. multiplications (one exp.): #MUL = 0.25t
  - Total complexity: $2 * (\#MUL + \#SQ) = 1.5t$
- This looks the same as regular exponentations, but since the operands have half the bit length compared to regular exponent., each operation (i.e., multipl. and squaring) is 4 times faster!
- Hence CRT is **4 times** faster than straightforward exponentiation

Content of this Chapter

- The RSA Cryptosystem
- Implementation aspects
- **Finding Large Primes**
- Attacks and Countermeasures
- Lessons Learned

Finding Large Primes

- Generating keys for RSA requires finding two large primes *p* and *q* such that $n = p * q$ is sufficiently large
- The size of *p* and *q* is typically half the size of the desired size of *n*
- To find primes, random integers are generated and tested for primality:



- The random number generator (RNG) should be non-predictable otherwise an attacker could guess the factorization of *n*

Primality Tests

- Factoring *p* and *q* to test for primality is typically not feasible
- However, we are not interested in the factorization, we only want to know whether *p* and *q* are composite
- Typical primality tests are probabilistic, i.e., they are not 100% accurate but their output is correct with very high probability
- A probabilistic test has two outputs:
  - „p‘ is composite“ – always true
  - „p‘ is a prime“ – only true with a certain probability
- Among the well-known primality tests are the following
  - Fermat Primality-Test
  - Miller-Rabin Primality-Test

Fermat Primality-Test

- Basic idea: Fermat's Little Theorem holds for all primes, i.e., if a number *p'* is found for which $a^{p'-1} \not\equiv 1 \bmod p'$, it is not a prime

Algorithm: Fermat Primality-Test

Input: Prime candidate *p'*, security parameter *s*

Output: „*p'* is composite" or „*p'* is likely a prime"

1. **FOR** *i = 1* **TO** *s*
2. choose random $a \in \{2,3, \ldots, p'-2\}$
3. **IF** $a^{p'-1} \not\equiv 1 \bmod p'$ **THEN**
4. **RETURN** „*p'* is composite"
5. **RETURN** „*p'* is likely a prime"

- For certain numbers („Carchimchael numbers") this test returns „*p'* is likely a prime" often – although these numbers are composite
- Therefore, the Miller-Rabin Test is preferred

Theorem for Miller-Rabin's test

- The more powerful Miller-Rabin Test is based on the following theorem

Theorem

Given the decomposition of an odd prime candidate p'

$$p' - 1 = 2^u * r$$

where r is odd. If we can find an integer a such that

$$a^r \not\equiv 1 \bmod p' \quad \text{and} \quad a^{r2^j} \not\equiv p' - 1 \bmod p'$$

For all $j = \{0,1, \ldots, u-1\}$, then p' is composite.

Otherwise it is probably a prime.

- This theorem can be turned into an algorithm

Miller-Rabin Primality-Test

Algorithm: Miller-Rabin Primality-Test

Input: Prime candidate *p'* with $p'-1 = 2^{u} * r$ security parameter *s*

Output: „*p'* is composite“ or „*p'* is likely a prime“

1. **FOR** $i = 1$ **TO** s
2. choose random $a \in \{2,3, \ldots, p'-2\}$
3. $z \equiv a^{r} \bmod p'$
4. **IF** $z \neq 1$ **AND** $z \neq p'-1$ **THEN**
5. **FOR** $j = 1$ **TO** $u-1$
6. $z \equiv z^{2} \bmod p'$
7. **IF** $z = 1$ **THEN**
8. **RETURN** „*p'* is composite“
9. **IF** $z \neq p'-1$ **THEN**
10. **RETURN** „*p'* is composite“
11. **RETURN** „*p'* is likely a prime“

Content of this Chapter

- The RSA Cryptosystem
- Implementation aspects
- Finding Large Primes
- **Attacks and Countermeasures**
- Lessons Learned

Attacks and Countermeasures 1/3

- There are two distinct types of attacks on cryptosystems
  - **Analytical attacks** try to break the mathematical structure of the underlying problem of RSA
  - **Implementation attacks** try to attack a real-world implementation by exploiting inherent weaknesses in the way RSA is realized in software or hardware

Attacks and Countermeasures 2/3

RSA is typically exposed to these analytical attack vectors

- **Mathematical attacks**
  - The best known attack is factoring of n in order to obtain $\Phi(n)$
  - Can be prevented using a sufficiently large modulus n
  - The current factoring record is 664 bits. Thus, it is recommended that n should have a bit length between 1024 and 3072 bits
- **Protocol attacks**
  - Exploit the malleability of RSA, i.e., the property that a ciphertext can be transformed into another ciphertext which decrypts to a related plaintext – without knowing the private key
  - Can be prevented by proper padding

Attacks and Countermeasures 3/3

- Implementation attacks can be one of the following
  - **Side-channel analysis**
    - Exploit physical leakage of RSA implementation (e.g., power consumption, EM emanation, etc.)
  - **Fault-injection attacks**
    - Inducing faults in the device while CRT is executed can lead to a complete leakage of the private key

More on all attacks can be found in Section 7.8 of *Understanding Cryptography*

Content of this Chapter

- The RSA Cryptosystem
- Implementation aspects
- Finding Large Primes
- Attacks and Countermeasures
- **Lessons Learned**

Lessons Learned

- RSA is the most widely used public-key cryptosystem
- RSA is mainly used for key transport and digital signatures
- The public key *e* can be a short integer, the private key *d* needs to have the full length of the modulus *n*
- RSA relies on the fact that it is hard to factorize *n*
- Currently 1024-bit cannot be factored, but progress in factorization could bring this into reach within 10-15 years. Hence, RSA with a 2048 or 3076 bit modulus should be used for long-term security
- A naïve implementation of RSA allows several attacks, and in practice RSA should be used together with padding

Content of this Chapter

- Diffie–Hellman Key Exchange
- The Discrete Logarithm Problem
- Security of the Diffie–Hellman Key Exchange
- The Elgamal Encryption Scheme

Diffie–Hellman Key Exchange: Overview

- Proposed in 1976 by **Whitfield Diffie and Martin Hellman**
- **Widely used**, e.g. in Secure Shell (SSH), Transport Layer Security (TLS), and Internet Protocol Security (IPSec)
- The Diffie–Hellman Key Exchange (DHKE) is a key exchange protocol and **not** used for encryption

  (For the purpose of encryption based on the DHKE, ElGamal can be used.)

Diffie–Hellman Key Exchange: Set-up

1. Choose a large prime p.
2. Choose an integer $\alpha \in \{2,3, \ldots, p-2\}$.
3. Publish p *and* α.

Diffie–Hellman Key Exchange

Alice		Bob
Choose random private key $k_{prA}=a \in \{1,2,\ldots,p-1\}$		Choose random private key $k_{prB}=b \in \{1,2,\ldots,p-1\}$
Compute corresponding public key $k_{pubA}= A = \alpha^a \bmod p$	$A \rightarrow$	
	$\leftarrow B$	Compute correspondig public key $k_{pubB}= B = \alpha^b \bmod p$
Compute common secret $k_{AB} = B^a = (\alpha^a)^b \bmod p$		Compute common secret $k_{AB} = A^b = (\alpha^b)^a \bmod p$

We can now use the joint key k_{AB} for encryption, e.g., with AES

$$y = AES_{kAB}(x) \quad \xrightarrow{\;y\;} \quad x = AES^{-1}_{kAB}(y)$$

Diffie–Hellman Key Exchange: Example

Domain parameters p=29, α=2

Alice		Bob
Choose random private key $k_{prA} = a = 5$		Choose random private key $k_{prB} = b = 12$
Compute corresponding public key $k_{pubA} = A = 2^5 = 3 \bmod 29$	A →	
	← B	Compute correspondig public key $k_{pubB} = B = 2^{12} = 7 \bmod 29$
Compute common secret $k_{AB} = B^a = 7^5 = 16 \bmod 29$		Compute common secret $k_{AB} = A^b = 3^{12} = 16 \bmod 29$

Proof of correctness:

Alice computes: $B^a = (\alpha^b)^a \mod p$
Bob computes: $A^b = (\alpha^a)^b \mod p$

i.e., Alice and Bob compute the same key k_{AB} !

The Discrete Logarithm Problem

Discrete Logarithm Problem (DLP) in Z_p^*

- Given is the finite cyclic group Z_p^* of order $p-1$ and a primitive element $\alpha \in Z_p^*$ and another element $\beta \in Z_p^*$.
- The DLP is the problem of determining the integer $1 \leq x \leq p-1$ such that $\alpha^x \equiv \beta \bmod p$
- This computation is called the discrete logarithm problem (DLP)

$$x = log_\alpha \beta \bmod p$$

- Example: Compute x for $5^x \equiv 41 \bmod 47$

Remark: For the coverage of groups and cylcic groups, we refer to Chapter 8 of *Understanding Cryptography*

The Generalized Discrete Logarithm Problem

- Given is a finite cyclic group G with the group operation $\circ$ and cardinality n.
- We consider a primitive element $\alpha \in G$ and another element $\beta \in G$.
- The discrete logarithm problem is finding the integer x, where $1 \leq x \leq n$, such that:

$$\beta = \underbrace{\alpha \circ \alpha \circ \alpha \circ \ldots \circ \alpha}_{x \text{ times}} = \alpha^x$$

The Generalized Discrete Logarithm Problem

The following discrete logarithm problems have been proposed for use in cryptography

1. The multiplicative group of the prime field Z_p *or a subgroup of it. For instance,* the classical DHKE uses this group (cf. previous slides), but also Elgamal encryption or the Digital Signature Algorithm (DSA).
2. The cyclic group formed by an elliptic curve (see Chapter 9)
3. The multiplicative group of a Galois field *GF(*2^m) or a subgroup of it. *S*chemes such as the DHKE can be realized with them.
4. Hyperelliptic curves or algebraic varieties, which can be viewed as generalization of elliptic curves.

Remark: The groups 1. and 2. are most often used in practice.

Attacks against the Discrete Logarithm Problem

- Security of many asymmetric primitives is based on the difficulty of computing the DLP in cyclic groups, i.e.,

  Compute x for a given α and β such that $\beta = \alpha \circ \alpha \circ \alpha \circ \ldots \circ \alpha = \alpha^x$

- The following algorithms for computing discrete logarithms exist
  - Generic algorithms: Work in any cyclic group
    - Brute-Force Search
    - Shanks' Baby-Step-Giant-Step Method
    - Pollard's Rho Method
    - Pohlig-Hellman Method
  - Non-generic Algorithms: Work only in specific groups, in particular in Z_p
    - The Index Calculus Method
- Remark: Elliptic curves can only be attacked with generic algorithms which are weaker than non-generic algorithms. Hence, elliptic curves are secure with shorter key lengths than the DLP in prime fields Z_p

Attacks against the Discrete Logarithm Problem

Summary of records for computing discrete logarithms in Z_p^*

Decimal digits	Bit length	Date
58	193	1991
68	216	1996
85	282	1998
100	332	1999
120	399	2001
135	448	2006
160	532	2007

In order to prevent attacks that compute the DLP, it is recommended to use primes with a length of at least 1024 bits for schemes such as Diffie-Hellman in Z_p^*

Security of the classical Diffie–Hellman Key Exchange

- Which information does Oscar have?
  - α, p
  - $k_{pubA} = A = \alpha^a \bmod p$
  - $k_{pubB} = B = \alpha^b \bmod p$
- Which information does Oscar want to have?
  - $k_{AB} = \alpha^{ba} = \alpha^{ab} = \bmod p$
  - This is kown as Diffie-Hellman Problem (DHP)
- The only known way to solve the DHP is to solve the DLP, i.e.

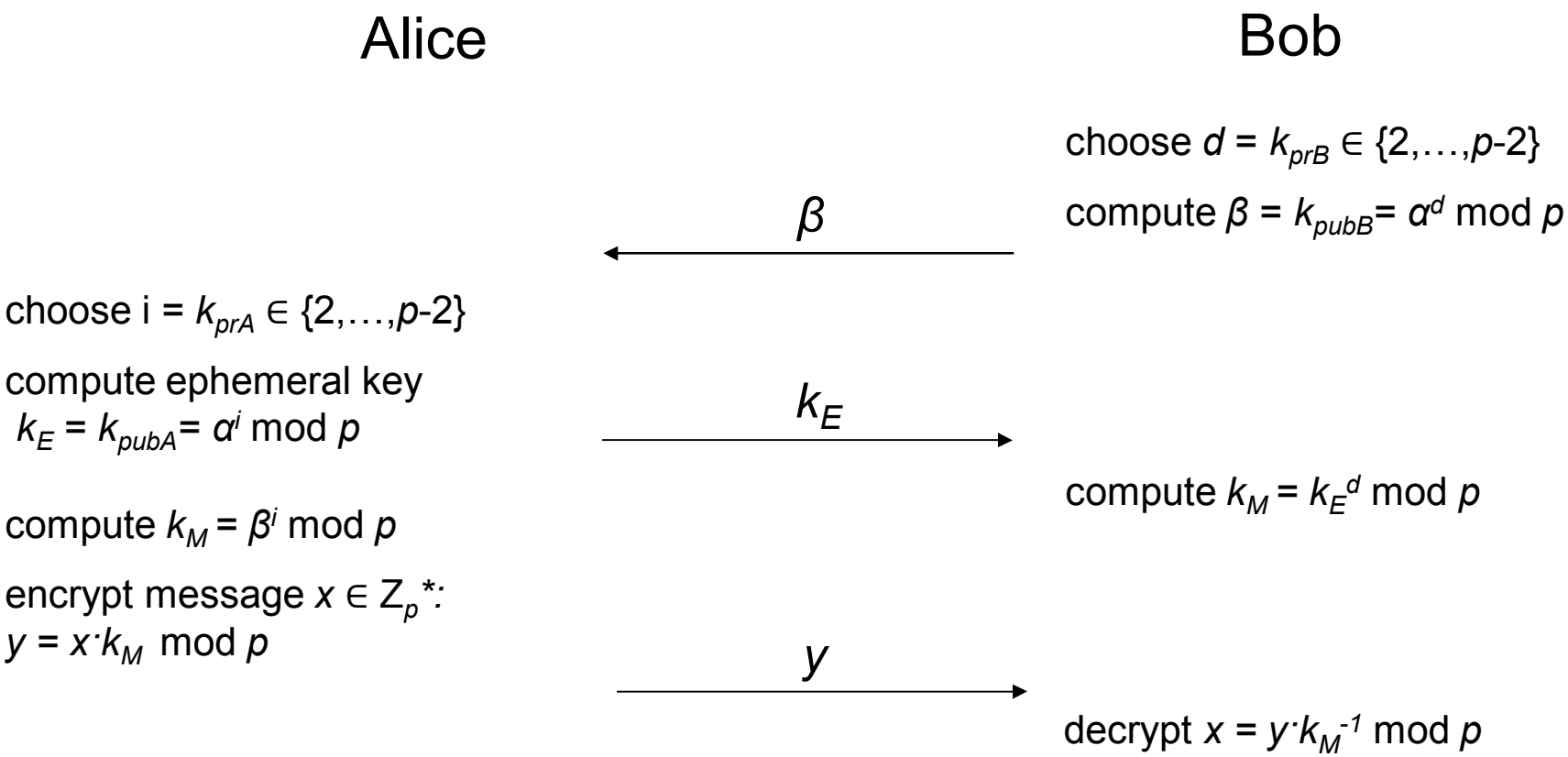
  1. Compute $a = log_\alpha A \bmod p$
  2. *Compute* $k_{AB} = B^a = \alpha^{ba} = \bmod p$

  It is conjectured that the DHP and the DLP are equivalent, i.e., solving the DHP implies solving the DLP.
- To prevent attacks, i.e., to prevent that the DLP can be solved, choose $p > 2^{1024}$

The Elgamal Encryption Scheme: Overview

- Proposed by Taher Elgamal in 1985
- Can be viewed as an extension of the DHKE protocol
- Based on the intractability of the discrete logarithm problem and the Diffie–Hellman problem

The Elgamal Encryption Scheme: Principle



This looks very similar to the DHKE! The actual Elgamal protocol re-orders the computations which helps to save one communication (cf. next slide)

The Elgamal Encryption Protocol

Alice		Bob
		choose large prime p
		choose primitive element $\alpha \in Z_p^*$ or in a subgroup of Z_p^*
		choose $d = k_{prB} \in \{2,\ldots,p\text{-}2\}$
	$k_{pubB} = (p, \alpha, \beta)$ ←	compute $\beta = k_{pubB} = \alpha^d \bmod p$
choose i = $k_{prA} \in \{2,\ldots,p\text{-}2\}$		
compute $k_E = k_{pubA} = \alpha^i \bmod p$		
compute masking key $k_M = \beta^i \bmod p$		
encrypt message $x \in Z_p^*$: $y = x \cdot k_M \bmod p$	(k_E, y) →	
		compute masking key $k_M = k_E^d \bmod p$
		decrypt $x = y \cdot k_M^{-1} \bmod p$

Computational Aspects

- Key Generation
  - Generation of prime p
  - p has to of size of at least 1024 bits
  - cf. Section 7.6 in *Understanding Cryptography* for prime-finding algorithms
- Encryption
  - Requires two modular exponentiations and a modular multiplictation
  - All operands have a bitlength of $\log_2 p$
  - Efficient execution requires methods such as the square-and-multiply algorithm (cf. Chapter 7)
- Decryption
  - Requires one modular exponentiation and one modulare inversion
  - As shown *in Understanding Cryptography*, the inversion can be computed from the ephemeral key

Security

- Passive attacks
  - Attacker eavesdrops p, α, $\beta = \alpha^d$, $k_E = \alpha^i$, $y = x \cdot \beta^i$ and wants to recover x
  - Problem relies on the DLP

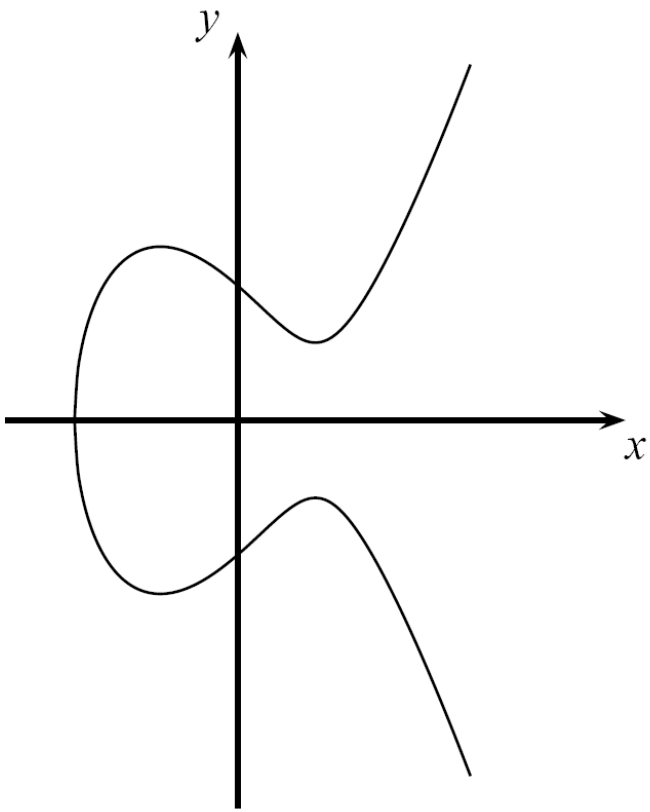
- Active attacks
  - If the public keys are not authentic, an attacker could send an incorrect public key (cf. Chapter 13)
  - An Attack is also possible if the secret exponent i is being used more than once (cf. *Understanding Cryptography* for more details on the attack)

Lessons Learned

- The Diffie–Hellman protocol is a widely used method for key exchange. It is based on cyclic groups.
- The discrete logarithm problem is one of the most important one-way functions in modern asymmetric cryptography. Many public-key algorithms are based on it.
- For the Diffie–Hellman protocol in Z_p^*, *the prime p should be at least 1024 bits* long. This provides a security roughly equivalent to an 80-bit symmetric cipher.
- For a better long-term security, a prime of length 2048 bits should be chosen.
- The Elgamal scheme is an extension of the DHKE where the derived session key is used as a multiplicative masked to encrypt a message.
- Elgamal is a probabilistic encryption scheme, i.e., encrypting two identical messages does not yield two identical ciphertexts.

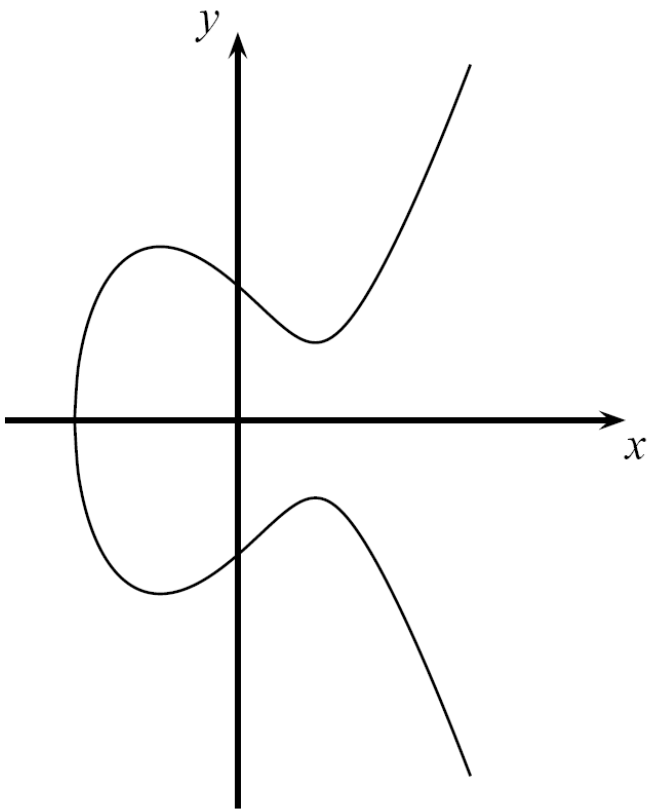
Content of this Chapter

- Introduction
- Computations on Elliptic Curves
- The Elliptic Curve Diffie-Hellman Protocol
- Security Aspects
- Implementation in Software and Hardware

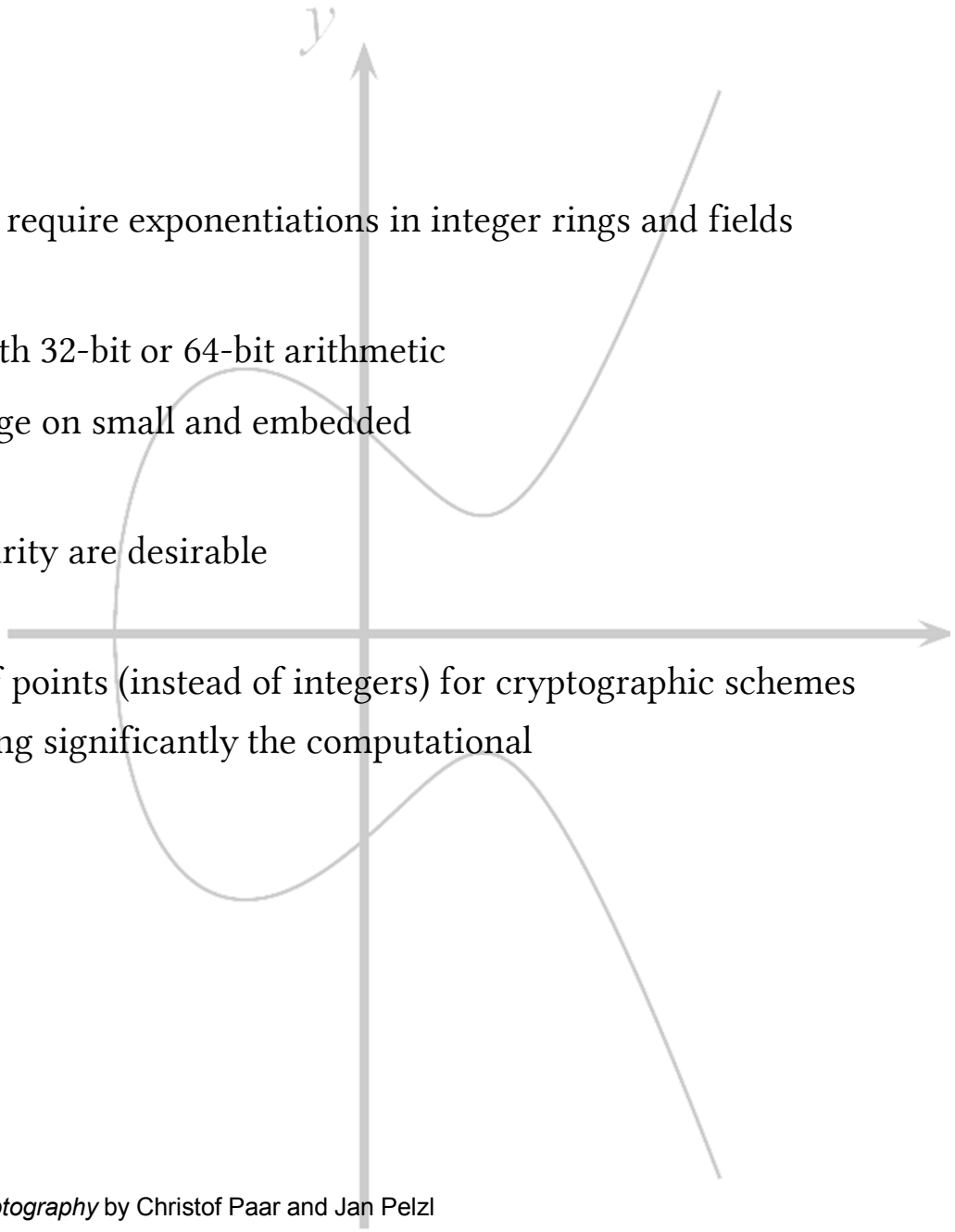


Content of this Chapter

- Introduction
- Computations on Elliptic Curves
- The Elliptic Curve Diffie-Hellman Protocol
- Security Aspects
- Implementation in Software and Hardware



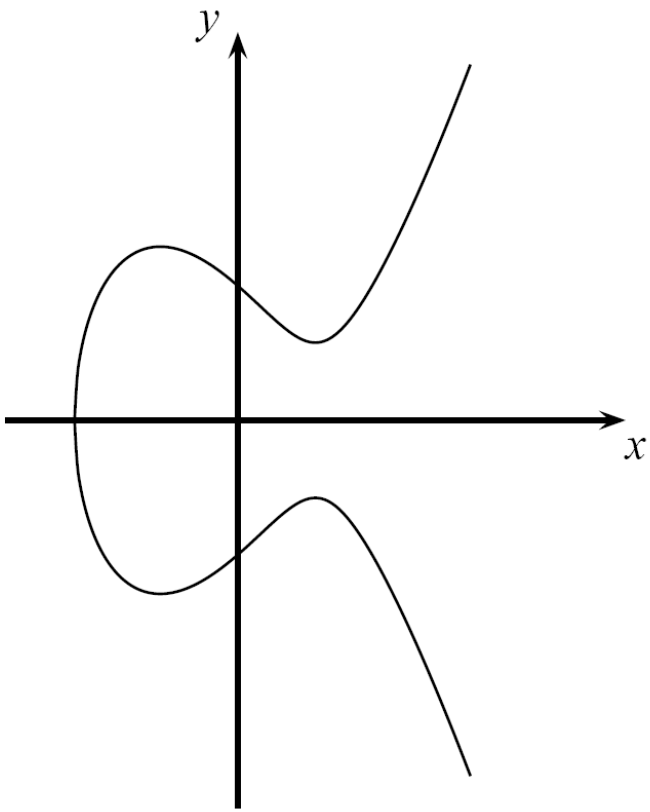
Motivation



- **Problem:**
  Asymmetric schemes like RSA and Elgamal require exponentiations in integer rings and fields with parameters of more than 1000 bits.
  - High computational effort on CPUs with 32-bit or 64-bit arithmetic
  - Large parameter sizes critical for storage on small and embedded
- **Motivation:**
  Smaller field sizes providing equivalent security are desirable
- **Solution:**
  Elliptic Curve Cryptography uses a group of points (instead of integers) for cryptographic schemes with coefficient sizes of 160-256 bits, reducing significantly the computational effort.

Content of this Chapter

- Introduction
- **Computations on Elliptic Curves**
- The Elliptic Curve Diffie-Hellman Protocol
- Security Aspects
- Implementation in Software and Hardware



Computations on Elliptic Curves

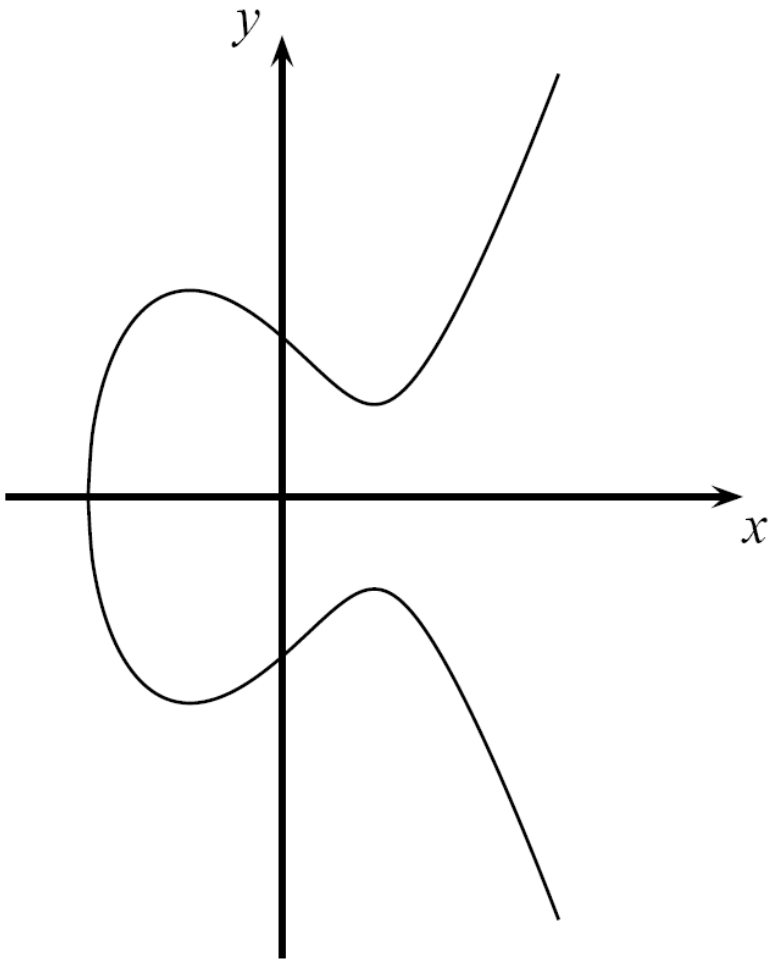
- Elliptic curves are polynomials that define points based on the (simplified) Weierstraß equation:

$$y^2 = x^3 + ax + b$$

for parameters a,b that specify the exact shape of the curve

- On the real numbers and with parameters a, b $\in$ R, an elliptic curve looks like this →

- Elliptic curves can not just be defined over the real numbers *R* but over many other types of finite fields.



Example: $y^2 = x^3 - 3x + 3$ over *R*

Computations on Elliptic Curves (ctd.)

- In cryptography, we are interested in elliptic curves module a prime p:

Definition: Elliptic Curves over prime fields

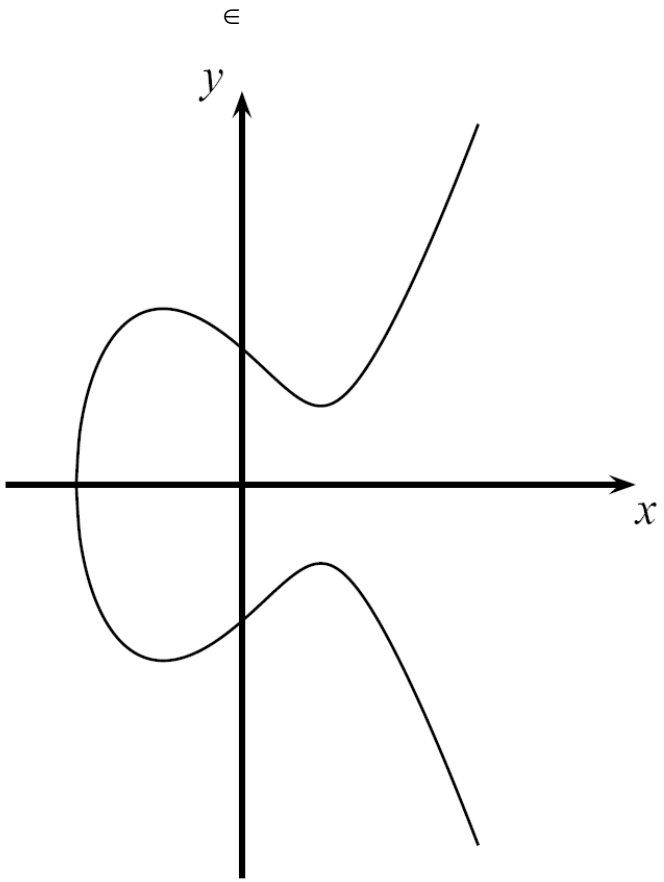
The elliptic curve over Z_p, $p>3$ is the set of all pairs $(x,y) \in Z_p$ which fulfill

$$y^2 = x^3 + ax + b \bmod p$$

together with an imaginary point of infinity θ, where $a,b \in Z_p$ and the condition

$$4a^3+27b^2 \neq 0 \bmod p.$$

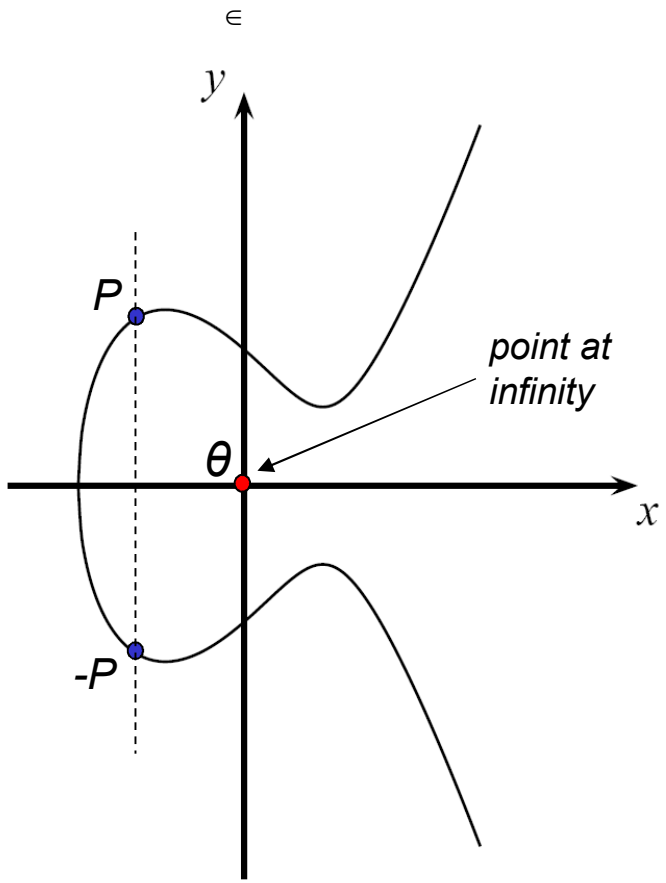
- Note that $Z_p = \{0,1,\ldots, p-1\}$ is a set of integers with modulo p arithmetic



Computations on Elliptic Curves (ctd.)

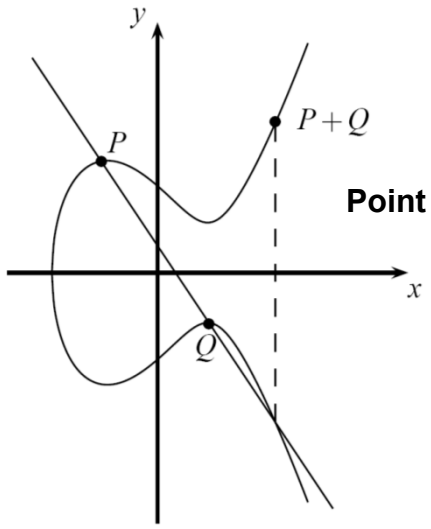
- Some special considerations are required to convert elliptic curves into a group of points
  - *In any group, a special element is required to allow for the identity operation, i.e.,*
    given $P \in E$: $P + \theta = P = \theta + P$
  - *This identity point (which is not on the curve) is additionally added to the group definition*
  - *This (infinite) identity point is denoted by θ*

- Elliptic Curve are symmetric along the x-axis
  - Up to two solutions *y and $-y$* exist for each quadratic residue x of the elliptic curve
  - For each point $P = (x,y)$, the inverse or negative point is defined as $-P = (x,-y)$

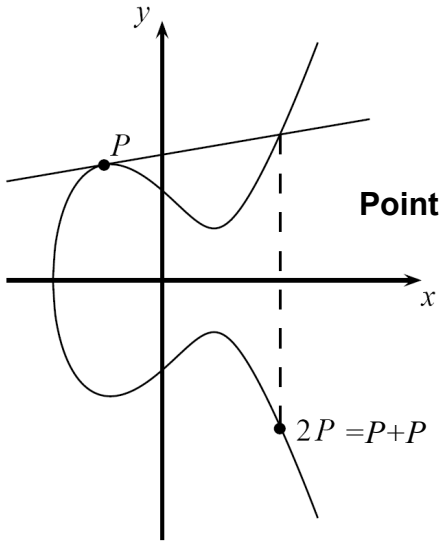


Computations on Elliptic Curves (ctd.)

- Generating a *group of points* on elliptic curves based on point addition operation $P+Q = R$, *i.e.*, $(x_P,y_P)+(x_Q,y_Q) = (x_R,y_R)$
- Geometric Interpretation of point addition operation
  - *Draw straight line through P and Q; if P=Q use tangent line instead*
  - *Mirror third intersection point of drawn line with the elliptic curve along the x-axis*
- Elliptic Curve Point Addition and Doubling Formulas



Point Addition



Point Doubling

$x_3 = s^2 - x_1 - x_2 \bmod p$ and $y_3 = s(x_1 - x_3) - y_1 \bmod p$

where

$$s = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} \bmod p & \text{; if P} \neq \text{Q (point addition)} \\ \dfrac{3x_1^2 + a}{2y_1} \bmod p & \text{; if P = Q (point doubling)} \end{cases}$$

Computations on Elliptic Curves (ctd.)

- **Example**: Given $E: y^2 = x^3+2x+2 \bmod 17$ and point $P=(5,1)$

  Goal: Compute $2P = P+P = (5,1)+(5,1)= (x_3,y_3)$

$$s = \frac{3x_1^2 + a}{2y_1} = (2 \cdot 1)^{-1}(3 \cdot 5^2 + 2) = 2^{-1} \cdot 9 \equiv 9 \cdot 9 \equiv 13 \bmod 17$$

$$x_3 = s^2 - x_1 - x_2 = 13^2 - 5 - 5 = 159 \equiv 6 \bmod 17$$

$$y_3 = s(x_1 - x_3) - y_1 = 13(5 - 6) - 1 = -14 \equiv 3 \bmod 17$$

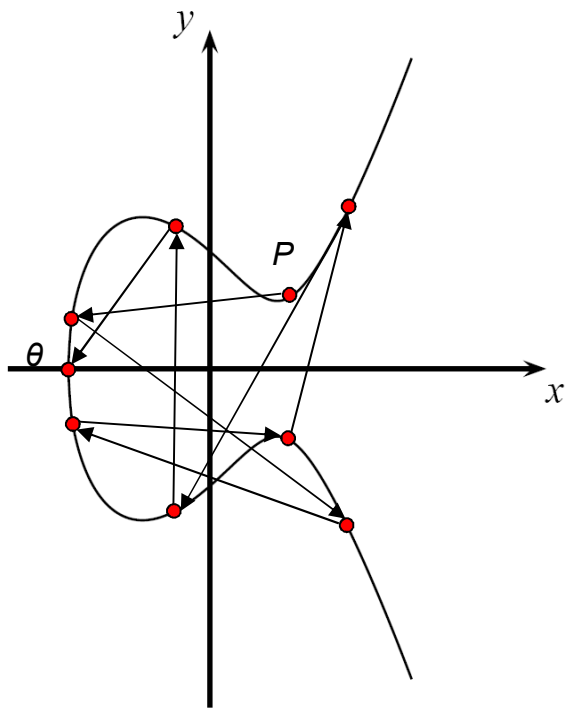
Finally $2P = (5,1) + (5,1) = (6,3)$

Computations on Elliptic Curves (ctd.)

- The points on an elliptic curve and the point at infinity θ form cyclic subgroups

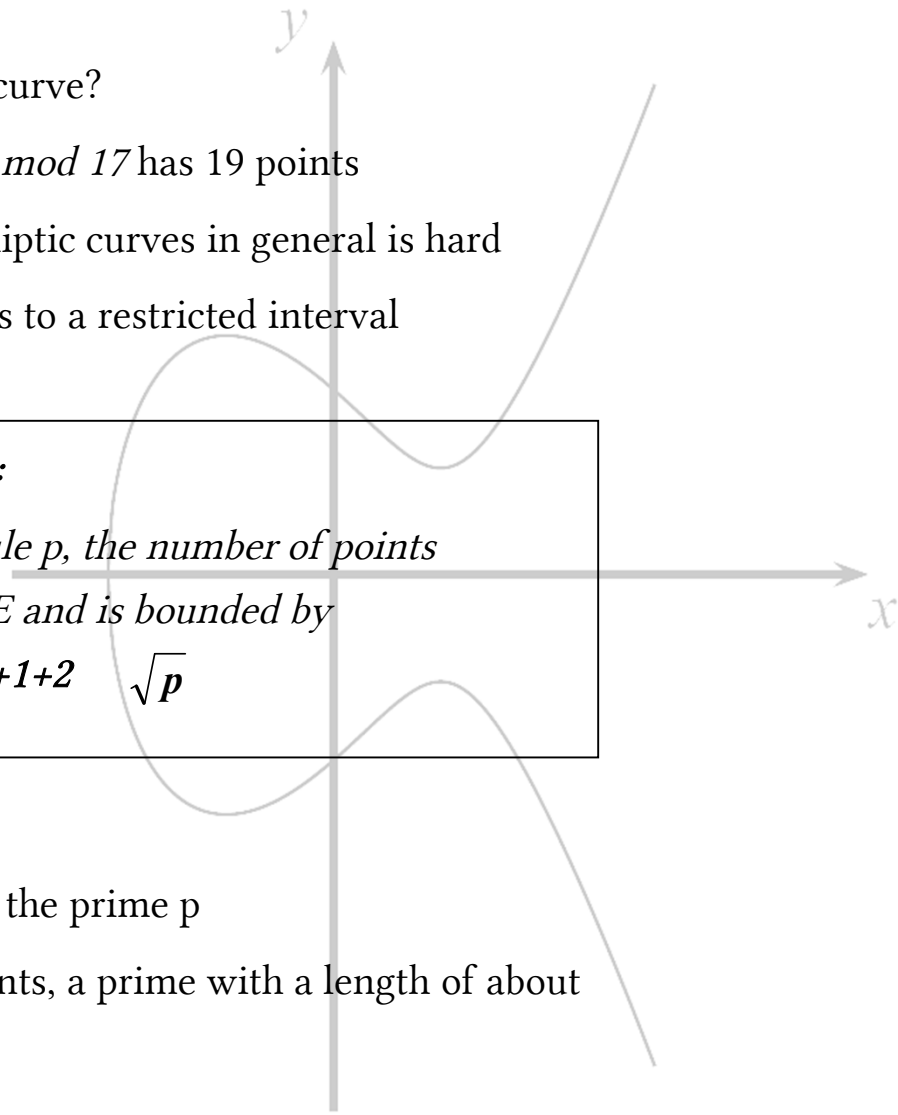
$2P = (5,1)+(5,1) = (6,3)$
$3P = 2P+P = (10,6)$
$4P = (3,1)$
$5P = (9,16)$
$6P = (16,13)$
$7P = (0,6)$
$8P = (13,7)$
$9P = (7,6)$
$10P = (7,11)$
$11P = (13,10)$
$12P = (0,11)$
$13P = (16,4)$
$14P = (9,1)$
$15P = (3,16)$
$16P = (10,11)$
$17P = (6,14)$
$18P = (5,16)$
$19P = \theta$

This elliptic curve has order #E = |E| = 19 since it contains 19 points in its cyclic group.



Number of Points on an Elliptic Curve

- How many points can be on an arbitrary elliptic curve?
  - Consider previous example: $E: y^2 = x^3+2x+2 \bmod 17$ has 19 points
  - However, determining the point count on elliptic curves in general is hard
- But Hasse's theorem bounds the number of points to a restricted interval

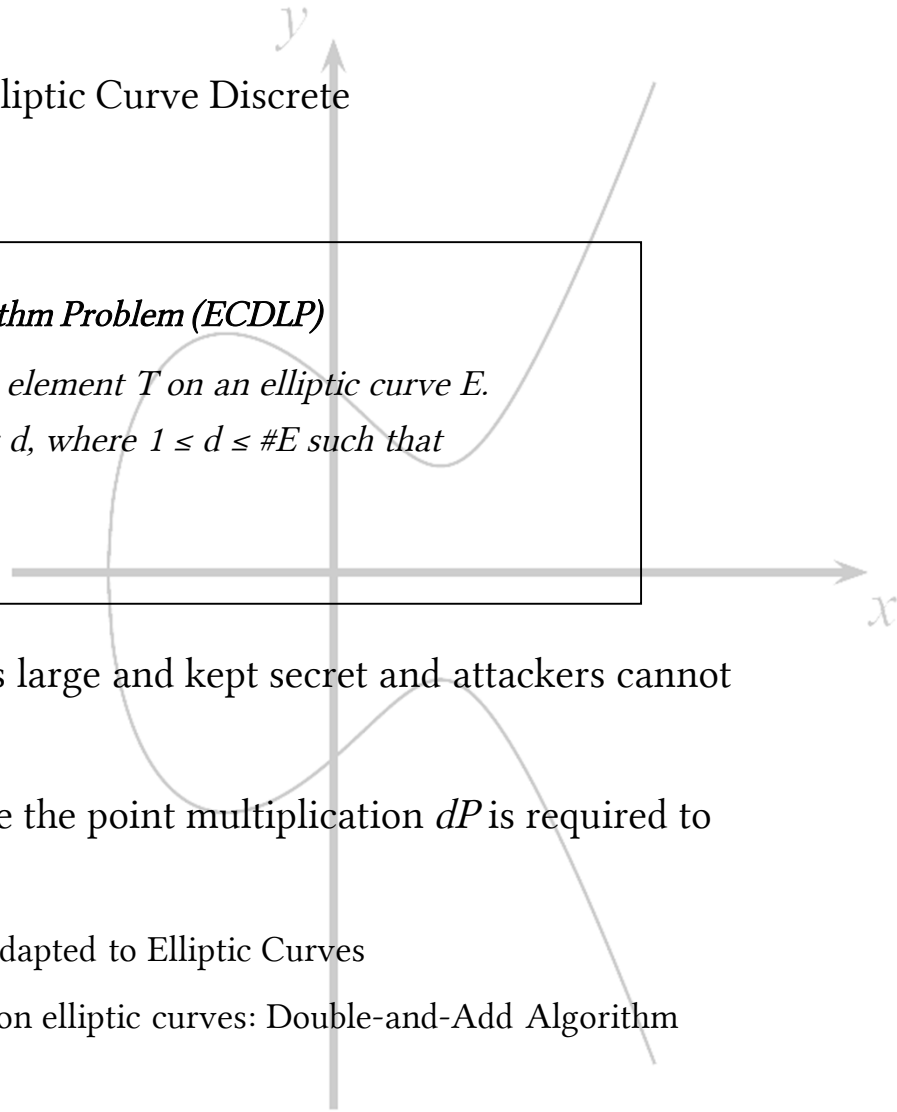


> ***Definition: Hasse's Theorem:***
>
> *Given an elliptic curve module p, the number of points on the curve is denoted by #E and is bounded by*
>
> $$p+1-2\sqrt{p} \le \#E \le p+1+2\sqrt{p}$$

- **Interpretation:** The number of points is „close to" the prime p
- **Example:** To generate a curve with about 2^{160} points, a prime with a length of about 160 bits is required

Elliptic Curve Discrete Logarithm Problem

- Cryptosystems rely on the hardness of the Elliptic Curve Discrete Logarithm Problem (ECDLP)



> ***Definition: Elliptic Curve Discrete Logarithm Problem (ECDLP)***
>
> *Given a primitive element P and another element T on an elliptic curve E.*
> *The ECDL problem is finding the integer d, where $1 \leq d \leq \#E$ such that*
>
> $$\underbrace{P + P + \ldots + P}_{d \text{ times}} = dP = T.$$

- Cryptosystems are based on the idea that d is large and kept secret and attackers cannot compute it easily
- If d is known, an efficient method to compute the point multiplication dP is required to create a reasonable cryptosystem
  - Known Square-and-Multiply Method can be adapted to Elliptic Curves
  - The method for efficient point multiplication on elliptic curves: Double-and-Add Algorithm

Double-and-Add Algorithm for Point Multiplication

Double-and-Add Algorithm

Input: Elliptic curve E, an elliptic curve point P and a scalar d with bits d_i

Output: $T = dP$

Initialization:

$T = P$

Algorithm:

FOR $i = t-1$ DOWNTO 0

$T = T + T \bmod n$

IF $d_i = 1$

$T = T + P \bmod n$

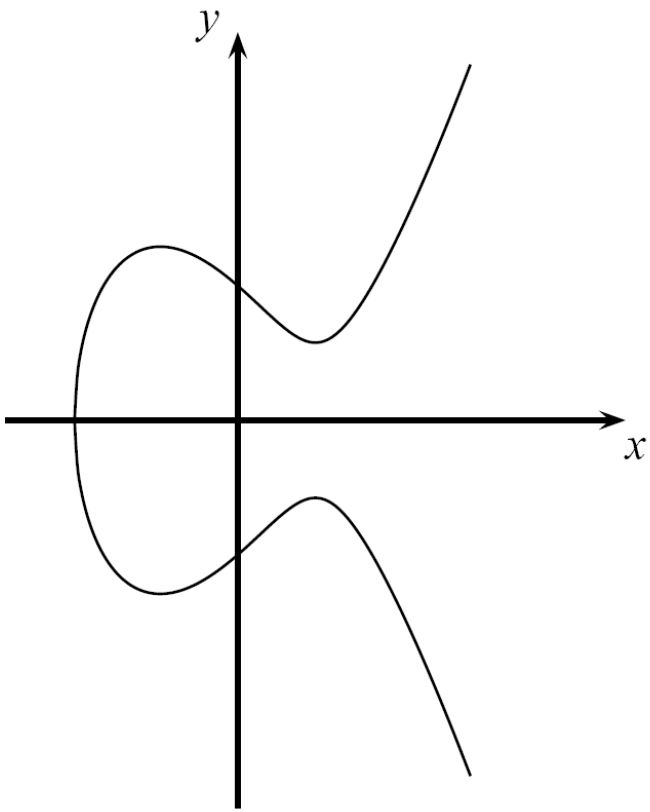
RETURN (T)

Example: $26P = (11010_2)P = (d_4 d_3 d_2 d_1 d_0)_2 \, P$.

Step		
#0	$P = \mathbf{1}_2 P$	inital setting
#1*a*	$P+P = 2P = \mathbf{10}_2 P$	DOUBLE (bit d_3)
#1*b*	$2P+P = 3P = 10^2 P + 1_2 P = \mathbf{11}_2 P$	ADD (bit d_3=1)
#2*a*	$3P+3P = 6P = 2(11_2 P) = \mathbf{110}_2 P$	DOUBLE (bit d_2)
#2*b*		no ADD ($d_2 = 0$)
#3*a*	$6P+6P = 12P = 2(110_2 P) = \mathbf{1100}_2 P$	DOUBLE (bit d_1)
#3*b*	$12P+P = 13P = 1100_2 P + 1_2 P = \mathbf{1101}_2 P$	ADD (bit d_1=1)
#4*a*	$13P+13P = 26P = 2(1101_2 P) = \mathbf{11010}_2 P$	DOUBLE (bit d_0)
#4*b*		no ADD ($d_0 = 0$)

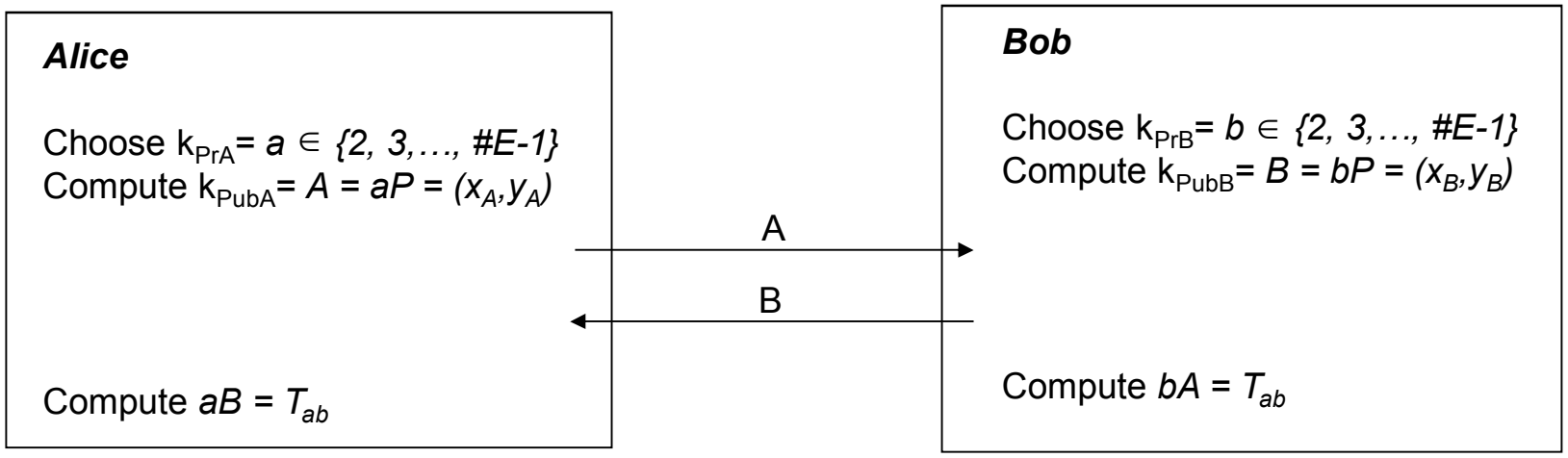
Content of this Chapter

- Introduction
- Computations on Elliptic Curves
- **The Elliptic Curve Diffie-Hellman Protocol**
- Security Aspects
- Implementation in Software and Hardware



The Elliptic Curve Diffie-Hellman Key Exchange (ECDH)

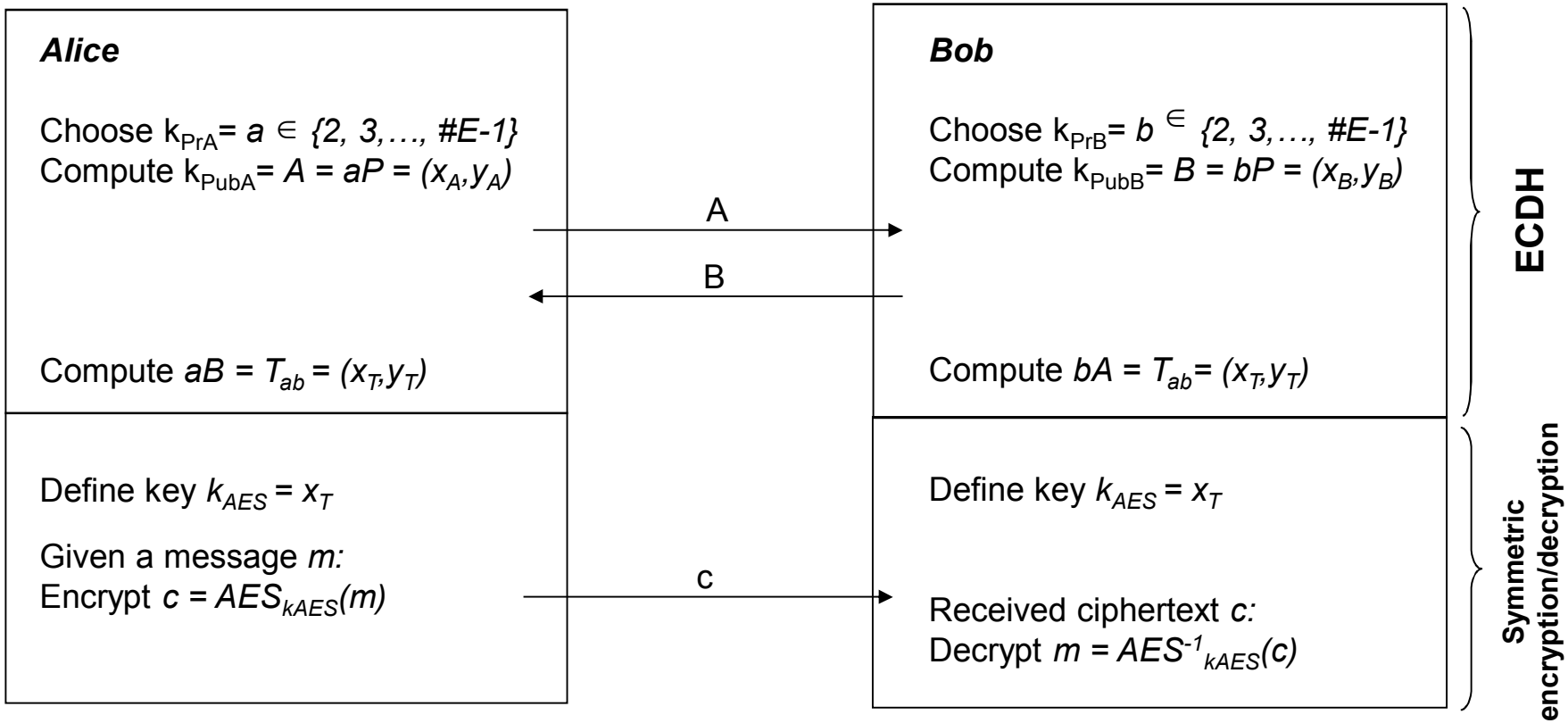
- Given a prime p, a suitable elliptic curve E and a point $P=(x_P,y_P)$
- The Elliptic Curve Diffie-Hellman Key Exchange is defined by the following protocol:



- Joint secret between Alice and Bob: $T_{AB} = (x_{AB}, y_{AB})$
- Proof for correctness:
  - Alice computes aB=a(bP)=abP
  - Bob computes bA=b(aP)=abP since group is associative
- One of the coordinates of the point T_{AB} (usually the x-coordinate) can be used as session key (often after applying a hash function)

The Elliptic Curve Diffie-Hellman Key Exchange (ECDH) (ctd.)

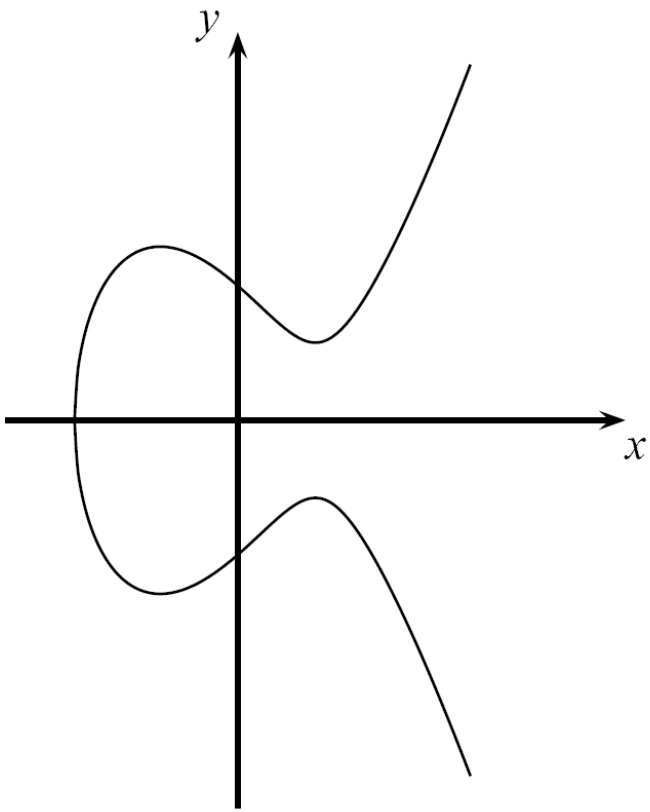
- The ECDH is often used to derive session keys for (symmetric) encryption
- One of the coordinates of the point T_{AB} (usually the x-coordinate) is taken as session key



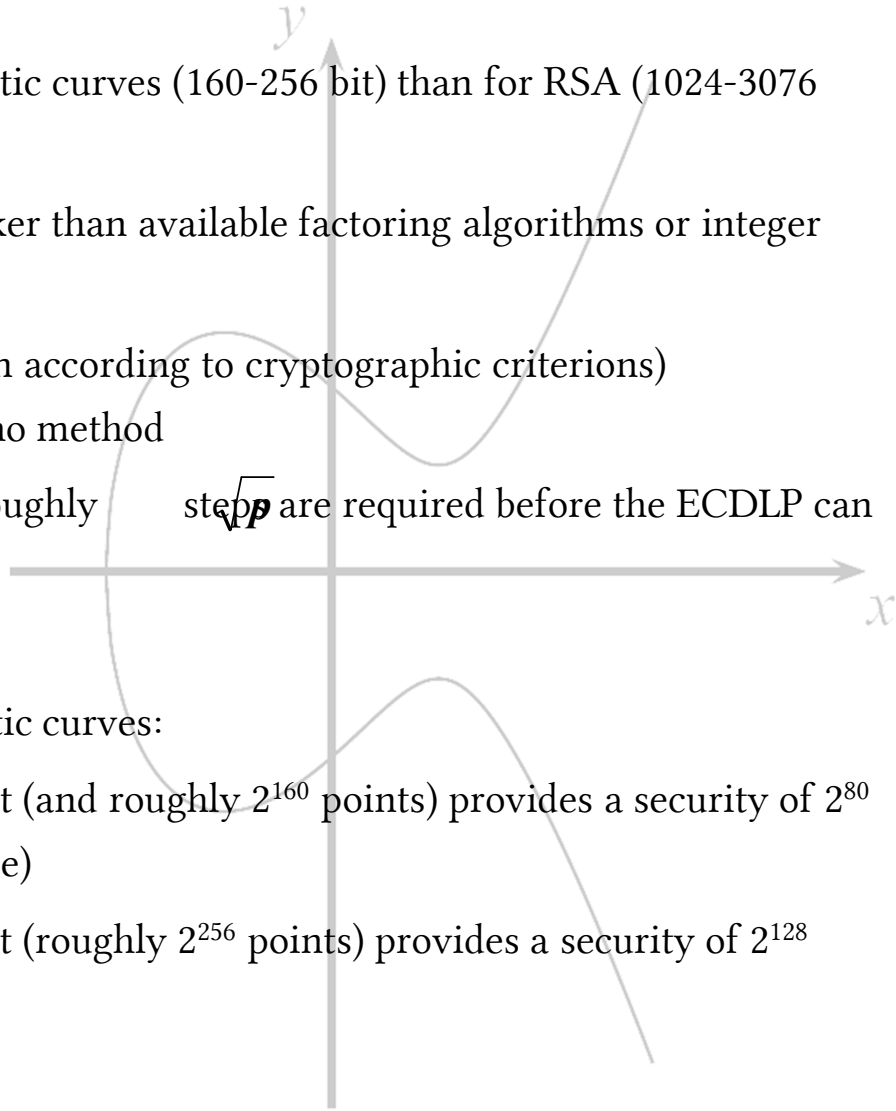
- In some cases, a hash function (see next chapters) is used to derive the session key

Content of this Chapter

- Introduction
- Computations on Elliptic Curves
- The Elliptic Curve Diffie-Hellman Protocol
- **Security Aspects**
- Implementation in Software and Hardware



Security Aspects

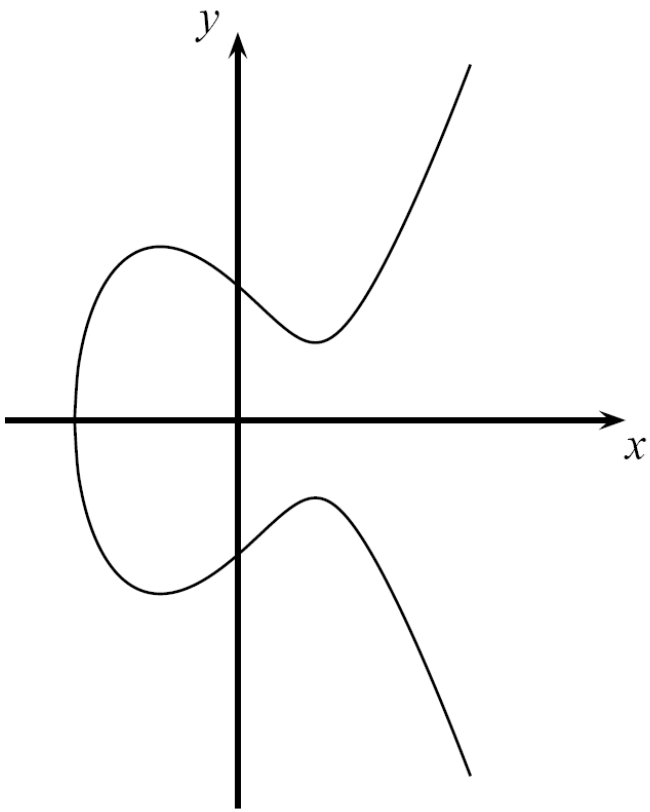


- Why are parameters signficantly smaller for elliptic curves (160-256 bit) than for RSA (1024-3076 bit)?
  - Attacks on groups of elliptic curves are weaker than available factoring algorithms or integer DL attacks
  - Best known attacks on elliptic curves (chosen according to cryptographic criterions) are the Baby-Step Giant-Step and Pollard-Rho method
  - Complexity of these methods: on average, roughly $\sqrt{p}$ steps are required before the ECDLP can be successfully solved

- Implications to practical parameter sizes for elliptic curves:
  - An elliptic curve using a prime p with 160 bit (and roughly 2^{160} points) provides a security of 2^{80} steps that required by an attacker (on average)
  - An elliptic curve using a prime p with 256 bit (roughly 2^{256} points) provides a security of 2^{128} steps on average

Content of this Chapter

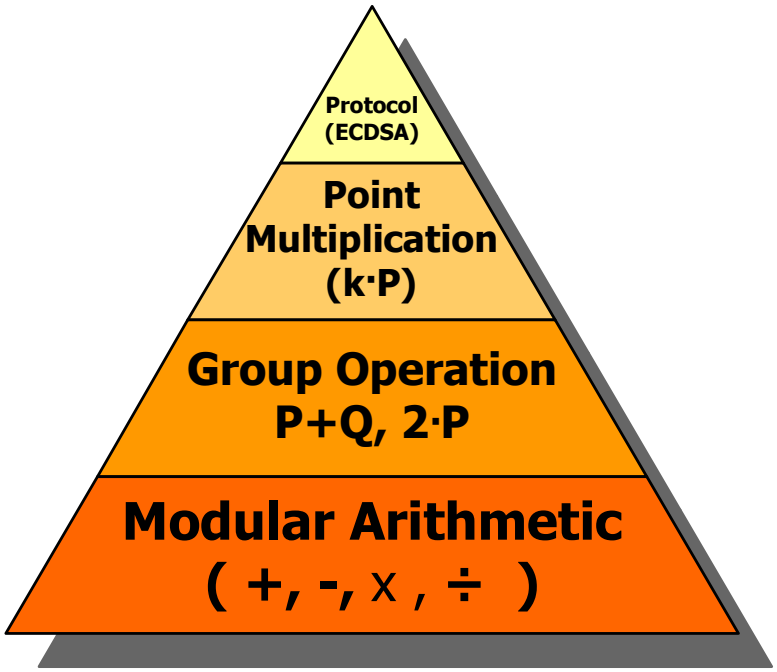
- Introduction
- Computations on Elliptic Curves
- The Elliptic Curve Diffie-Hellman Protocol
- Security Aspects
- **Implementation in Software and Hardware**



Implementations in Hardware and Software

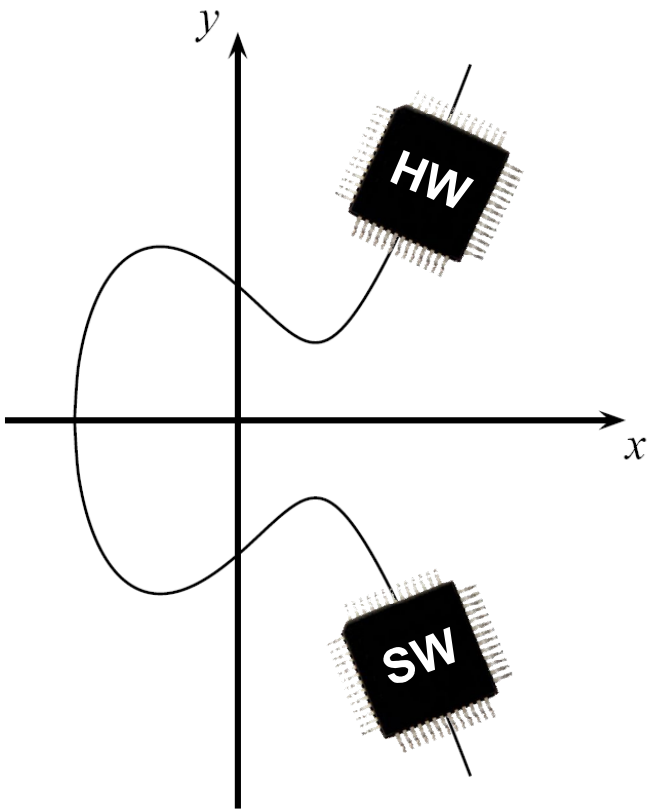
- Elliptic curve computations usually regarded as consisting of four layers:
  - Basic modular arithmetic operations are computationally most expensive
  - Group operation implements point doubling and point addition
  - Point multiplication can be implemented using the Double-and-Add method
  - Upper layer protocols like ECDH and ECDSA

- Most efforts should go in optimizations of the modular arithmetic operations, such as
  - Modular addition and subtraction
  - Modular multiplication
  - Modular inversion



Implementations in Hardware and Software

- Software implementations
  - Optimized 256-bit ECC implementation on 3GHz 64-bit CPU requires about *2 ms* per point multiplication
  - Less powerful microprocessors (e.g, on SmartCards or cell phones) even take significantly longer (>*10 ms*)
- Hardware implementations
  - High-performance implementations with 256-bit special primes can compute a point multiplication in a few hundred microseconds on reconfigurable hardware
  - Dedicated chips for ECC can compute a point multiplication even in a few ten microseconds



Lessons Learned

- Elliptic Curve Cryptography (ECC) is based on the discrete logarithm problem. It requires, for instance, arithmetic modulo a prime.
- ECC can be used for key exchange, for digital signatures and for encryption.
- ECC provides the same level of security as RSA or discrete logarithm systems over Z_p with considerably shorter operands (approximately 160–256 bit vs. 1024–3072 bit), which results in shorter ciphertexts and signatures.
- In many cases ECC has performance advantages over other public-key algorithms.
- ECC is slowly gaining popularity in applications, compared to other public-key schemes, i.e., many new applications, especially on embedded platforms, make use of elliptic curve cryptography.

Content of this Chapter

- The principle of digital signatures
- Security services
- The RSA digital signature scheme
- The Digital Signature Algorithm (DSA)

Content of this Chapter

- **The principle of digital signatures**
- Security services
- The RSA digital signature scheme
- The Digital Signature Algorithm (DSA)

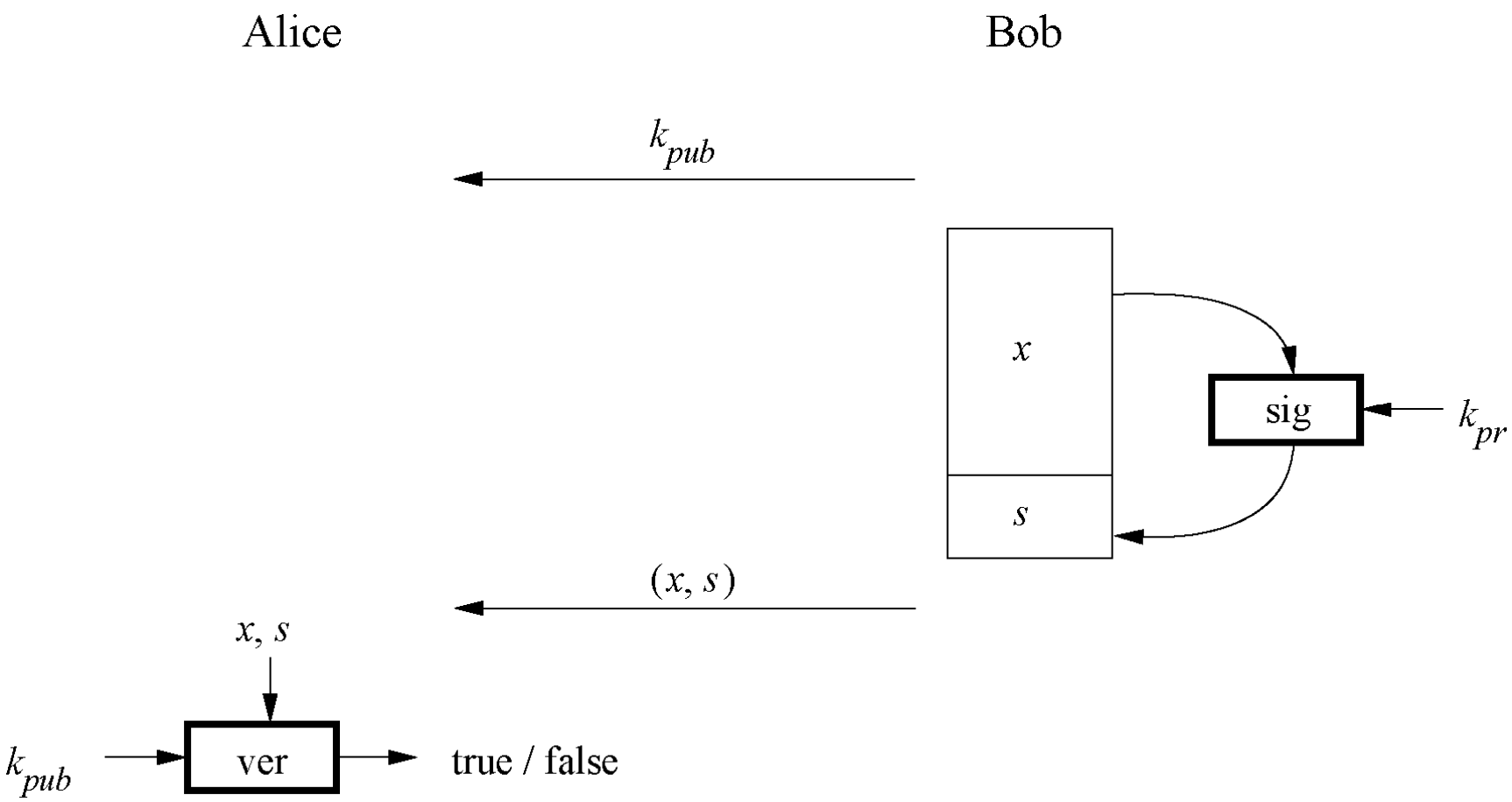
Motivation

- Alice orders a pink car from the car salesmen Bob
- After seeing the pink car, Alice states that she has never ordered it:
- How can Bob prove towards a judge that Alice has ordered a pink car? (And that he did not fabricate the order himself)

$\Rightarrow$ Symmetric cryptography fails because both Alice and Bob can be malicious

$\Rightarrow$ Can be achieved with public-key cryptography

Basic Principle of Digital Signatures



Main idea

- For a given message *x,* a digital signature is appended to the message (just like a conventional signature).
- Only the person with the private key should be able to generate the signature.
- The signature must change for every document.

$\Rightarrow$The signature is realized as a function with the message *x* and the private key as input.

$\Rightarrow$The public key and the message *x* are the inputs to the verification function.

Content of this Chapter

- The principle of digital signatures
- **Security services**
- The RSA digital signature scheme
- The Digital Signature Algorithm (DSA)

Core Security Services

The objectives of a security systems are called *security services*.

1. **Confidentiality***:* Information is kept secret from all but authorized parties.
2. **Integrity:** Ensures that a message has not been modified in transit.
3. **Message Authentication:** Ensures that the sender of a message is authentic. An alternative term is data origin authentication.
4. **Non-repudiation:** Ensures that the sender of a message can not deny the creation of the message. (c.f. order of a pink car)

Additional Security Services

5. **Identification/entity authentication:** Establishing and verification of the identity of an entity, e.g. a person, a computer, or a credit card.
6. **Access control:** Restricting access to the resources to privileged entities.
7. **Availability:** The electronic system is reliably available.
8. **Auditing:** Provides evidences about security relevant activities, e.g., by keeping logs about certain events.
9. **Physical security:** Providing protection against physical tampering and/or responses to physical tampering attempts
10. **Anonymity:** Providing protection against discovery and misuse of identity.

Content of this Chapter

- The principle of digital signatures
- Security services
- **The RSA digital signature scheme**
- The Digital Signature Algorithm (DSA)

Main idea of the RSA signature scheme

To generate the private and public key:

- Use the same key generation as RSA encryption.

To generate the signature:

- "encrypt" the message *x* with the private key

$$s = sig_{K_{priv}}(x) = x^d \bmod n$$

- Append *s* to message *x*

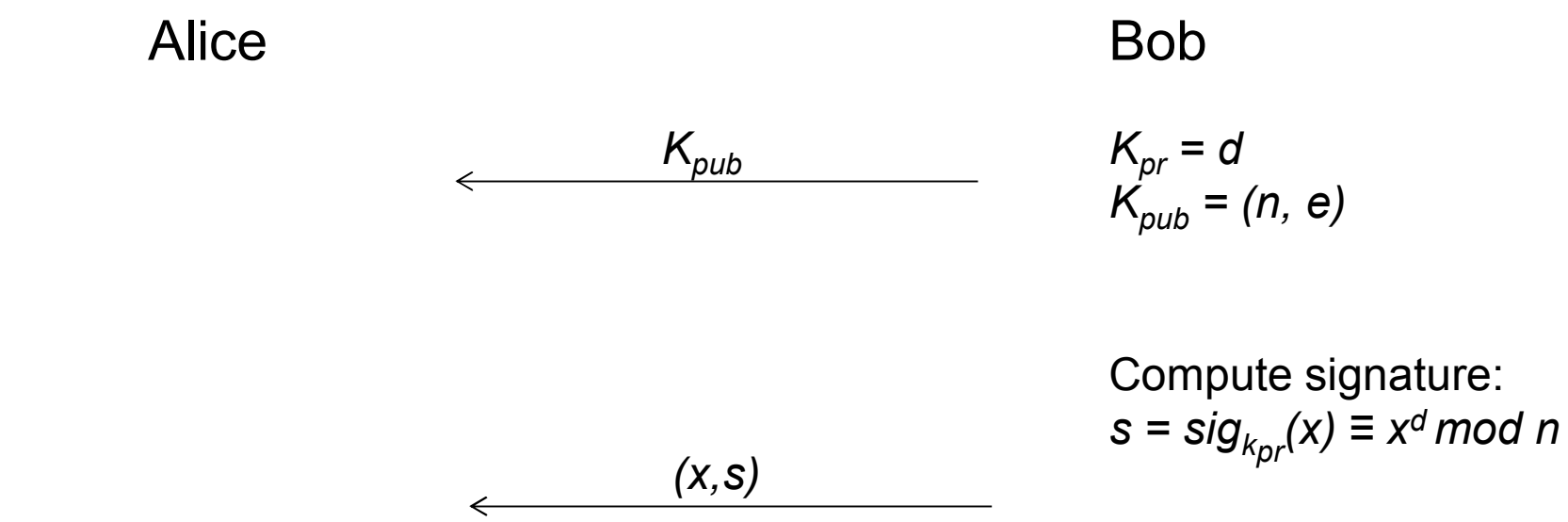
To verify the signature:

- "decrypt" the signature with the public key

$$x' = ver_{K_{pub}}(s) = s^e \bmod n$$

- If $x = x'$, the signature is valid

The RSA Signature Protocol



Security and Performance of the RSA Signature Scheme

Security:

The same constrains as RSA encryption: n needs to be at least 1024 bits to provide a security level of 80 bit.

$\Rightarrow$ The signature, consisting of s, needs to be at least 1024 bits long

Performance:

The signing process is an exponentiation with the private key and the verification process an exponentiation with the public key e.

$\Rightarrow$ Signature verification is very efficient as a small number can be chosen for the public key.

Existential Forgery Attack against RSA Digital Signature

Alice Oscar Bob

$K_{pr} = d$
$K_{pub} = (n, e)$

(n,e) ←

(n,e) ←

1. Choose signature:
   $s \in Z_n$
2. Compute message:
   $x \equiv s^e \bmod n$

(x,s) ←

Verification:
$s^e \equiv x' \bmod n$

since $s^e = (x^d)^e \equiv x \bmod n$
→ Signature is valid

Existential Forgery and Padding

- An attacker can generate valid message-signature pairs (x,s)
- But an attack can only choose the signature s and NOT the message x

$\Rightarrow$ Attacker cannot generate messages like „Transfer $1000 into Oscar‘s account“

Formatting the message x according to a *padding scheme* can be used to make sure that an attacker cannot generate valid (x,s) pairs.

(A messages x generated by an attacker during an Existential Forgery Attack will not coincide with the padding scheme. For more details see Chapter 10 in *Understanding Cryptography.)*

Content of this Chapter

- The principle of digital signatures
- Security services
- The RSA digital signature scheme
- **The Digital Signature Algorithm (DSA)**

Facts about the Digital Signature Algorithm (DSA)

- Federal US Government standard for digital signatures (DSS)
- Proposed by the National Institute of Standards and Technology (NIST)
- DSA is based on the Elgamal signature scheme
- Signature is only 320 bits long
- Signature verification is slower compared to RSA

The Digital Signature Algorithm (DSA)

Key generation of DSA:

1. Generate a prime p with $2^{1023} < p < 2^{1024}$
2. Find a prime divisor q of $p-1$ with $2^{159} < q < 2^{160}$
3. Find an integer α with $\text{ord}(\alpha)=q$
4. Choose a random integer d with $0<d<q$
5. Compute $\beta \equiv \alpha^d \bmod p$

The keys are:

$k_{pub} = (p,q,\alpha,\beta)$

$k_{pr} = (d)$

The Digital Signature Algorithm (DSA)

DSA signature generation :

Given: message x, signature s, private key d and public key (p,q,α,β)

1. Choose an integer as random ephemeral key k_E with $0<k_E<q$
2. Compute $r \equiv (\alpha^{kE} \bmod p) \bmod q$
3. Computes $s \equiv (\mathrm{SHA}(x)+d \cdot r)\, k_E^{-1} \bmod q$

The signature consists of (r,s)

SHA denotes the hashfunction SHA-1 which computes a 160-bit fingerprint of message x. (See Chapter 11 of *Understanding Cryptography* for more details)

The Digital Signature Algorithm (DSA)

DSA signature verification

Given: message x, signature s and public key (p,q,α,β)

1. Compute auxiliary value $w \equiv s^{-1} \bmod q$
2. Compute auxiliary value $u_1 \equiv w \cdot SHA(x) \bmod q$
3. Compute auxiliary value $u_2 \equiv w \cdot r \bmod q$
4. Compute $v \equiv (\alpha^{u1} \cdot \beta^{u2} \bmod p) \bmod q$

If $v \equiv r \bmod q \rightarrow$ signature is valid

If $v \not\equiv r \bmod q \rightarrow$ signature is invalid

Proof of DSA:

We show need to show that the signature (*r,s*) in fact satisfied the condition $r \equiv v \bmod q$:

$$s \equiv (\mathrm{SHA}(x)) + d \cdot r) \cdot k_E^{-1} \bmod q$$

$$\Leftrightarrow \quad k_E \equiv s^{-1}\,\mathrm{SHA}(x) + d \cdot s^{-1}\, r \bmod q$$

$$\Leftrightarrow \quad k_E \equiv u_1 + d \cdot u_2 \bmod q$$

We can raise α to either side of the equation if we reduce modulo *p*:

$$\Leftrightarrow \quad \alpha^{kE} \bmod p \equiv \alpha^{u1+d \cdot u2} \bmod p$$

Since $\beta \equiv \alpha^d \bmod p$ we can write:

$$\Leftrightarrow \quad \alpha^{kE} \bmod p \equiv \alpha^{u1}\,\beta^{u2} \bmod p$$

We now reduce both sides of the equation modulo q:

$$\Leftrightarrow \quad (\alpha^{kE} \bmod p) \bmod q \equiv (\alpha^{u1}\,\beta^{u2} \bmod p) \bmod q$$

Since $r \equiv \alpha^{kE} \bmod p \bmod q$ and $v \equiv (\alpha^{u1}\,\beta^{u2} \bmod p) \bmod q$, this expression is identical to:

$$\Leftrightarrow \quad r \equiv v$$

Example

Alice

Bob

Key generation:
1. choose $p = 59$ and $q = 29$
2. choose $\alpha = 3$
3. choose private key $d = 7$
4. $\beta = \alpha^{\beta} = 3^7 \equiv 4 \bmod 59$

$(p, q, \alpha, \beta)=(59, 29, 3, 4)$ ←

Sign:
Compute has of message $H(x)=26$
1. Choose ephermal key $k_E=10$
2. $r = (3^{10} \bmod 59) \equiv 20 \bmod 29$
3. $s = (26 + 7 \cdot 20) \cdot 3) \equiv 5 \bmod 29$

$(x,(r, s))=(x,20, 5)$ ←

Verify:
$w \equiv 5^{-1} \equiv 6 \bmod 29$
$u_1 \equiv 6 \cdot 26 \equiv 11 \bmod 29$
$u_2 \equiv 6 \cdot 20 \equiv 4 \bmod 29$
$v = (3^{11} \cdot 4^4 \bmod 59) \bmod 29 = 20$
$v \equiv$ r mod 29 → valid signature

Security of DSA

To solve the discrete logarithm problem in p the powerful index calculus method can be applied. But this method cannot be applied to the discrete logarithm problem of the subgroup q. Therefore q can be smaller than p. For details see Chapter 10 and Chapter 8 of *Understanding Cryptography* .

p	q	hash output (min)	security levels
1024	160	160	80
2048	224	224	112
3072	256	256	128

Standardized parameter bit lengths and security levels for the DSA

Elliptic Curve Digital Signature Algorithm (ECDSA)

- Based on Elliptic Curve Cryptography (ECC)
- Bit lengths in the range of 160-256 bits can be chosen to provide security equivalent to 1024-3072 bit RSA (80-128 bit symmetric security level)
- One signature consists of two points, hence the signature is twice the used bit length (i.e., 320-512 bits for 80-128 bit security level).
- The shorter bit length of ECDSA often result in shorter processing time

For more details see Section 10.5 in *Understanding Cryptography*

Lessons Learned

- Digital signatures provide message integrity, message authentication and non-repudiation.
- RSA is currently the most widely used digital signature algorithm.
- Competitors are the Digital Signature Standard (DSA) and the Elliptic Curve Digital Signature Standard (ECDSA).
- RSA verification can be done with short public keys e. Hence, in practice, RSA verification is usually faster than signing.
- DSA and ECDSA have shorter signatures than RSA
- In order to prevent certain attacks, RSA should be used with padding.
- The modulus of DSA and the RSA signature schemes should be at least 1024- bits long. For true long-term security, a modulus of length 3072 bits should be chosen. In contrast, ECDSA achieves the same security levels with bit lengths in the range 160–256 bits.

Fig.. 1.2 Absorbing and squeezing phase of Keccak

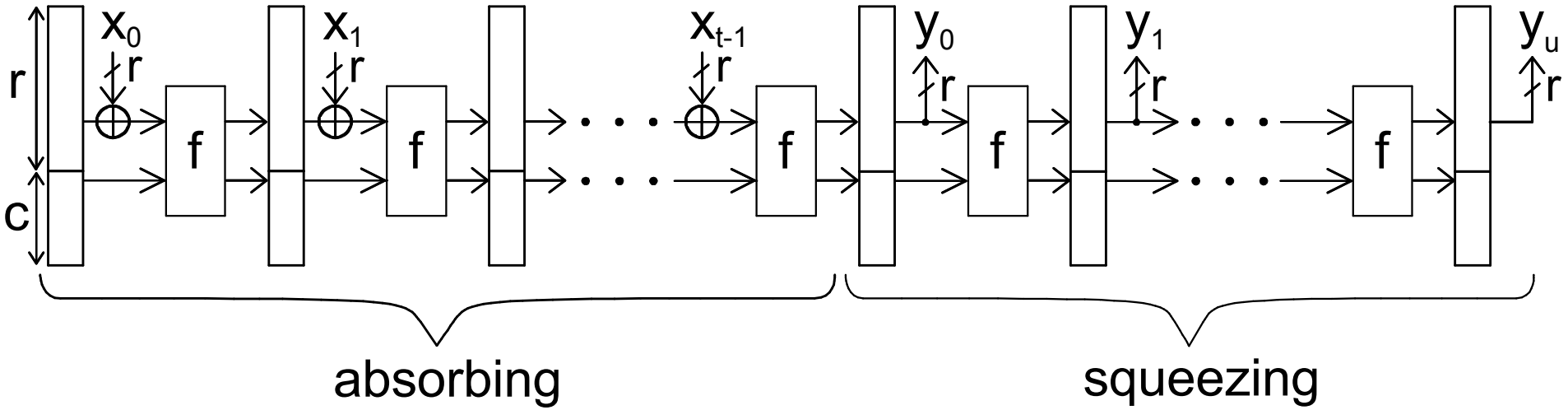


Fig. 1.3 The internal structure of Keccak

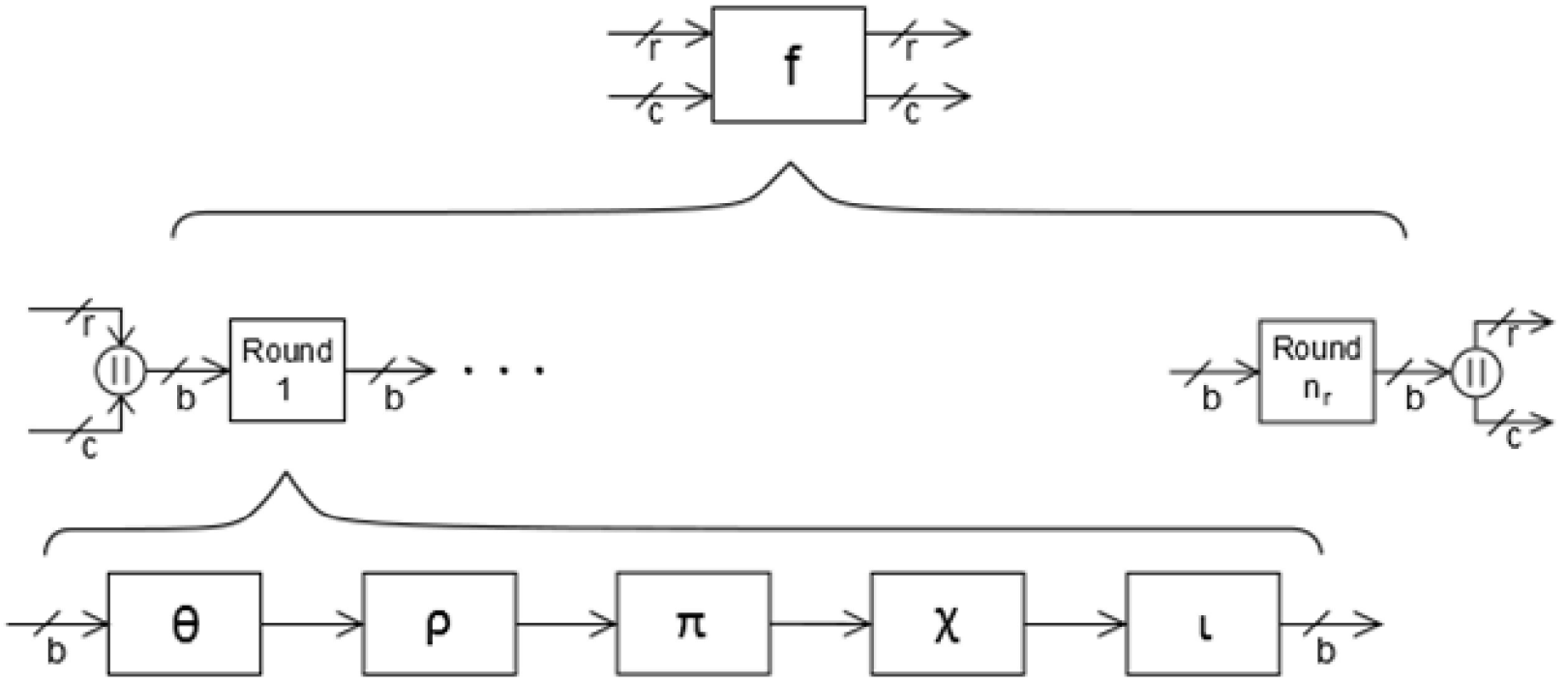


Fig. 1.4 The state of Keccak

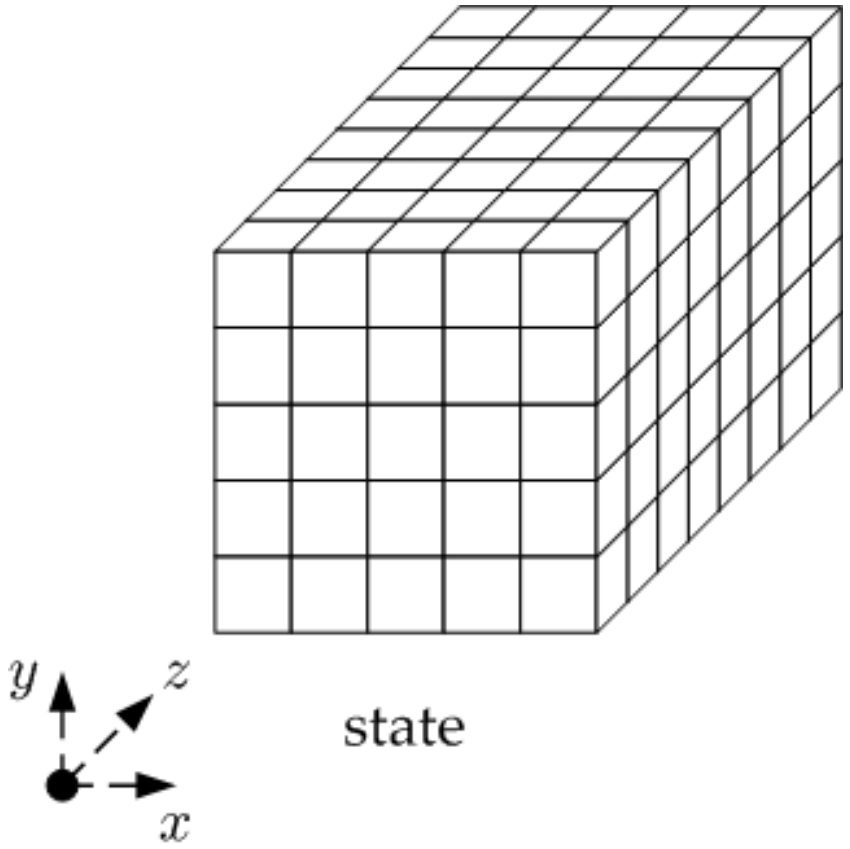


Fig. 1.5 The Theta Step of Keccak – visually

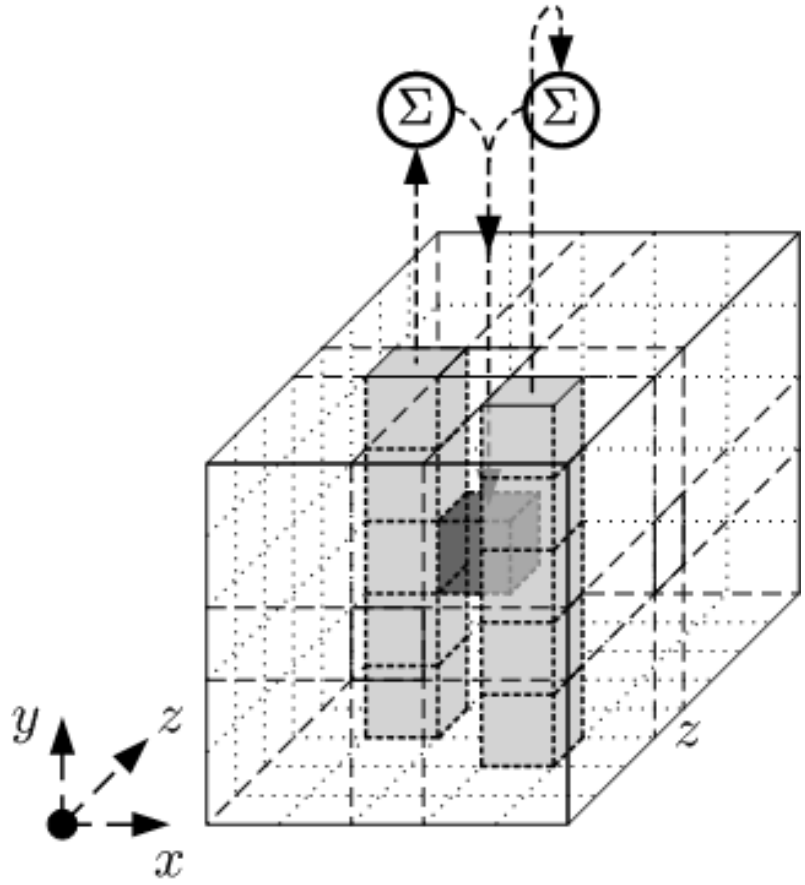


Fig. 1.5 The Theta Step of Keccak – pseudo code

- Input: state array A[x,y]
- Output: manipulated state array A[x,y]

- $C[x] = A[x,0] \oplus A[x,1] \oplus A[x,2] \oplus A[x,3] \oplus A[x,4]$ $\quad x = 0\ldots4$
- $D[x] = C[x-1] \oplus \text{rot}(C[x+1],1)$ $\quad x = 0\ldots4$
- $A[x,y] = A[x,y] \oplus D[x]$ $\quad x,y = 0\ldots4$

Table 1.3 The rotation constants of Keccak

	x = 3	x = 4	x = 0	x = 1	x = 2
y=2	25	39	3	10	43
y=1	55	20	36	44	6
y=0	28	27	0	1	62
y=4	56	14	18	2	61
y=3	21	8	41	45	15

Fig. 1.6 The Chi Step of Keccak

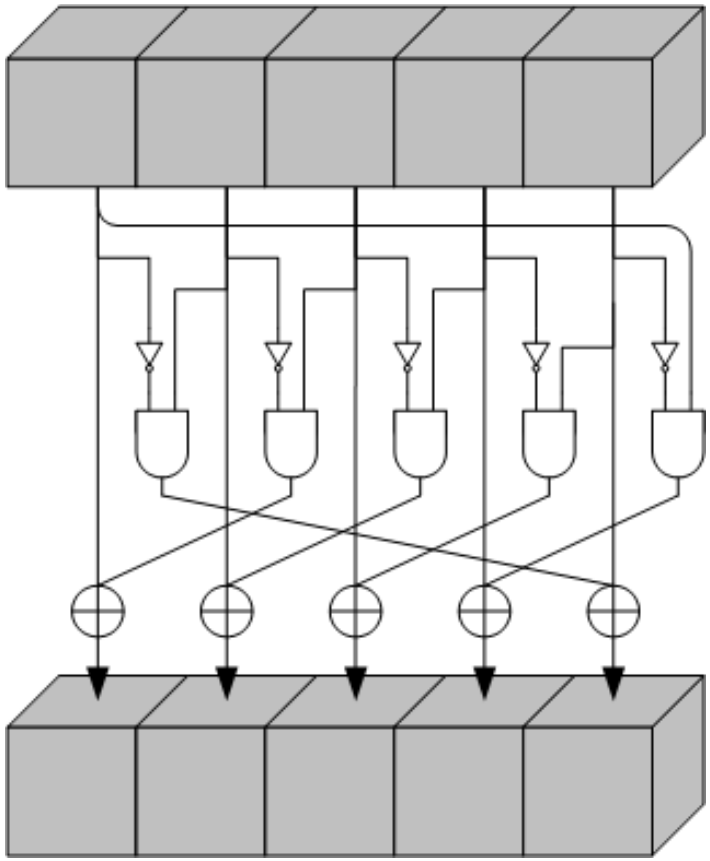


Table 1.4 The round constants of Keccak

RC[0] = 0x0000000000000001	RC[12] = 0x000000008000808B
RC[1] = 0x0000000000008082	RC[13] = 0x800000000000008B
RC[2] = 0x800000000000808A	RC[14] = 0x8000000000008089
RC[3] = 0x8000000080008000	RC[15] = 0x8000000000008003
RC[4] = 0x000000000000808B	RC[16] = 0x8000000000008002
RC[5] = 0x0000000080000001	RC[17] = 0x8000000000000080
RC[6] = 0x8000000080008081	RC[18] = 0x000000000000800A
RC[7] = 0x8000000000008009	RC[19] = 0x800000008000000A
RC[8] = 0x000000000000008A	RC[20] = 0x8000000080008081
RC[9] = 0x0000000000000088	RC[21] = 0x8000000000008080
RC[10] = 0x0000000080008009	RC[22] = 0x0000000080000001
RC[11] = 0x000000008000000A	RC[23] = 0x8000000080008008

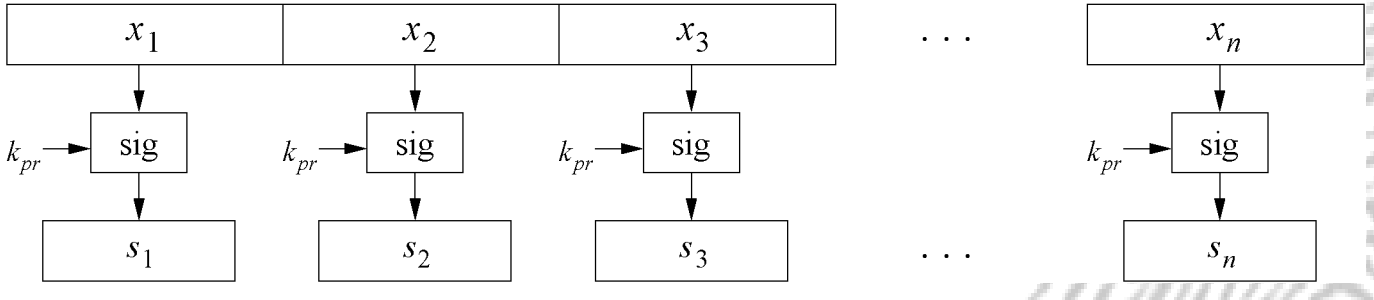
Content of this Chapter

- **Why we need hash functions**
- How does it work
- Security properties
- Algorithms
- Example: The Secure Hash Algorithm SHA-1

Motivation

Problem:

Naive signing of long messages generates a signature of same length.



- Three Problems
- Computational overhead
- Message overhead
- Security limitations
- For more info see Section 11.1 in "*Understanding Cryptography*".

Solution:

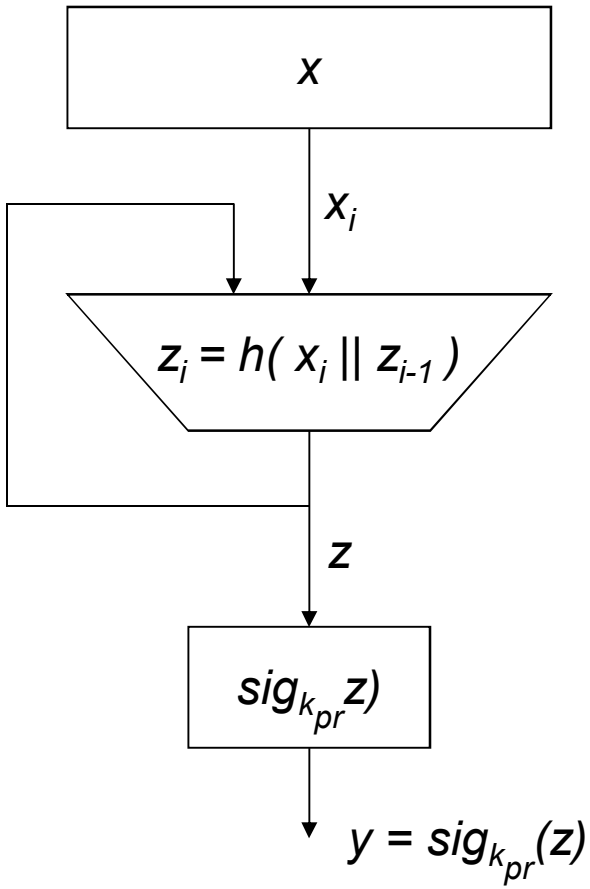
Instead of signing the whole message, sign only a digest (=hash)

Also secure, but much faster

Needed:

Hash Functions

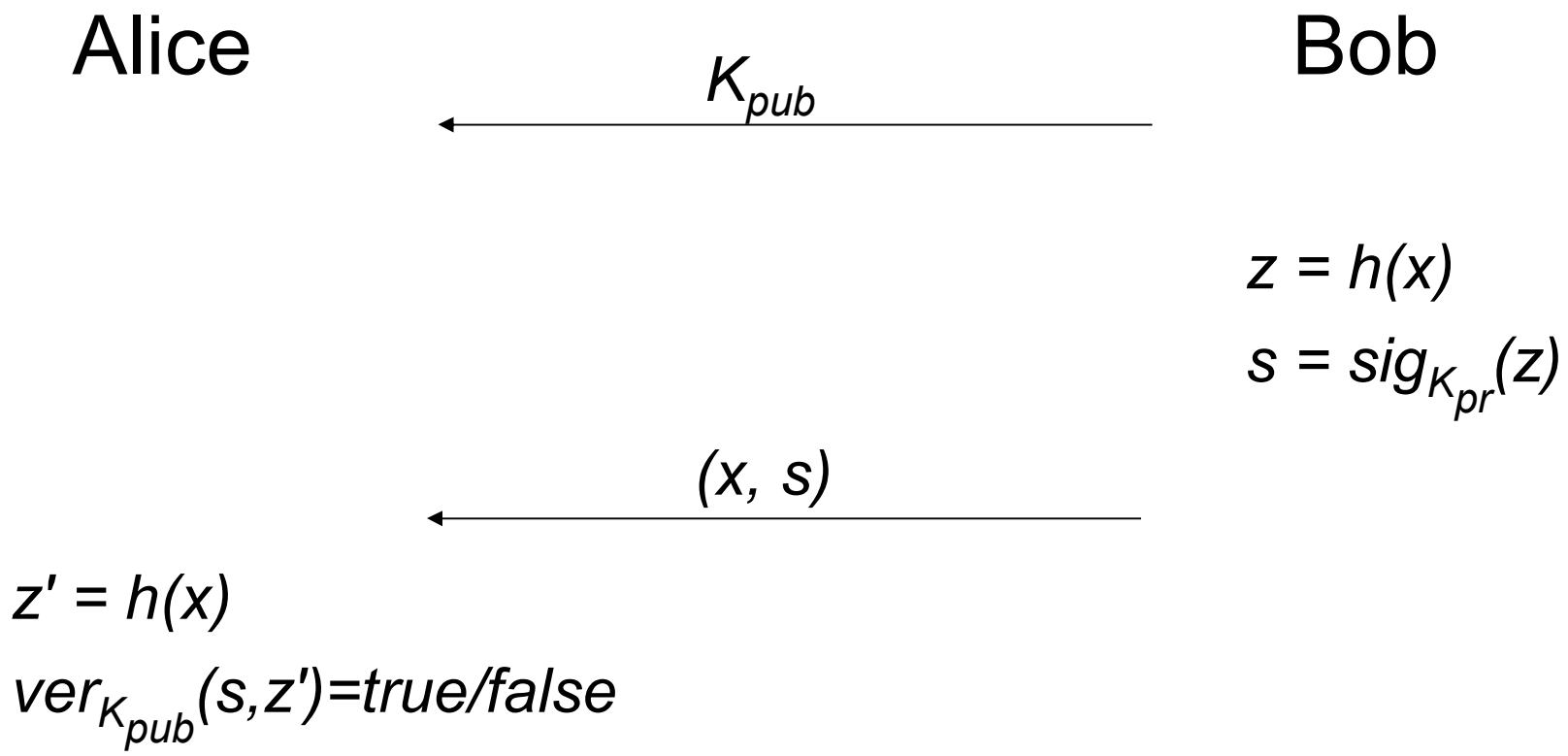
Digital Signature with a Hash Function



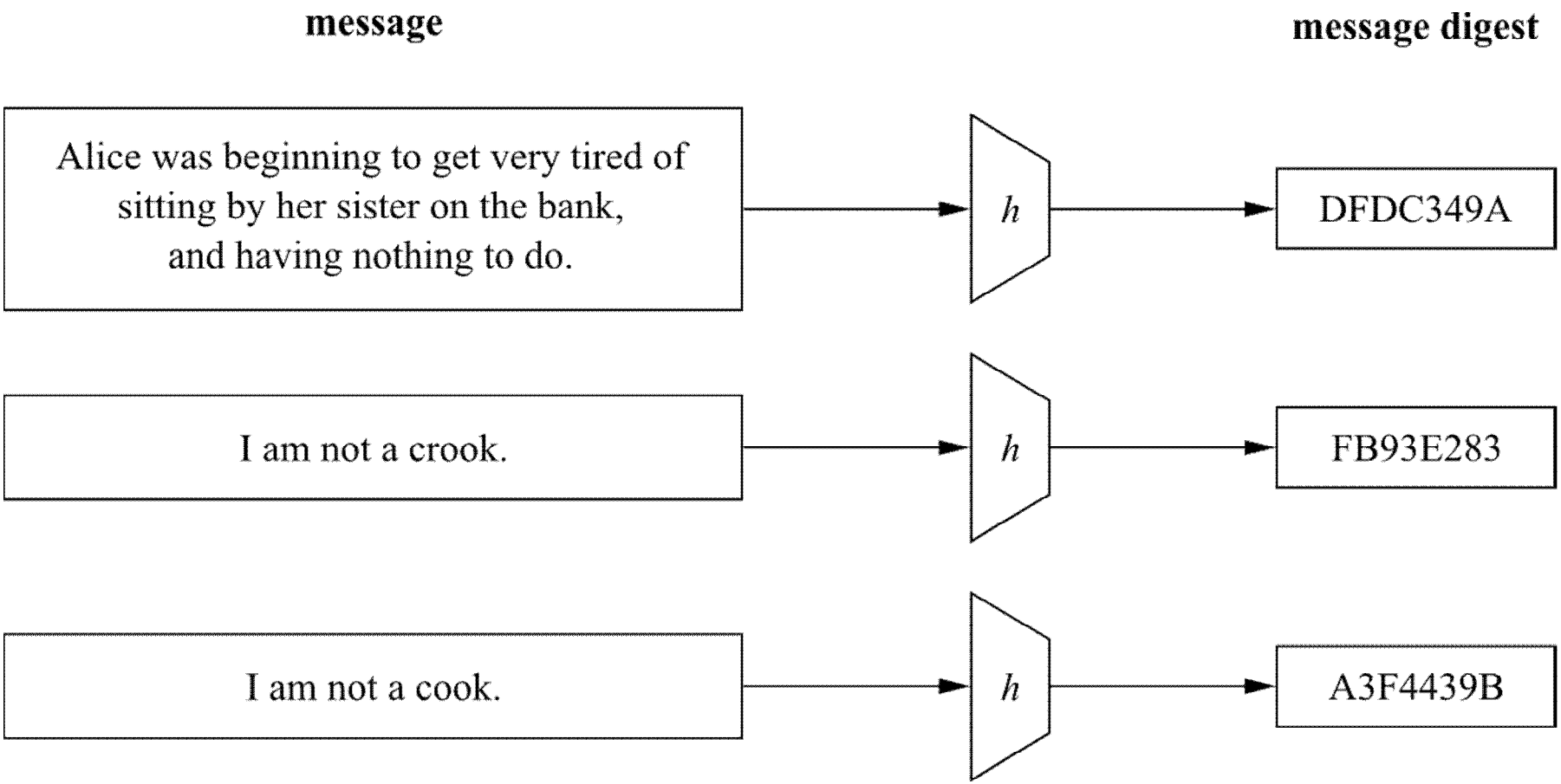
Notes:

- *x* has fixed length
- *z, y* have fixed length
- *z, x* do not have equal length in general
- *h(x)* does not require a key.
- *h(x)* is public.

- **Basic Protocol for Digital Signatures with a Hash Function:**



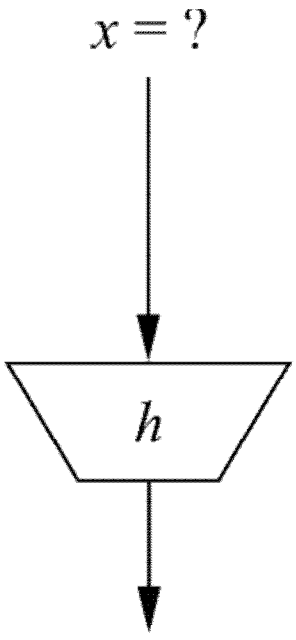
Principal input–output behavior of hash functions



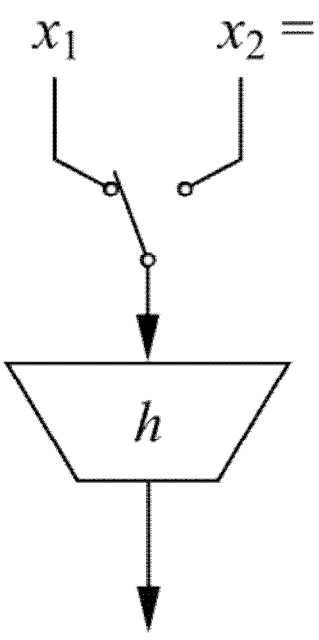
Content of this Chapter

- Why we need hash functions
- How does it work
- **Security properties**
- Algorithms
- Example: The Secure Hash Algorithm SHA-1

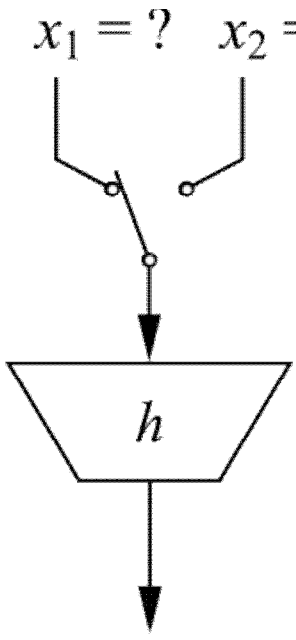
The three security properties of hash functions



preimage resistance



second preimage resistance



collision resistance

Hash Funktionen: Security Properties

- **Preimage resistance:** For a given output z, it is impossible to find any input x such that h(x) = z, i.e., h(x) is one-way.
- **Second preimage resistance:** Given x_1, and thus $h(x_1)$, it is computationally infeasible to find any x_2 such that $h(x_1) = h(x_2)$.
- **Collision resistance:** It is computationally infeasible to find any pairs $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$.

Hash Funktionen: Security

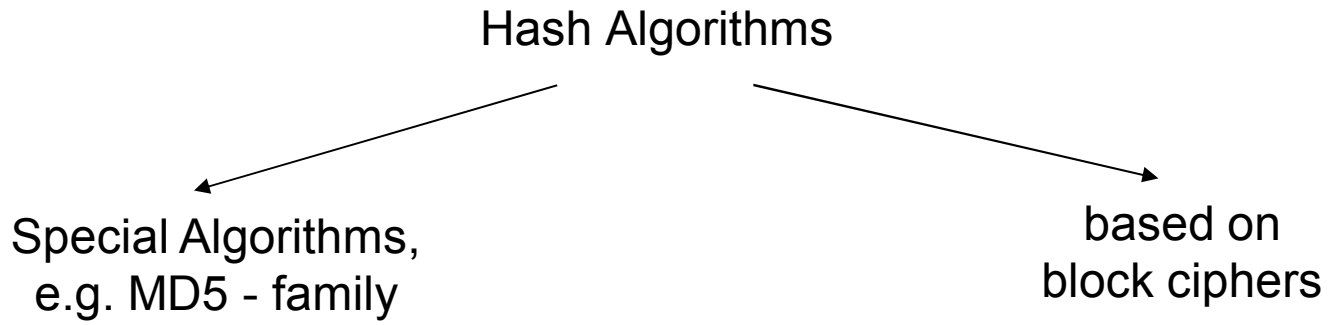
It turns out that collison resistance causes most problems

- How hard is it to find a collision with a probability of 0.5 ?
- Related Problem: How many people are needed such that two of them have the same birthday with a probability of 0.5 ?
- No! Not 365/2=183. 23 are enough ! This is called the birthday paradoxon (Search takes $\approx\sqrt{2^n}$ steps) .
- For more info see Chapter 11.2.3 in *Understanding Cryptography*.
- To deal with this paradox, hash functions need a output size of at least 160 bits.

Content of this Chapter

- Why we need hash functions
- How does it work
- Security properties
- **Algorithms**
- Example: The Secure Hash Algorithm SHA-1

Hash Funktionen: Algorithms



- **MD5 -** family
- **SHA-1**: output - 160 Bit; input - 512 bit chunks of message *x;*

  operations - bitwise AND, OR, XOR, complement und cyclic shifts.
- **RIPE-MD 160:** output - 160 Bit; input - 512 bit chunks of message *x*; operations – like in SHA-1, but two in parallel and combinations of them after each round.

Content of this Chapter

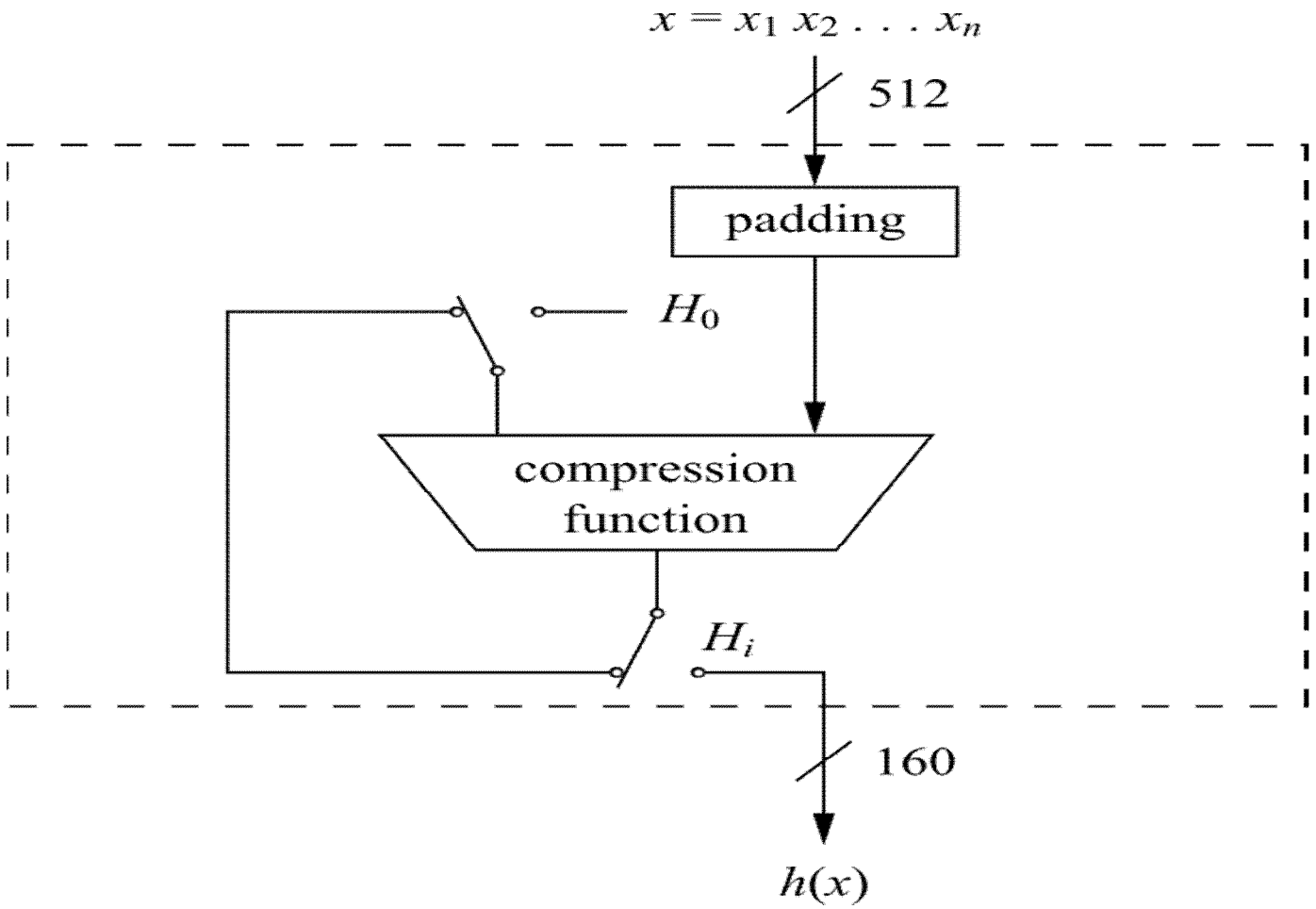
- Why we need hash functions
- How does it work
- Security properties
- Algorithms
- **Example: The Secure Hash Algorithm SHA-1**

SHA-1

- Part of the MD-4 family.
- Based on a Merkle-Dåmgard construction.
- 160-bit output from a message of maximum length 2^{64} bit.
- Widely used (even tough some weaknesses are known)

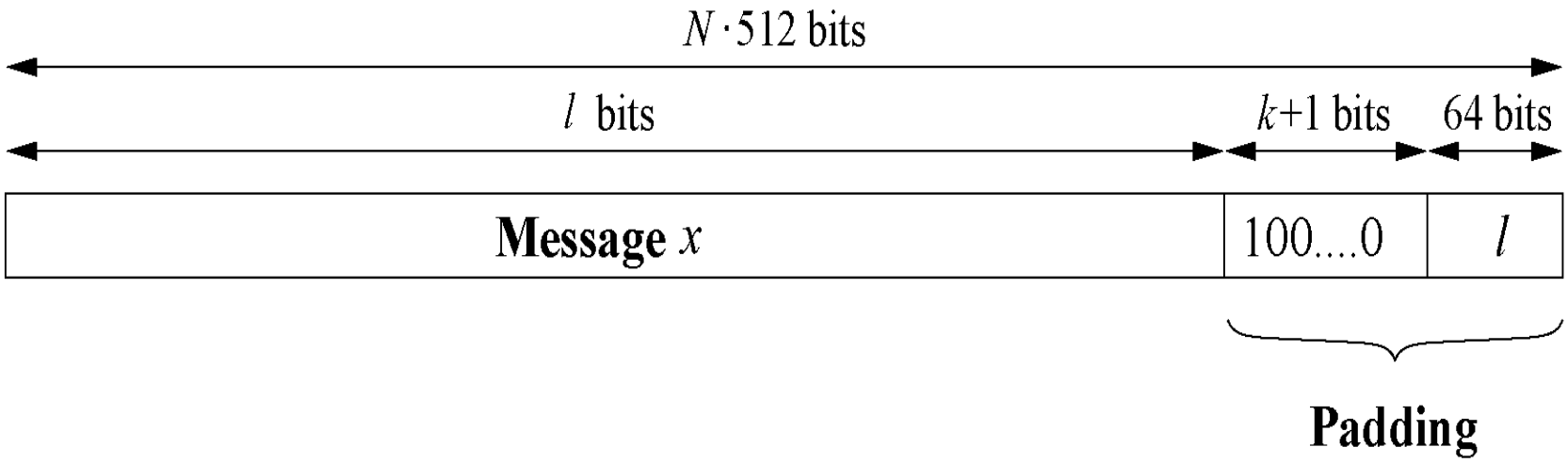
SHA-1 High Level Diagramm

- Compression Function consists of 80 rounds which are divided into four stages of 20 rounds each



SHA-1: Padding

- Message x has to be padded to fit a size of a multiple of 512 bit.
- $k \equiv 512 - 64 - 1 - l = 448 - (l + 1) \bmod 512$.



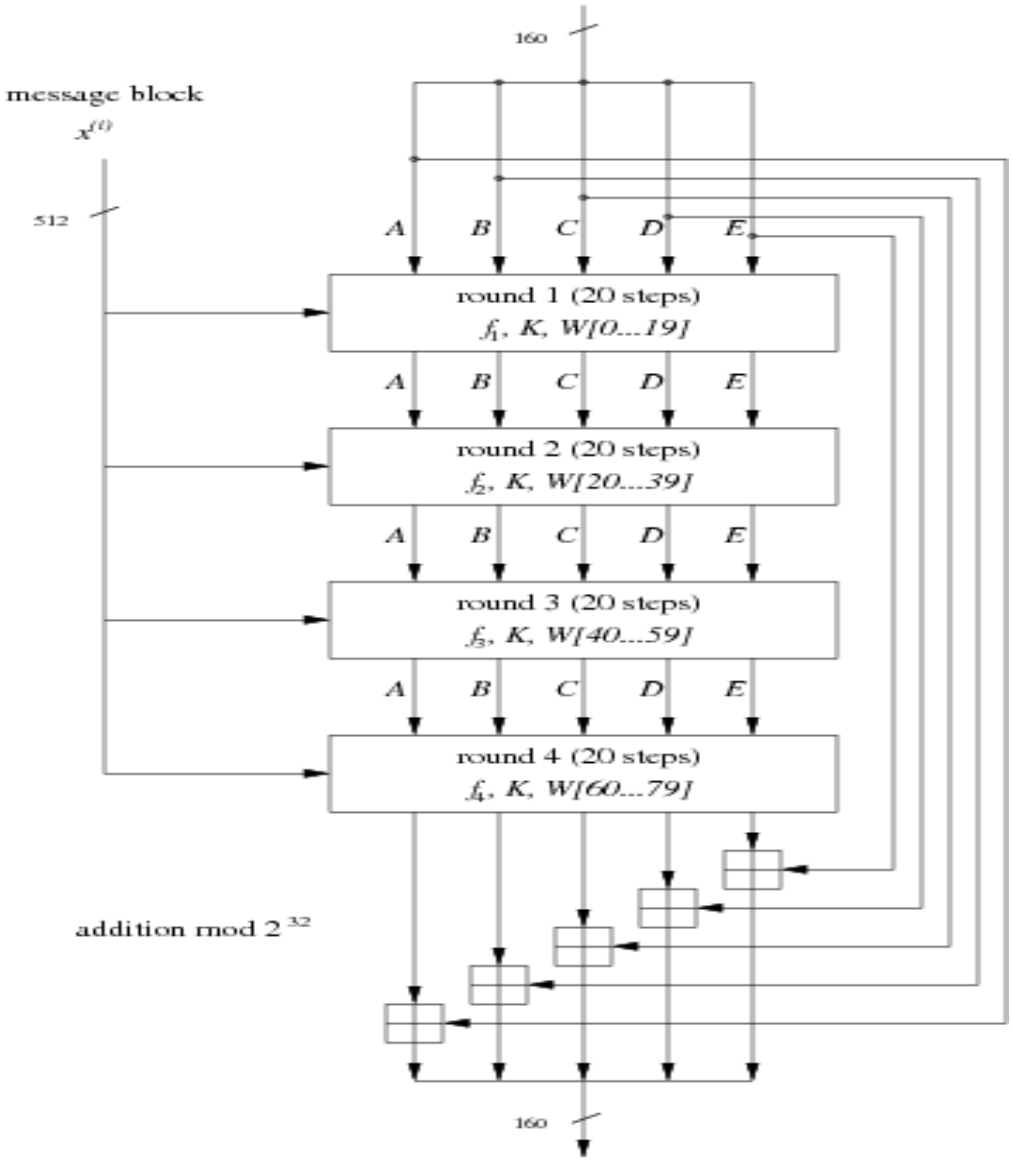
SHA-1: Hash Computation

- Each message block x_i is processed in four stages with 20 rounds each

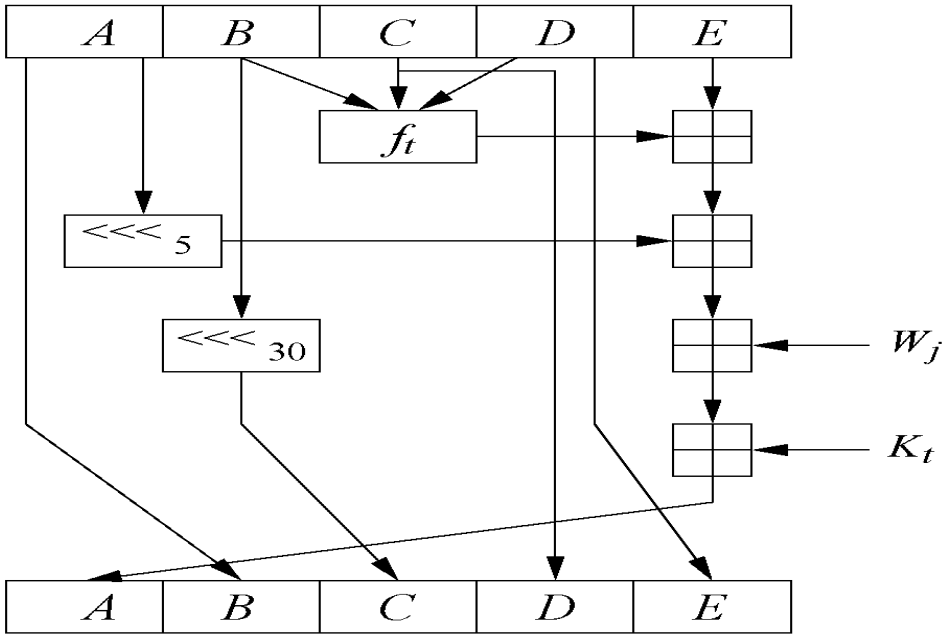
SHA-1 uses:

- A message schedule which computes a 32-bit word W0,W1,...,W79 for each of the 80 rounds
- Five working registers of size of 32 bits A,B,C,D,E
- A hash value H_i consisting of five 32-bit words $H_i^{(0)}$, $H_i^{(1)}$, $H_i^{(2)}$, $H_i^{(3)}$, $H_i^{(4)}$
- In the beginning, the hash value holds the initial value H_0, which is replaced by a new hash value after the processing of each single message block.
- The final hash value H_n is equal to the output h(x) of SHA-1.

SHA-1: All four stages



SHA-1: Internals of a Round



Stage t	Round j	Constant K_t	Function f_t
1	00…19	K=5A827999	f(B,C,D)=(B∧C)∨(¯B∧D)
2	20…39	K=6ED9EBA1	f(B,C,D)=B⊕C⊕D
3	40…59	K=8F1BBCDC	f(B,C,D)=(B⊕C)∨(B⊕D)∨(C⊕D)
4	60…79	K=CA62C1D6	f(B,C,D)=B⊕C⊕D

Lessons Learned: **Hash-Funktionen**

- Hash functions are keyless. The two most important applications of hash functions are their use in digital signatures and in message authentication codes such as HMAC.
- The three security requirements for hash functions are one-wayness, second preimage resistance and collision resistance.
- Hash functions should have at least 160-bit output length in order to withstand collision attacks; 256 bit or more is desirable for long-term security.
- MD5, which was widely used, is insecure. Serious security weaknesses have been found in SHA-1, and the hash function should be phased out. The SHA-2 algorithms all appear to be secure.
- The ongoing SHA-3 competition will result in new standardized hash functions in a few years.

Further Informations: **Hash-Funktionen**

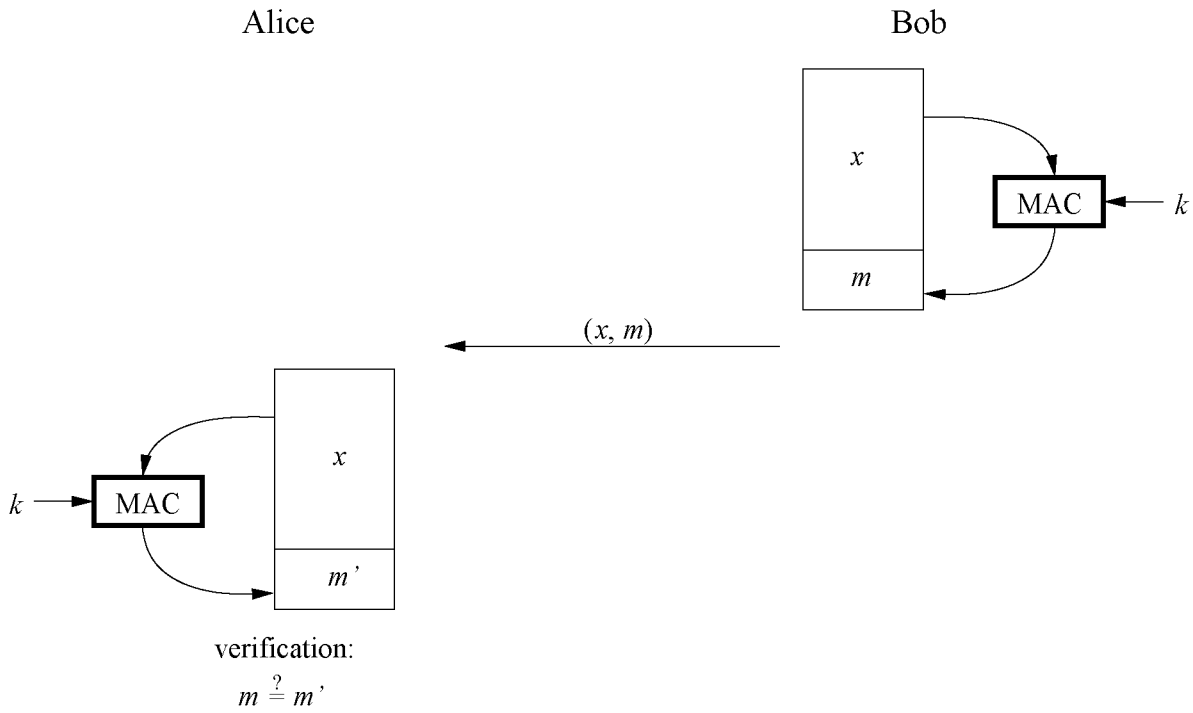
- Overview over many Hash Functions with Spezifications:
  - http://ehash.iaik.tugraz.at/wiki/The_Hash_Function_Zoo
- Birthday Paradox: Wikipedia has a nice explanation
  - http://en.wikipedia.org/wiki/Birthday_problem
- SHA Standards
  - SHA1+2: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf
  - SHA3 Overview: http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo
- CrypTool is a learning program which also can hash:
  - http://www.cryptool.org/

Content of this Chapter

- The principle behind MACs
- The security properties that can be achieved with MACs
- How MACs can be realized with hash functions and with block ciphers

Principle of Message Authentication Codes

- Similar to digital signatures, MACs append an authentication tag to a message
- MACs use a symmetric key k for generation and verification
- Computation of a MAC: $m = \text{MAC}_k(x)$



Properties of Message Authentication Codes

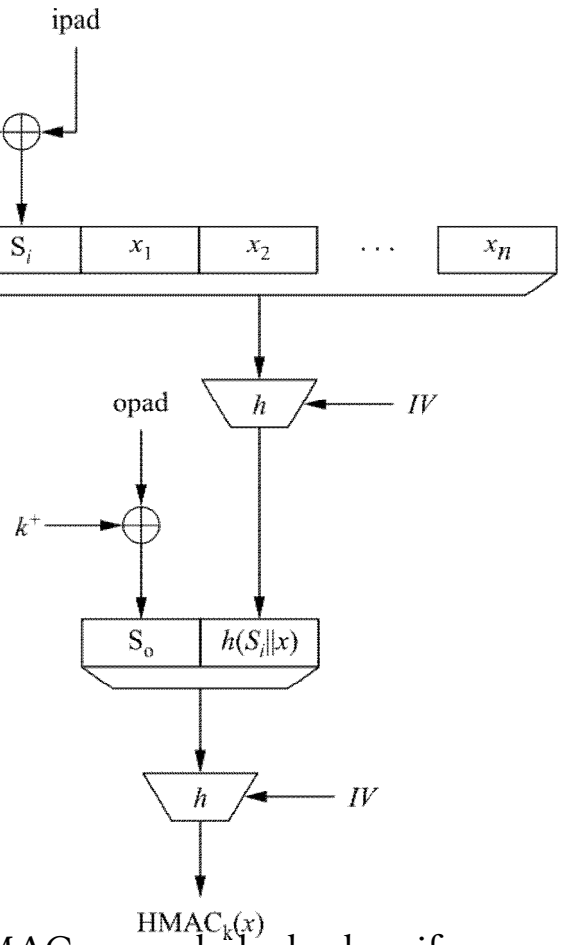
1. **Cryptographic checksum**
   A MAC generates a cryptographically secure authentication tag for a given message.
2. **Symmetric**
   MACs are based on secret symmetric keys. The signing and verifying parties must share a secret key.
3. **Arbitrary message size**
   MACs accept messages of arbitrary length.
4. **Fixed output length**
   MACs generate fixed-size authentication tags.
5. **Message integrity**
   MACs providemessage integrity: Any manipulations of a message during transit will be detected by the receiver.
6. **Message authentication**
   The receiving party is assured of the origin of the message.
7. **No nonrepudiation**
   Since MACs are based on symmetric principles, they do not provide nonrepudiation.

MACs from Hash Functions

- MAC is realized with cryptographic hash functions (e.g., SHA-1)
- HMAC is such a MAC built from hash functions
- Basic idea: Key is hashed together with the message
- Two possible constructions:
  - secret prefix MAC: $m = \text{MAC}_k(x) = h(k||x)$
  - secret suffix MAC: $m = \text{MAC}_k(x) = h(x||k)$
- Attacks:
  - secret prefix MAC: Attack MAC for the message $x = (x_1, x_2, \ldots, x_n, x_{n+1})$, *where* x_{n+1} is an arbitrary additional block, can be constructed from m *without knowing the* secret key
  - secret suffix MAC: find collision x and x_0 such that $h(x) = h(x_0)$, then $m = h(x||k) = h(x_O||k)$
- Idea: Combine secret prefix and suffix: HMAC (cf. next slide)

HMAC

- Proposed by Mihir Bellare, Ran Canetti and Hugo Krawczyk in 1996
- Scheme consists of an inner and outer hash
  - k^+ is expanded key k
  - expanded key k^+ is XORed with the inner pad
  - ipad = 00110110,00110110, . . .,00110110
  - opad = 01011100,01011100, . . .,01011100
  - $\text{HMAC}_k(x) = h[(k^+ \oplus \text{opad}) // h[(k^+ \oplus \text{ipad}) // x]]$

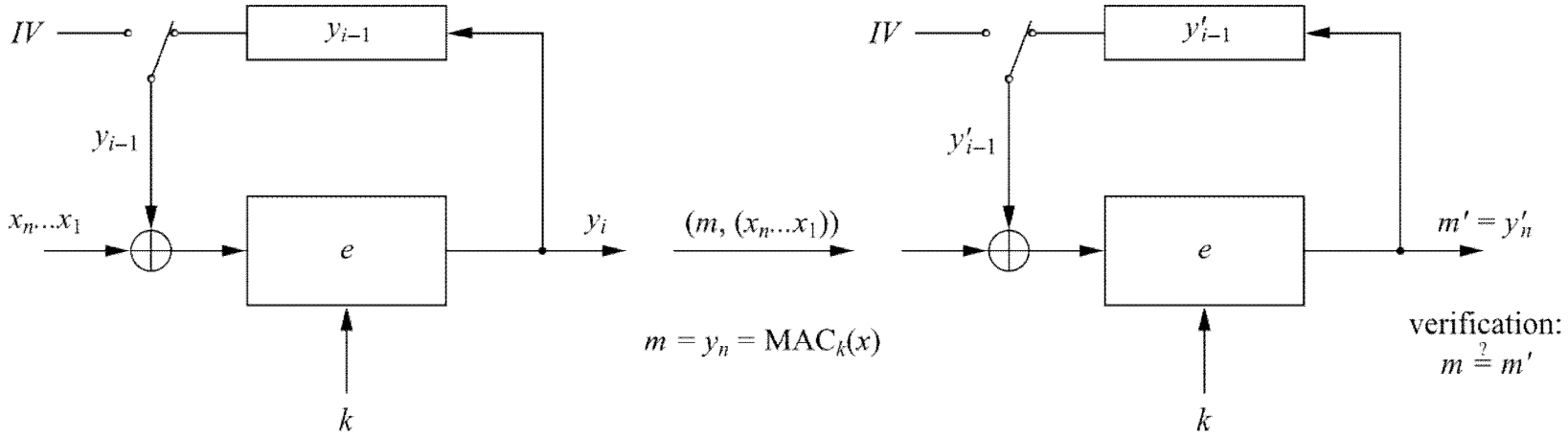


- HMAC is provable secure which means (informally speaking): The MAC can only be broken if a collision for the hash function can be found.

MACs from Block Ciphers

- MAC constructed from block ciphers (e.g. AES)
- Popular: Use AES in CBC mode

- CBC-MAC:



CBC-MAC

- MAC Generation
  - Divide the message x into blocks x_i
  - Compute first iteration $y_1 = e_k(x_1 \oplus IV)$
  - Compute $y_i = e_k(x_i \oplus y_{i-1})$ for the next blocks
  - Final block is the MAC value: $m = \text{MAC}_k(x) = y_n$

- MAC Verification
  - Repeat MAC computation (m')
  - Compare results:In case $m' = m$, the message is verified as correct
  - In case $m' \neq m$, the message and/or the MAC value m have been altered during transmission

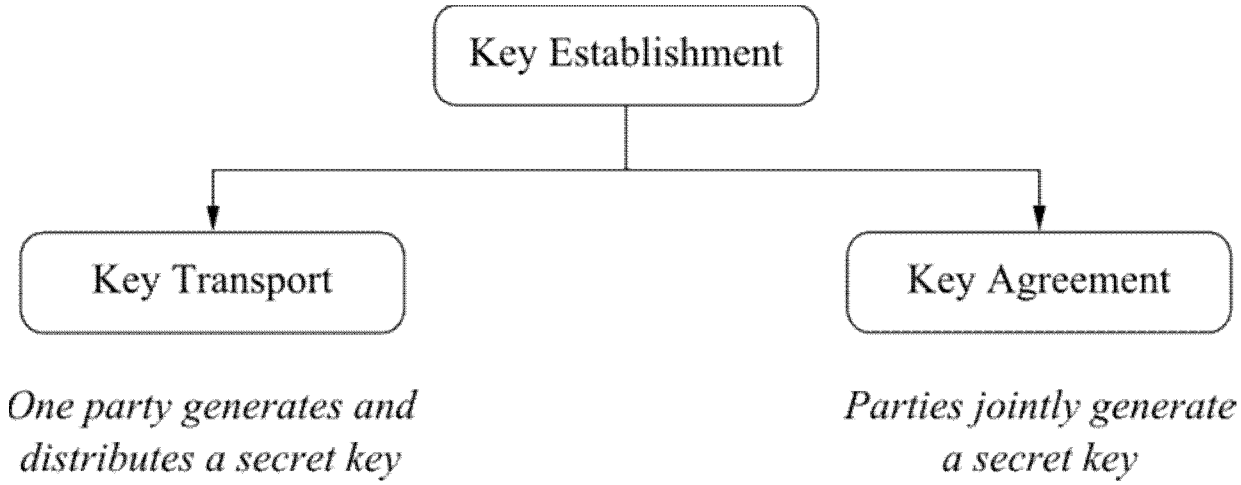
Lessons Learned

- MACs provide two security services, *message integrity and message authentication,* using symmetric techniques. MACs are widely used in protocols.
- Both of these services also provided by digital signatures, but MACs are much faster as they are based on symmetric algorithms.
- MACs do not provide nonrepudiation.
- In practice, MACs are either based on block ciphers or on hash functions.
- HMAC is a popular and very secure MAC, used in many practical protocols such as TLS.

Content of this Chapter

- Introduction
- The n^2 Key Distribution Problem
- Symmetric Key Distribution
- Asymmetric Key Distribution
  - Man-in-the-Middle Attack
  - Certificates
  - Public-Key Infrastructure

Classification of Key Establishment Methods



In an ideal key agreement protocol, no single party can control what the key value will be.

Key Freshness

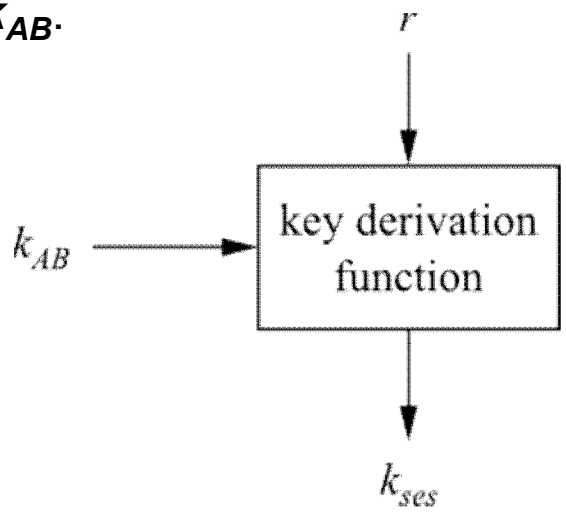
It is often desirable to frequently change the key in a cryptographic system.

Reasons for key freshness include:

- If a key is exposed (e.g., through hackers), there is limited damage if the key is changed often
- Some cryptographic attacks become more difficult if only a limited amount of ciphertext was generated under one key
- If an attacker wants to recover long pieces of ciphertext, he has to recover several keys which makes attacks harder

Key Derivation

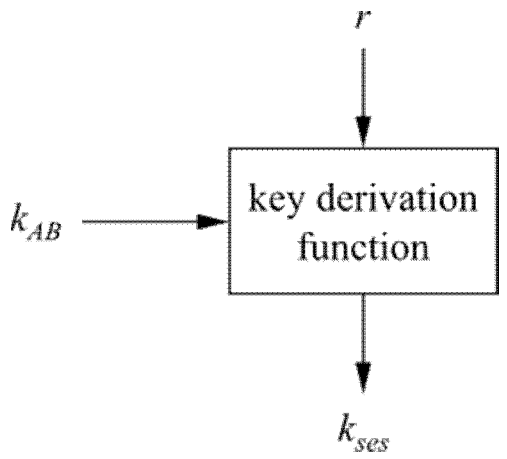
- In order to achieve key freshness, we need to generate new keys frequently.
- Rather than performing a full key establishment every time (which is costly in terms of computation and/or communication), we can **derive multiple session keys k_{ses} from a given key k_{AB}.**



- The key k_{AB} is fed into a key derivation function together with a nonce r („number used only once“).
- Every different value for r yields a different session key

Key Derivation

- The key derivation function is a computationally simple function, e.g., a block cipher or a hash function



- Example for a basic protocol:

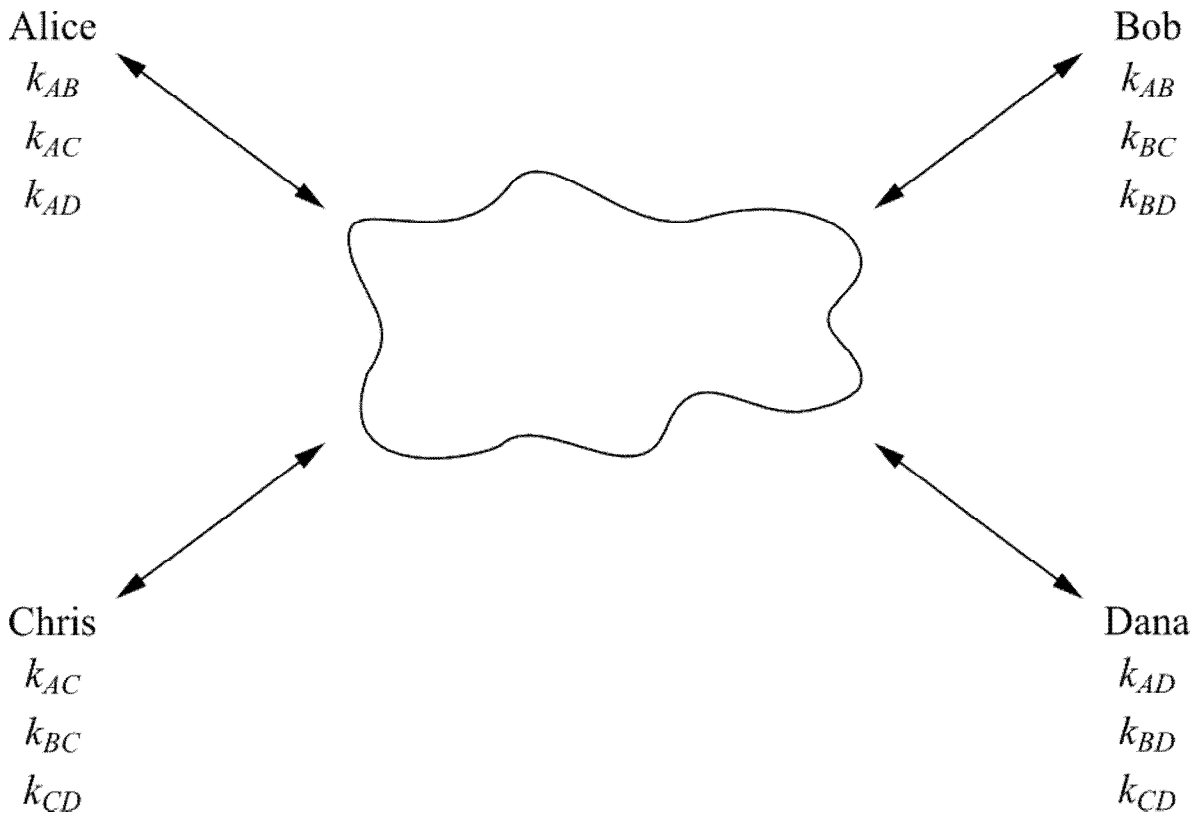
Alice		**Bob**
		generate nonce *r*
	r ←	
derive session key $K_{ses} = e_{kAB}(r)$		derive session key $K_{ses} = e_{kAB}(r)$

Content of this Chapter

- Introduction
- **The n^2 Key Distribution Problem**
- Symmetric Key Distribution
- Asymmetric Key Distribution
  - Man-in-the-Middle Attack
  - Certificates
  - Public-Key Infrastructure

The n^2 Key Distribution Problem

- Simple situation: Network with n users. Every user wants to communicate securely with every of the other n-1 users.
- Naïve approach: Every pair of users obtains an individual key pair

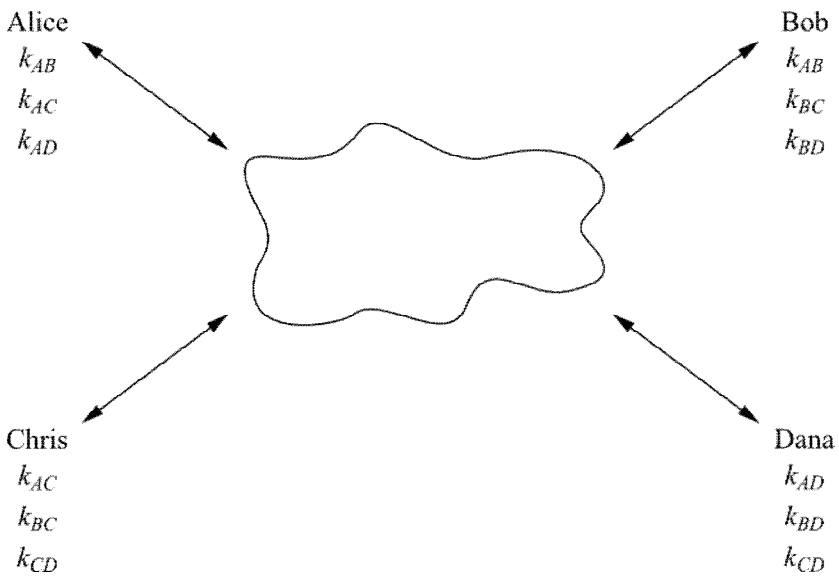


The n^2 Key Distribution Problem

Shortcomings

- There are $n\ (n\text{-}1) \approx n^2$ keys in the system
- There are $n\ (n\text{-}1)/2$ key pairs
- If a new user Esther joins the network, new keys k_{XE} have to be transported via secure channels (!) to each of the existing usersa

$\Rightarrow$ Only works for small networks which are relatively static



Example: mid-size company with 750 employees

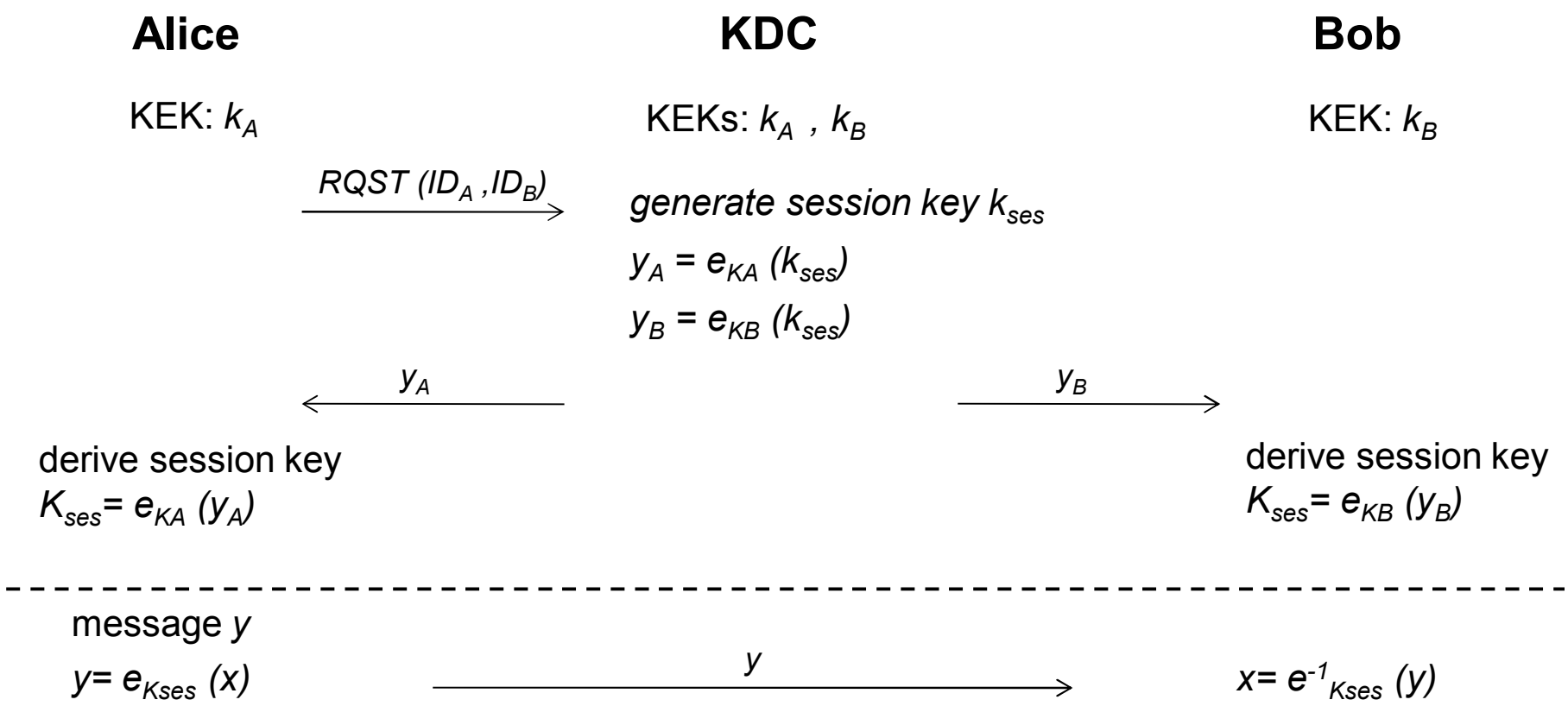
- 750 x 749 = 561,750 keys must be distributed securely

Content of this Chapter

- Introduction
- The n^2 Key Distribution Problem
- **Symmetric Key Distribution**
- Asymmetric Key Distribution
  - Man-in-the-Middle Attack
  - Certificates
  - Public-Key Infrastructure

Key Establishment with Key Distribution Center

- Key Distribution Center (KDC) = Central party, trusted by all users
- KDC shares a **key encryption key (KEK)** with each user
- Principle: KDC sends session keys to users which are encrypted with KEKs



Key Establishment with Key Distribution Center

- Advantages over previous approach:
  - Only *n* long-term key pairs are in the system
  - If a new user is added, a secure key is only needed between the user and the KDC (the other users are not affected)
  - Scales well to moderately sized networks

- *Kerberos* (a popular authentication and key distribution protocol) is based on KDCs
- More information on KDCs and Kerberos: Section 13.2 of *Understanding Cryptography*

Key Establishment with Key Distribution Center

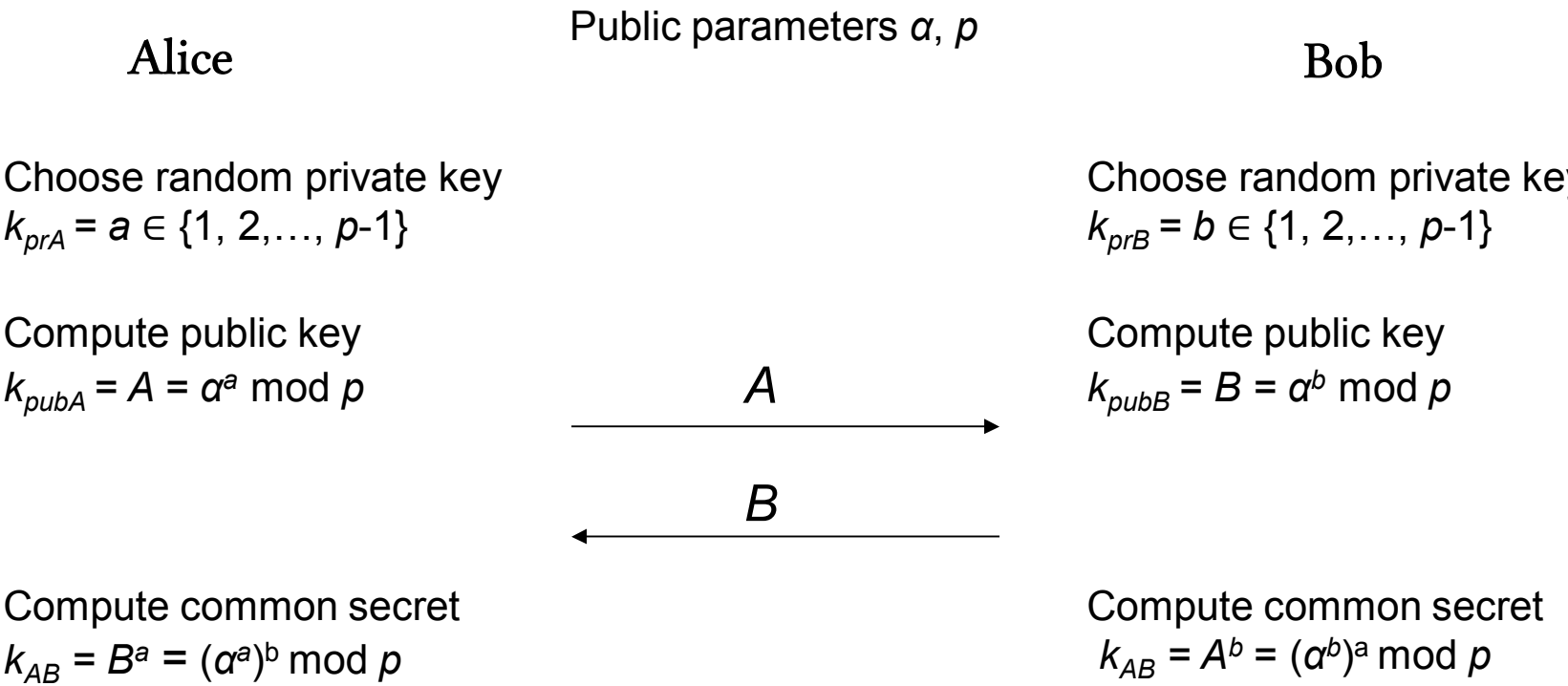
Remaining problems:

- **No *Perfect Forward Secrecy***: If the KEKs are compromised, an attacker can decrypt past messages if he stored the corresponding ciphertext
- **Single point of failure**: The KDC stores all KEKs. If an attacker gets access to this database, all past traffic can be decrypted.
- **Communication bottleneck**: The KDC is involved in every communication in the entire network (can be countered by giving the session keys a long life time)
- For more advanced attacks (e.g., key confirmation attack): Cf. Section 13.2 of *Understanding Cryptography*

Content of this Chapter

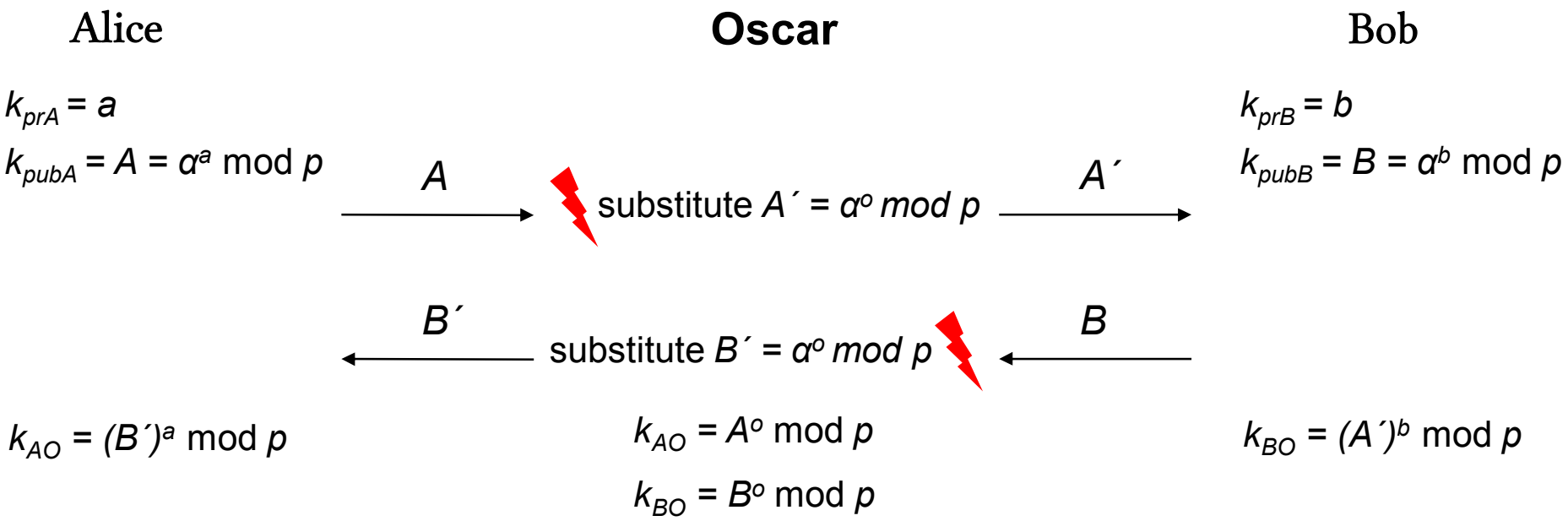
- Introduction
- The n^2 Key Distribution Problem
- Symmetric Key Distribution
- **Asymmetric Key Distribution**
  - **Man-in-the-Middle Attack**
  - Certificates
  - Public-Key Infrastructure

Recall: Diffie–Hellman Key Exchange (DHKE)



- Widely used in practice
- If the parameters are chosen carefully (especially a prime $p > 2^{1024}$), the DHKE is secure against *passive* (i.e., listen-only) attacks
- However: If the attacker can *actively* intervene in the communciation, the man-in-the-middle attack becomes possible

Man-in-the-Middle Attack



- Oscar computes a session key k_{AO} with Alice, and k_{BO} with Bob
- However, Alice and Bob think they are communicationg with each other !
- The attack efficiently performs 2 DH key-exchanges: Oscar-Alice and Oscar-Bob
- Here is why the attack works:

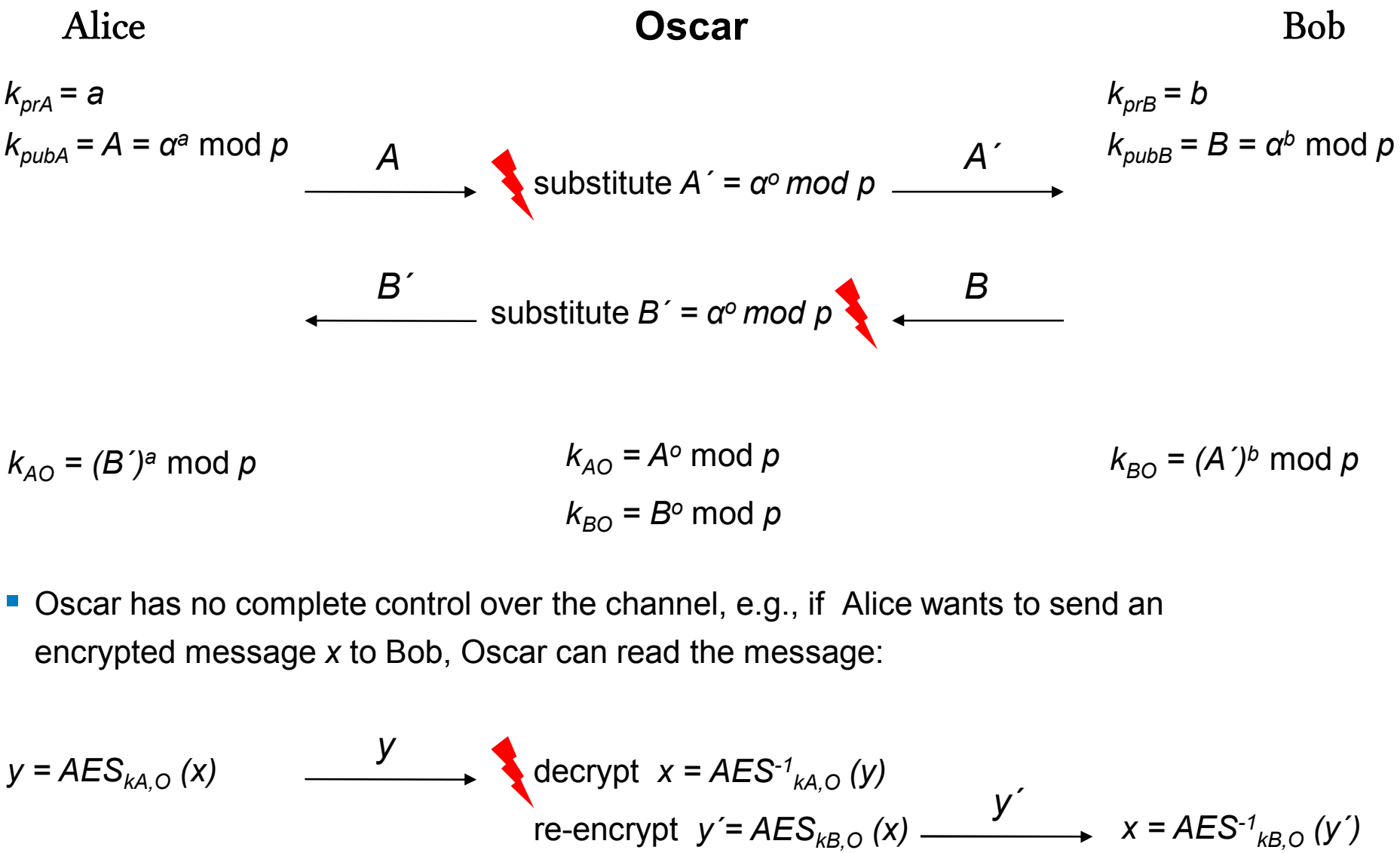
Alice computes: $k_{AO} = (B')^a = (\alpha^o)^a$

Oscar computes: $k_{AO} = A^o = (\alpha^a)^o$

Bob computes: $k_{BO} = (A')^b = (\alpha^o)^b$

Oscar computes: $k_{BO} = B^o = (\alpha^a)^o$

Implications of the Man-in-the-Middle Attack



- Oscar has no complete control over the channel, e.g., if Alice wants to send an encrypted message x to Bob, Oscar can read the message:

Very, very important facts about the Man-in-the-Middle Attack

- **The man-in-the-middle-attack is not restricted to DHKE; it is applicable to any public-key scheme, e.g. RSA encryption. ECDSA digital signature, etc. etc.**
- The attack works always by the same pattern: Oscar replaces the public key from one of the parties by his own key.
- The attack is also known as MIM attack or Janus attack

- Q: What is the underlying problem that makes the MIM attack possible?
- A: The public keys are not authenticated: When Alice receives a public key which is allegedly from Bob, she has no way of knowing whether it is in fact his. (After all, a key consists of innocent bits; it does not smell like Bob‘s perfume or anything like that)

Even though public keys can be sent over unsecure channels, they require authenticated channels.

Content of this Chapter

- Introduction
- The n^2 Key Distribution Problem
- Symmetric Key Distribution
- Asymmetric Key Distribution
  - Man-in-the-Middle Attack
  - **Certificates**
  - Public-Key Infrastructure

Certificates

- In order to authenticate public keys (and thus, prevent the MIM attack) , all public keys are digitally signed by a central trusted authority.
- Such a construction is called *certificate*

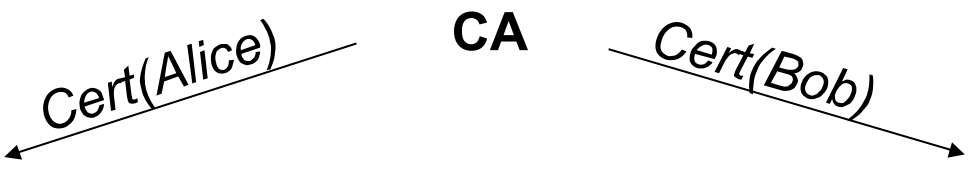
certificate = public key + ID(user) + digital signature over public key and ID

- In its most basic form, a certificate for the key k_{pub} of user Alice is:

$$\textbf{Cert(Alice)} = (\boldsymbol{k_{pub}}, \textbf{\textit{ID(Alice)}}, \boldsymbol{sig_{K_{CA}}}(\boldsymbol{k_{pub}}, \textbf{ID(Alice)}))$$

- Certificates bind the identity of user to her public key
- The trusted authority that issues the certificate is referred to as ***certifying authority (CA)***
- „Issuing certificates“ means in particular that the CA computes the signature $sig_{K_{CA}}(k_{pub})$ using its (super secret!) private key k_{CA}
- The party who receives a certificate, e.g., Bob, verifies Alice‘s public key using the public key of the CA

Diffie–Hellman Key Exchange (DHKE) with Certificates



$k_{prA} = a$

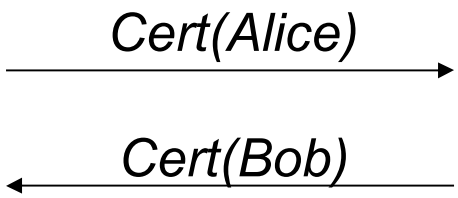
$k_{pubA} = A$

$Cert(Alice) = ((A, ID_A), sig_{KCA}(A, ID_A))$

$k_{prB} = b$

$k_{pubB} = B = \alpha^b \bmod p$

$Cert(Bob) = ((B, ID_B), sig_{KCA}(B, ID_B))$



verify certificate
$ver_{Kpub,CA}(Cert(Bob))$

if verification is correct:
Compute common secret
$k_{AB} = B^a = (\alpha^a)^b \bmod p$

verify certificate
$ver_{Kpub,CA}(Cert(Alice))$

if verification is correct:
Compute common secret
$k_{AB} = A^b = (\alpha^b)^a \bmod p$

Certificates

- Note that verfication requires the public key of the CA for $ver_{Kpub,CA}$
- In principle, an attacker could run a MIM attack when $k_{pub,CA}$ is being distributed

  $\Rightarrow$ The public CA keys must also be distributed via an authenticated channel!

- Q: So, have we gained anything?
  After all, we try to protect a public key (e.g., a DH key) by using yet another public-key scheme (digital signature for the certificate)?

- A: YES! The difference from before (e.g., DHKE without certificates) is that **we only need to distribute the public CA key *once***, often at the set-upt time of the system

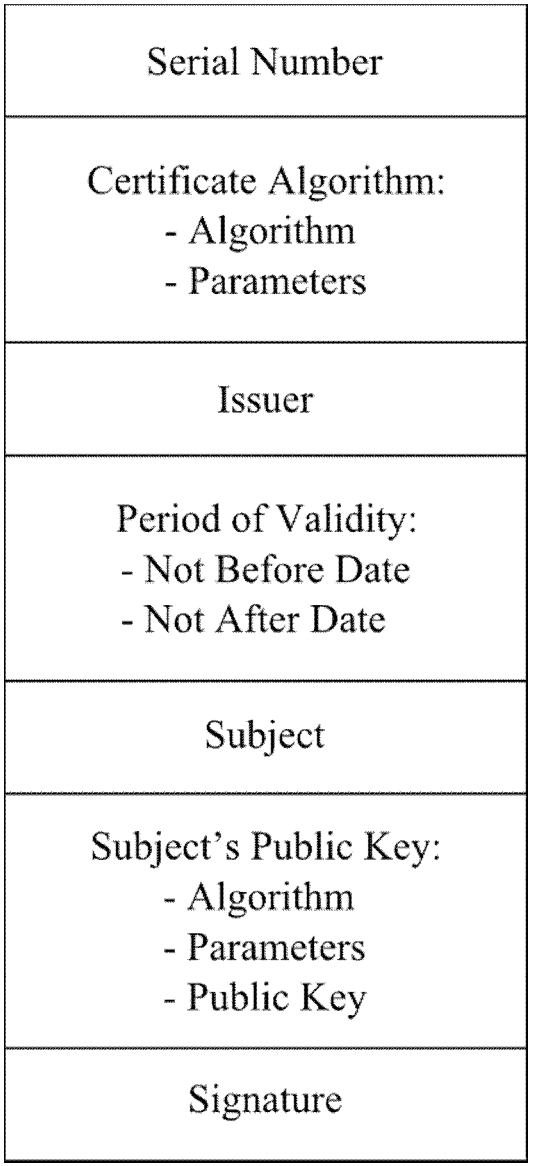
- Example: Most web browsers are shipped with the public keys of many CAs. The „authenticated channel“ is formed by the (hopefully) correct distribution of the original browser software.

Content of this Chapter

- Introduction
- The n^2 Key Distribution Problem
- Symmetric Key Distribution
- Asymmetric Key Distribution
  - Man-in-the-Middle Attack
  - Certificates
  - **Public-Key Infrastructure**

Certificates in the Real World

- In the wild certificates contain much more information than just a public key and a signature.
- X509 is a popular signature standard. The main fields of such a certificate are shown to the right.
- Note that the „Signature“ at the bottom is computed over all other fields in the certifcate (after hashing of all those fields).
- It is important to note that there are **two** public-key schemes involved in every certificate:
  1. The public-key that actually is protected by the signature („Subject‘s Public Key“ on the right). This was the public Diffie-Hellman key in the earlier examples.
  2. The digital signature algorithm used by the CA to sign the certificate data.
- For more information on certificates, see Section 13.3 of *Understanding Cryptography*



Remaining Issues with PKIs

There are many additional problems when certificates are to be used in systems with a large number of participants. The more pressing ones are:

1. Users communicate which other whose certificates are issued by different CAs
   - This requires cross-certification of CAs, e.g.. CA1 certifies the public-key of CA2. If Alice trusts „her“ CA1, cross-certification ensures that she also trusts CA2. This is called a „chain of trust“ and it is said that „trust is delegated“.
2. Certificate Revocation Lists (CRLs)
   - Another real-world problem is that certificates must be revoced, e.g., if a smart card with certificate is lost or if a user leaves an organization. For this, CRLs must be sent out periodically (e.g., daily) which is a burden on the bandwidth of the system.

More information on PKIs and CAs can be found in Section 13.3 of *Understanding Cryptography*