



NETWORK SECURITY

Ali Shakiba

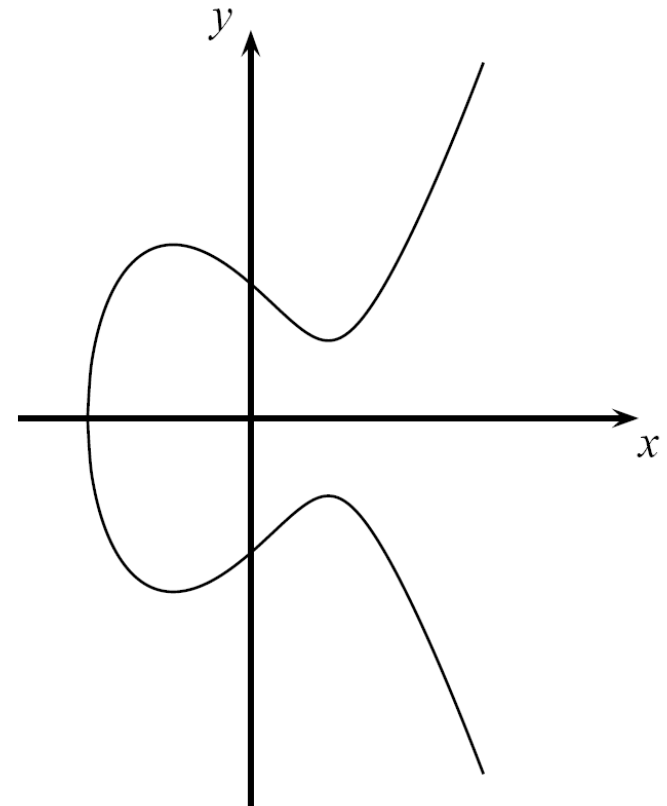
Vali-e-Asr University of Rafsanjan

ali.shakiba@vru.ac.ir

www.1ali.ir

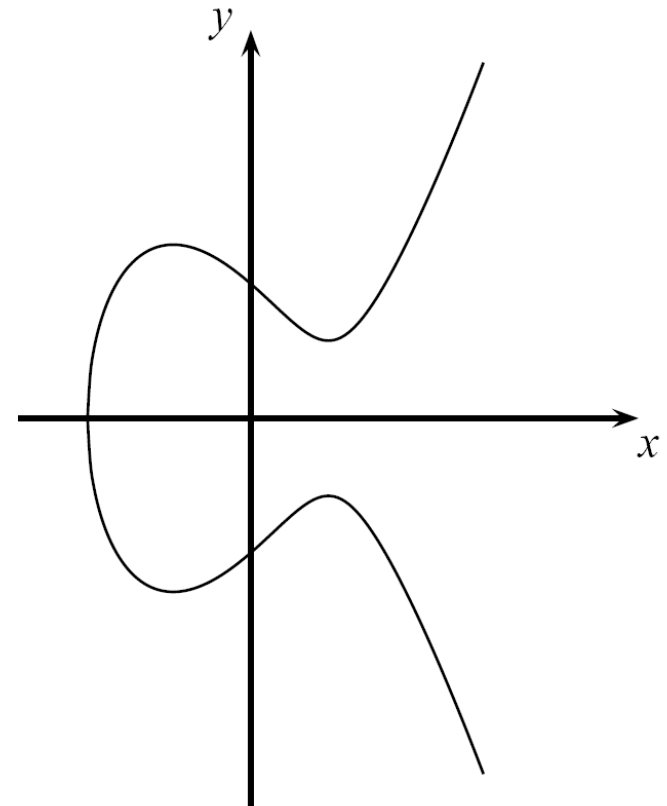
■ Content of this Chapter

- 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.

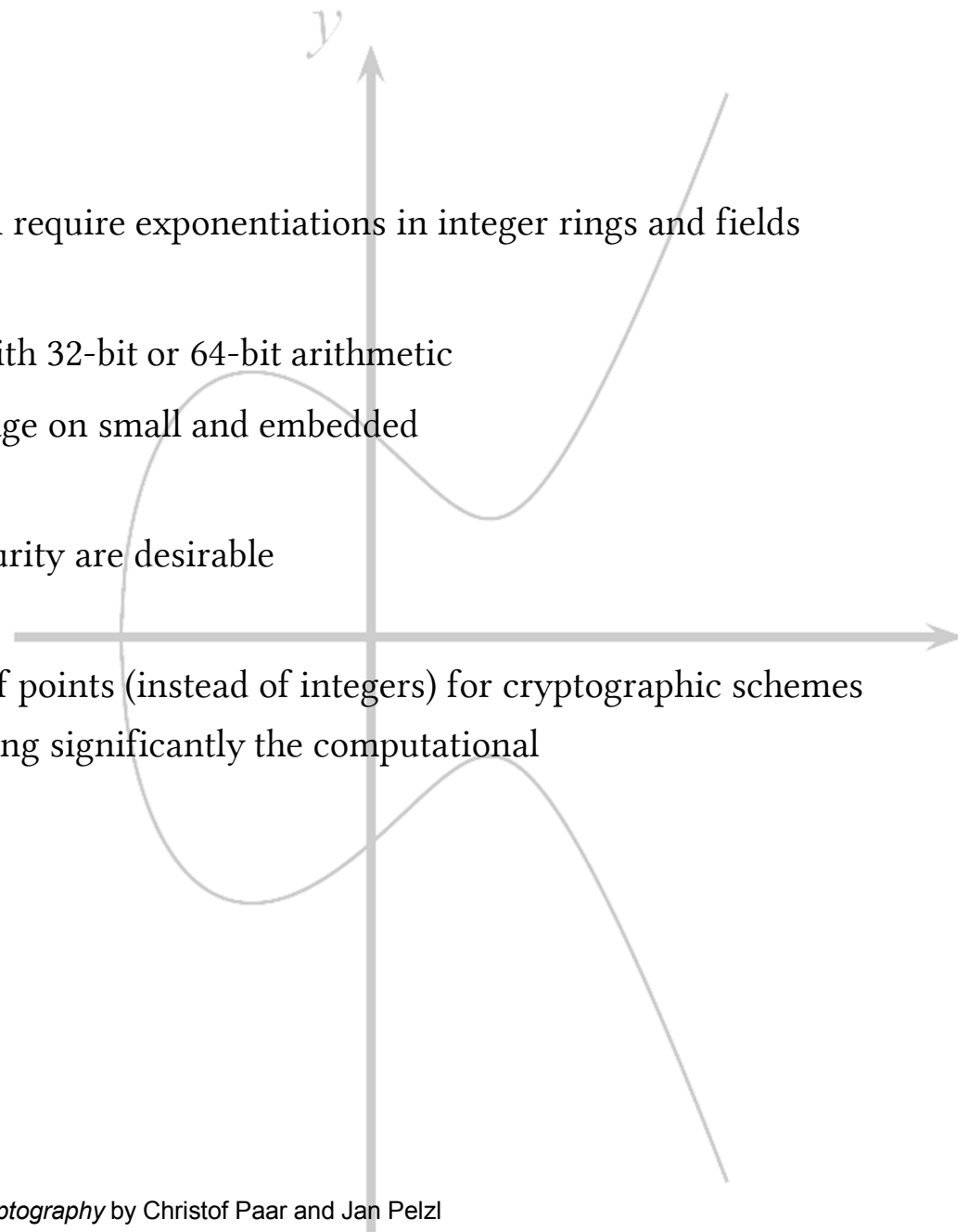
- High computational effort on CPUs with 32-bit or 64-bit arithmetic
- Large parameter sizes critical for storage on small and embedded

■ Motivation:

Smaller field sizes providing equivalent security are desirable

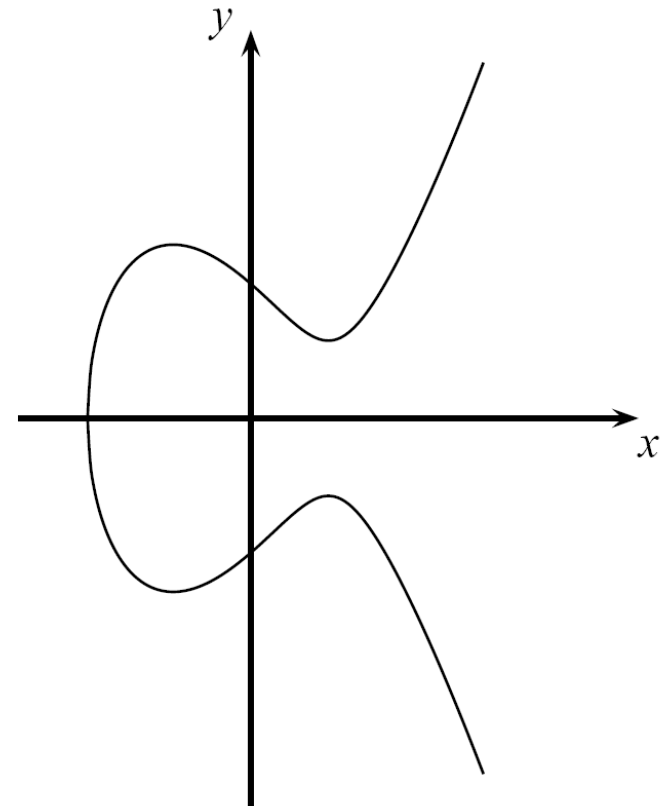
■ Solution:

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



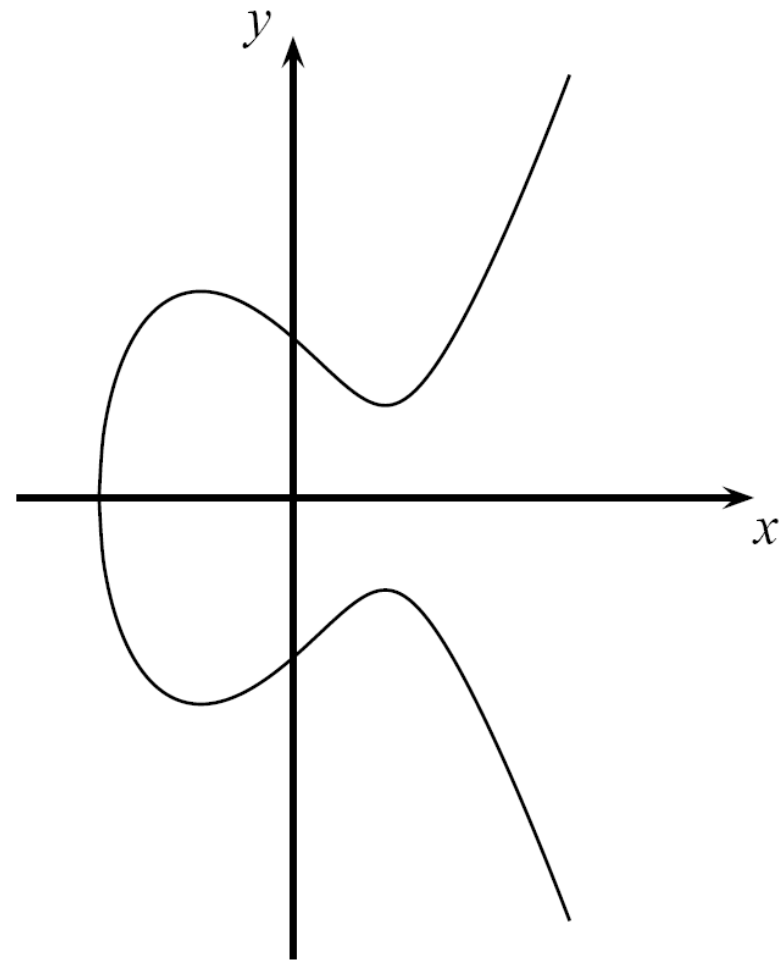
■ Computations on Elliptic Curves

- 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 $a, b \in \mathbb{R}$, an elliptic curve looks like this →
- Elliptic curves can not just be defined over the real numbers \mathbb{R} but over many other types of finite fields.



Example: $y^2 = x^3 - 3x + 3$ over \mathbb{R}

■ Computations on Elliptic Curves (ctd.)

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

Definition: Elliptic Curves over prime fields

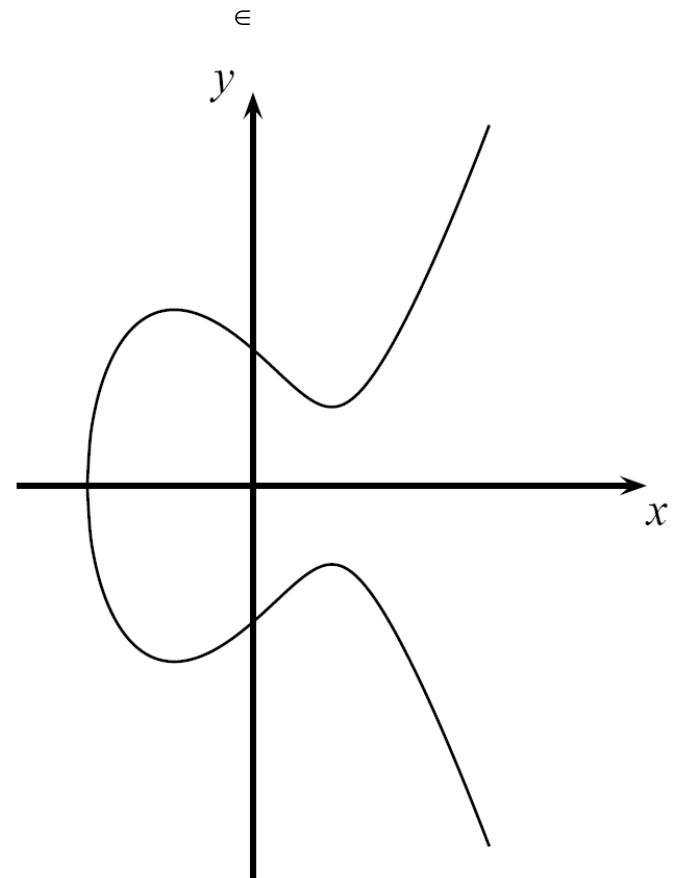
The elliptic curve over Z_p , $p > 3$ is the set of all pairs $(x, y) \in Z_p$ which fulfill

$$y^2 = x^3 + ax + b \pmod{p}$$

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

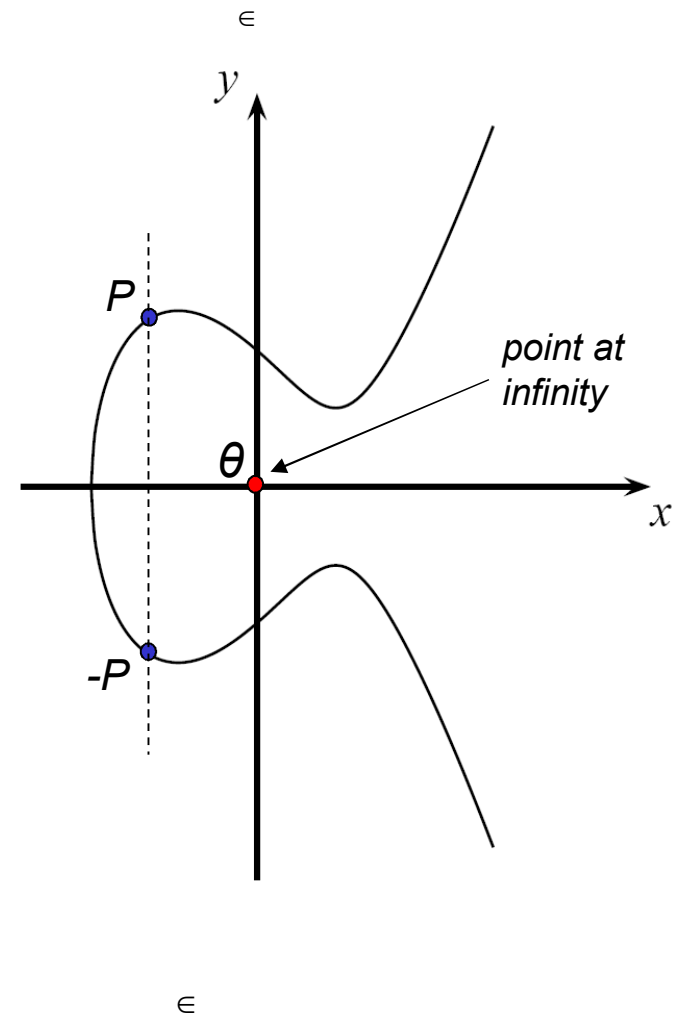
$$4a^3 + 27b^2 \neq 0 \pmod{p}.$$

- Note that $Z_p = \{0, 1, \dots, p-1\}$ 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 $P \in \mathcal{E}: P + \theta = P = \theta + P$
 - 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 Curves 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 $-P = (x, -y)$



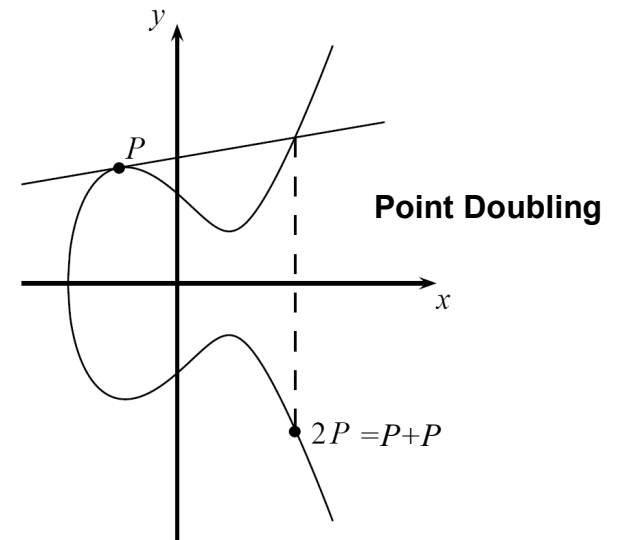
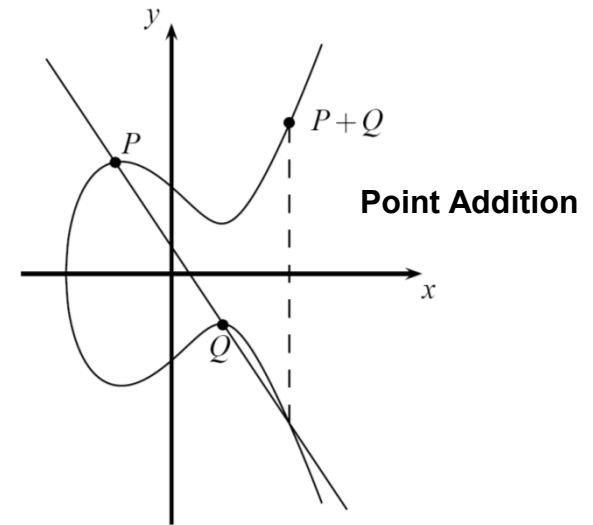
■ Computations on Elliptic Curves (ctd.)

- 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

$$x_3 = s^2 - x_1 - x_2 \pmod p \quad \text{and} \quad y_3 = s(x_1 - x_3) - y_1 \pmod p$$

where

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



■ Computations on Elliptic Curves (ctd.)

■ **Example:** Given $E: y^2 = x^3 + 2x + 2 \pmod{17}$ and point $P=(5,1)$

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

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

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

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

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

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

$$11P = (13,10)$$

$$3P = 2P + P = (10,6)$$

$$12P = (0,11)$$

$$4P = (3,1)$$

$$13P = (16,4)$$

$$5P = (9,16)$$

$$14P = (9,1)$$

$$6P = (16,13)$$

$$15P = (3,16)$$

$$7P = (0,6)$$

$$16P = (10,11)$$

$$8P = (13,7)$$

$$17P = (6,14)$$

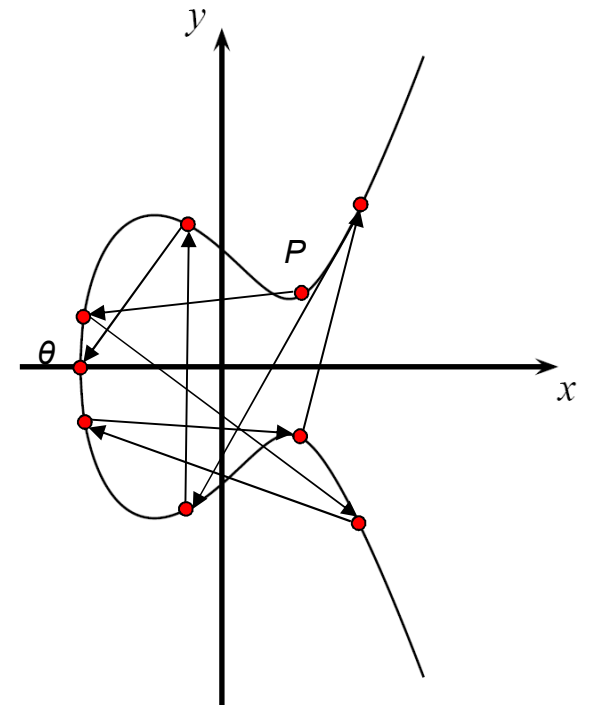
$$9P = (7,6)$$

$$18P = (5,16)$$

$$10P = (7,11)$$

$$19P = \theta$$

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: $E: y^2 = x^3 + 2x + 2 \pmod{17}$ 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

$$p+1-2\sqrt{p} \leq \#E \leq p+1+2\sqrt{p}$$

- **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

$$\underbrace{P + P + \dots + P}_{d \text{ times}} = dP = T.$$

- 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 $d_i = 1$

$$T = T + P \bmod n$$

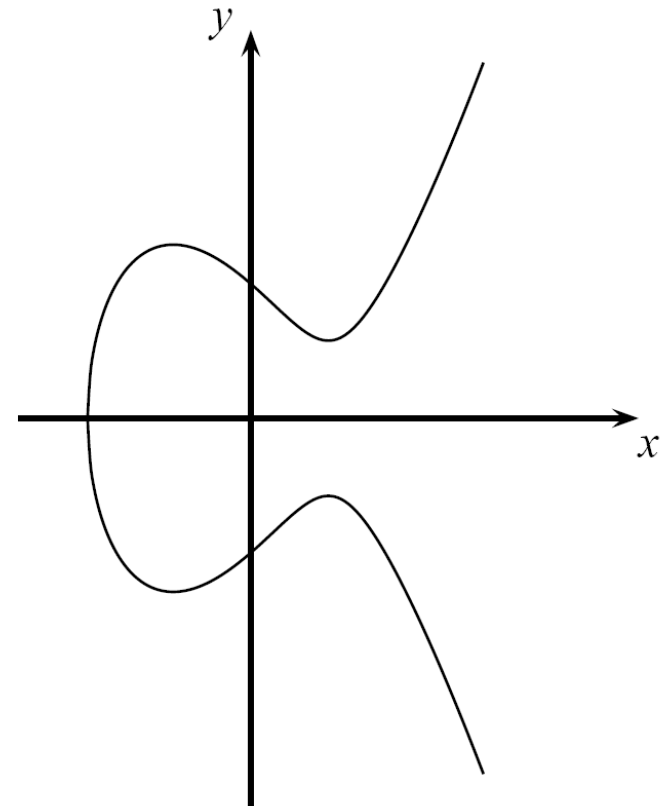
RETURN (T)

Example: $26P = (11010_2)P = (d_4d_3d_2d_1d_0)_2 P$.

Step		
#0	$P = 1_2P$	initial setting
#1a	$P+P = 2P = 10_2P$	DOUBLE (bit d_3)
#1b	$2P+P = 3P = 10^2 P + 1_2P = 11_2P$	ADD (bit $d_3=1$)
#2a	$3P+3P = 6P = 2(11_2P) = 110_2P$	DOUBLE (bit d_2)
#2b		no ADD ($d_2 = 0$)
#3a	$6P+6P = 12P = 2(110_2P) = 1100_2P$	DOUBLE (bit d_1)
#3b	$12P+P = 13P = 1100_2P + 1_2P = 1101_2P$	ADD (bit $d_1=1$)
#4a	$13P+13P = 26P = 2(1101_2P) = 11010_2P$	DOUBLE (bit d_0)
#4b		no ADD ($d_0 = 0$)

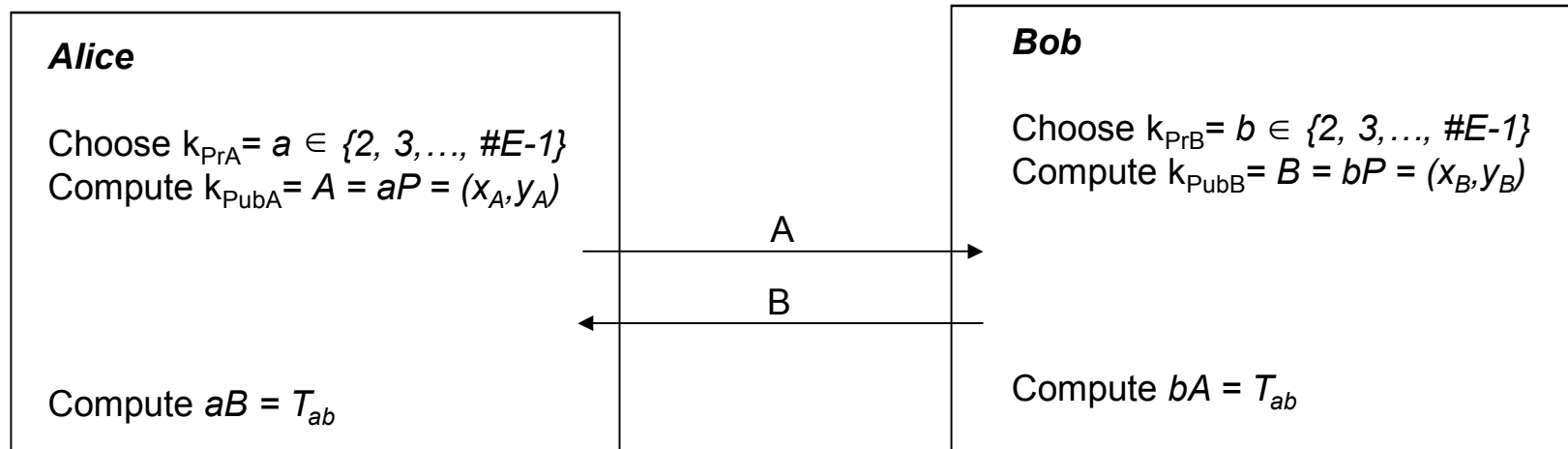
■ Content of this Chapter

- 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)

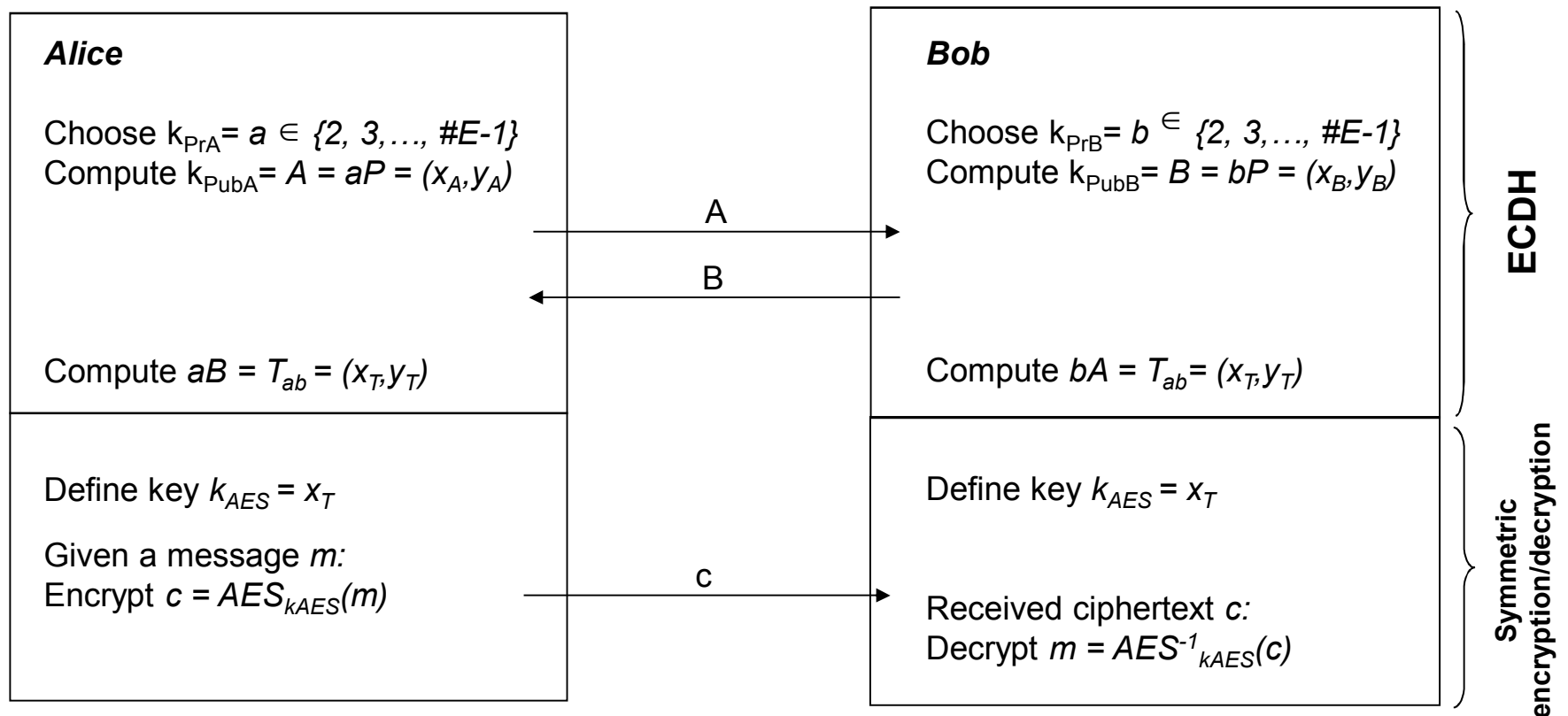
- Given a prime p , a suitable elliptic curve E and a point $P=(x_P, y_P)$
- The Elliptic Curve Diffie-Hellman Key Exchange is defined by the following protocol:



- 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.)

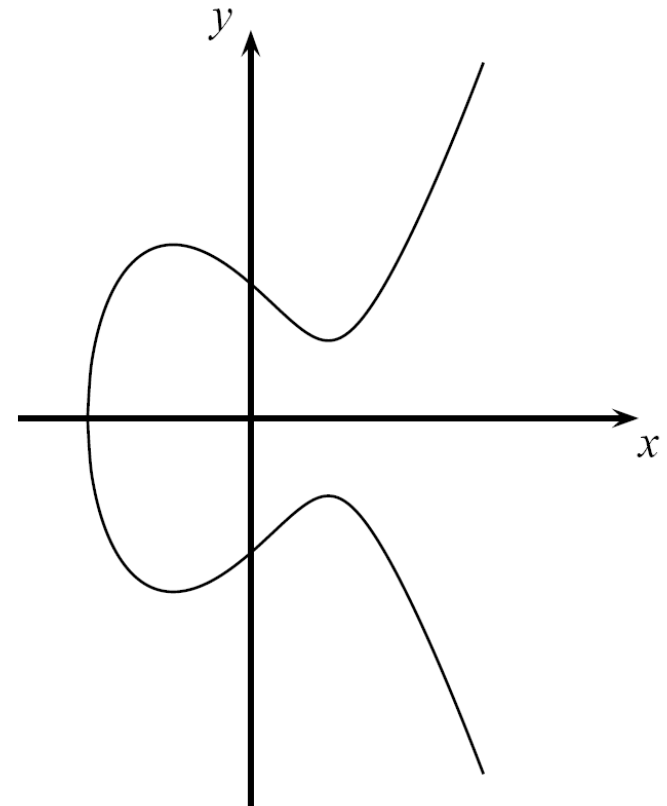
- The ECDH is often used to derive session keys for (symmetric) encryption
- One of the coordinates of the point T_{AB} (usually the x-coordinate) is taken as session key



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

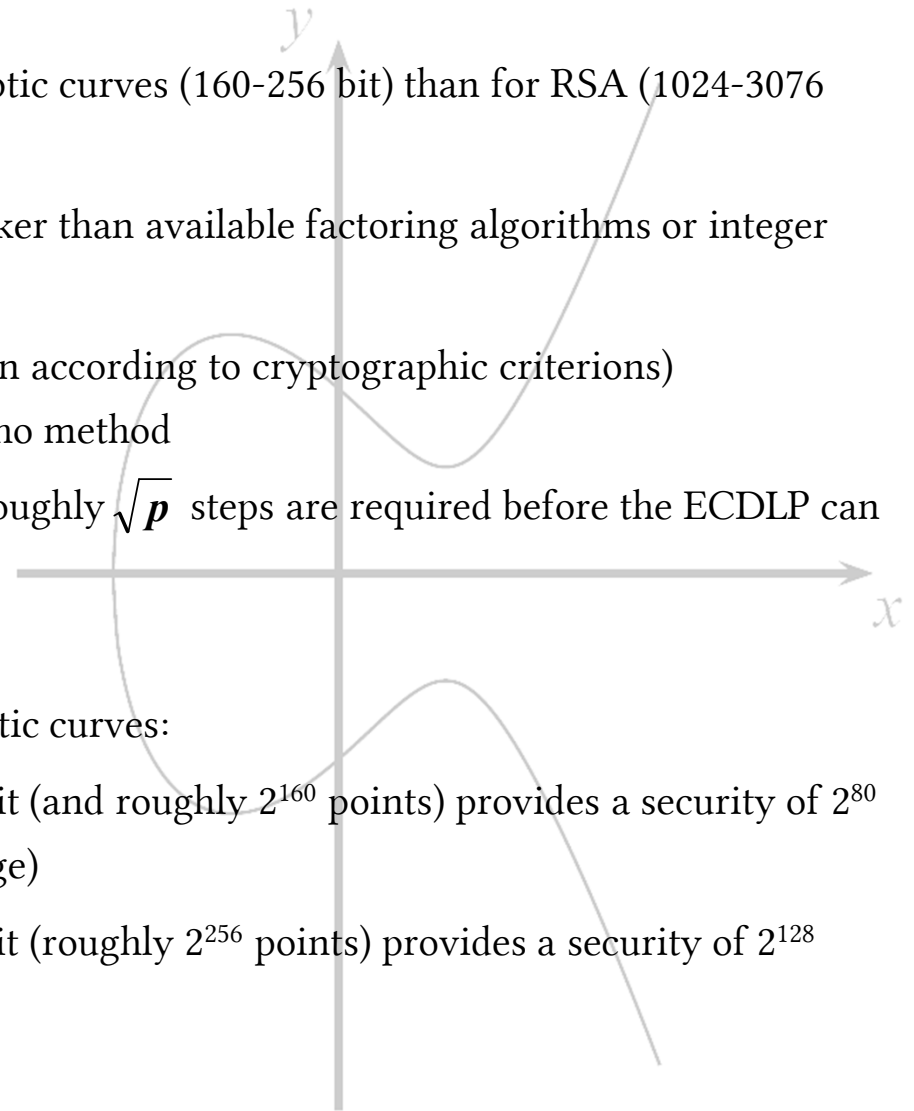
■ Content of this Chapter

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



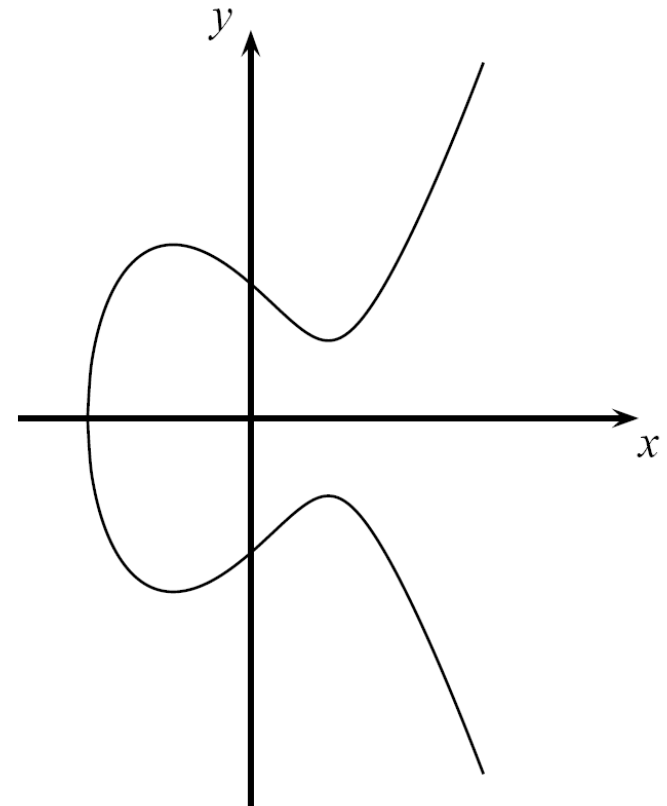
■ Security Aspects

- Why are parameters significantly 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



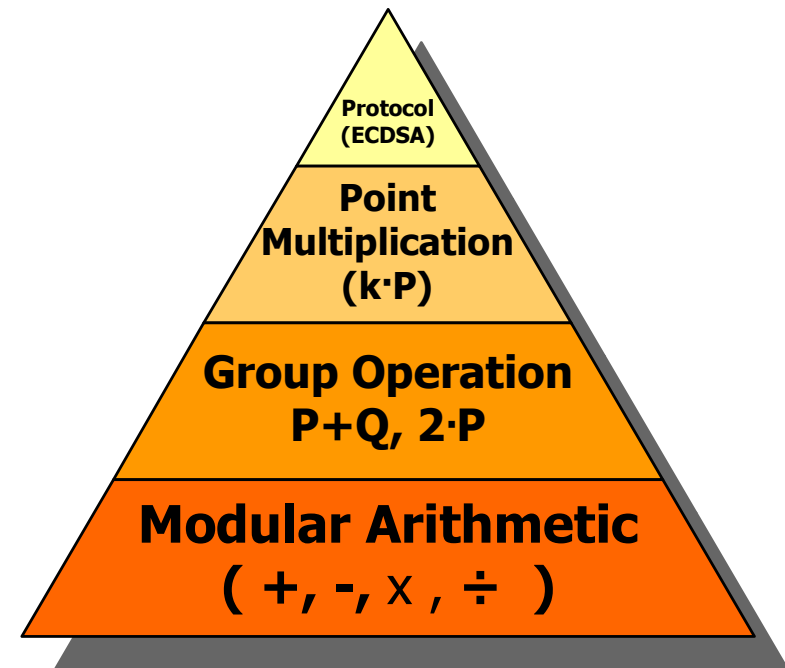
■ Content of this Chapter

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



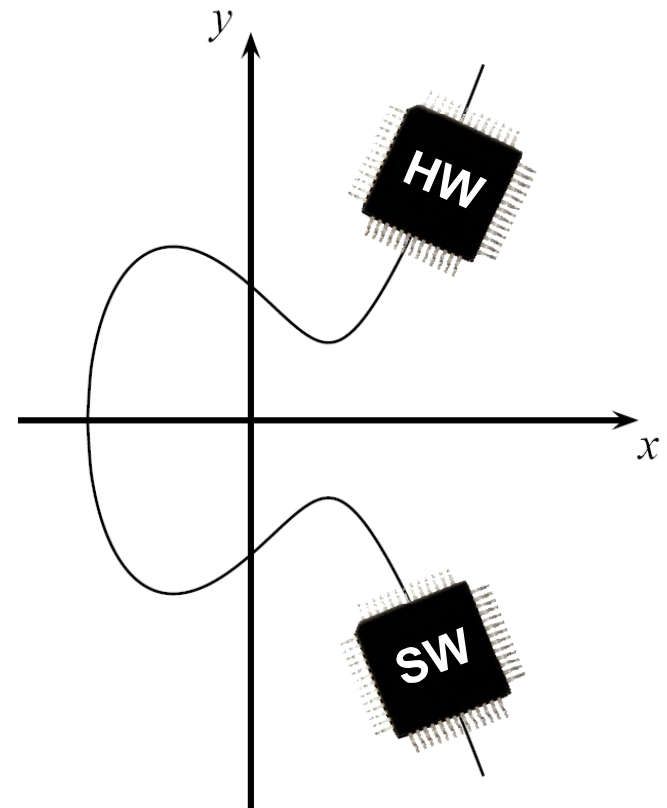
■ Implementations in Hardware and Software

- 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 ($>10\text{ ms}$)
- 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)

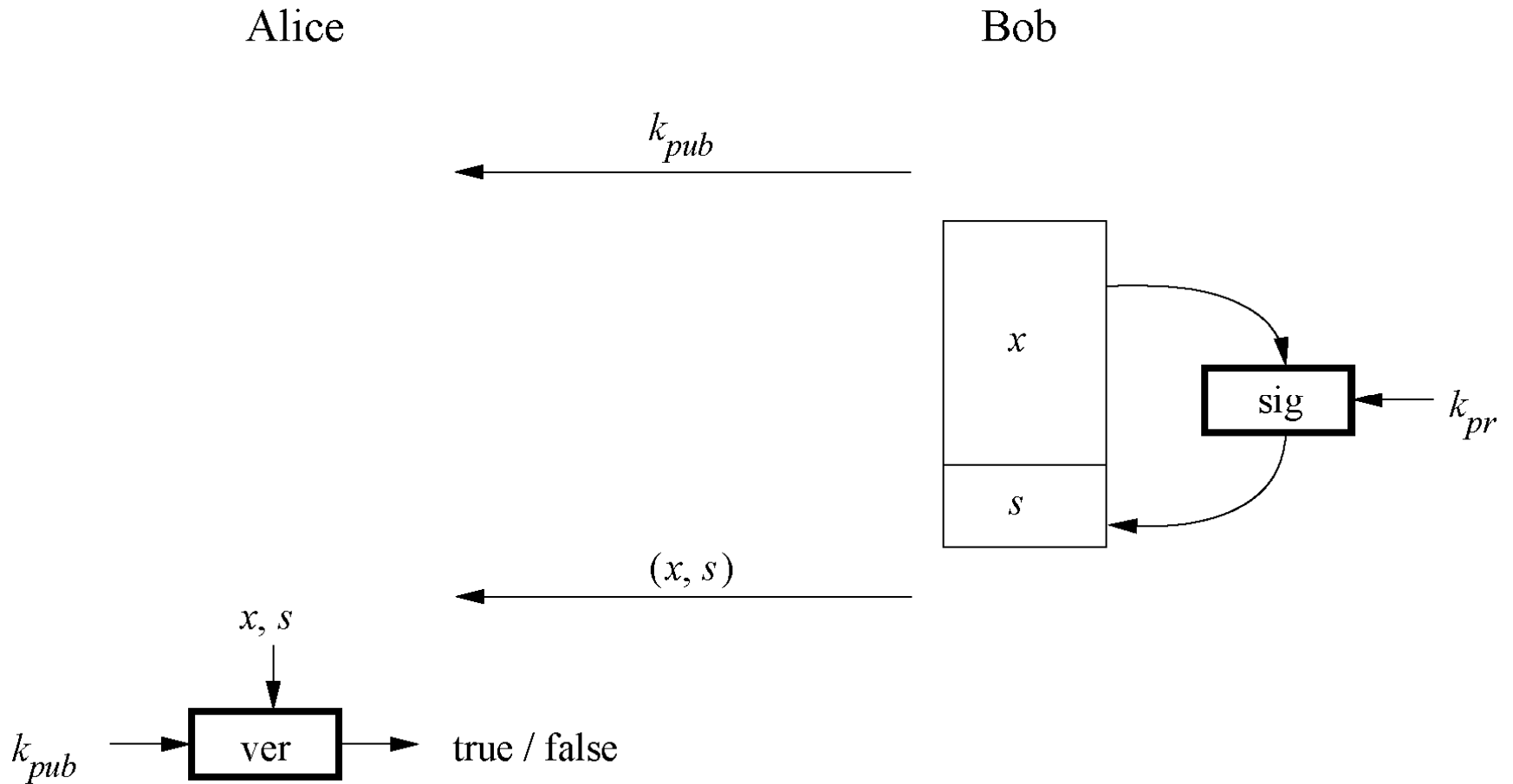
Content of this Chapter

- 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 salesman 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)
- ⇒ Symmetric cryptography fails because both Alice and Bob can be malicious
- ⇒ 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.
- ⇒ The signature is realized as a function with the message x and the private key as input.
- ⇒ 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

$$s = \text{sig}_{K_{\text{priv}}}(x) = x^d \text{ mod } n$$

- Append s to message x

To verify the signature:

- “decrypt” the signature with the public key

$$x' = \text{ver}_{K_{\text{pub}}}(s) = s^e \text{ mod } n$$

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

■ The RSA Signature Protocol

Alice

Bob

← K_{pub}

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

← (x, s)

Compute signature:

$$s = \text{sig}_{K_{pr}}(x) \equiv x^d \pmod n$$

Verify signature:

$$x' \equiv s^e \pmod n$$

If $x' \equiv x \pmod n \rightarrow$ valid signature

If $x' \not\equiv x \pmod n \rightarrow$ invalid signature

■ Security and Performance of the RSA Signature Scheme

Security:

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

⇒ 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 .

⇒ 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

← (n, e)

← (n, e)

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

1. Choose signature:
 $s \in \mathbb{Z}_n$
2. Compute message:
 $x \equiv s^e \pmod n$

← (x, s)

Verification:

$$s^e \equiv x' \pmod n$$

$$\text{since } s^e = (x^d)^e \equiv x \pmod n$$

→ 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
- ⇒ 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 $\beta \equiv \alpha^d \pmod{p}$

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 $r \equiv (\alpha^{k_E} \bmod p) \bmod q$
3. Computes $s \equiv (\text{SHA}(x) + d \cdot r) k_E^{-1} \bmod q$

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

If $v \equiv r \pmod{q} \rightarrow$ signature is valid

If $v \not\equiv r \pmod{q} \rightarrow$ signature is invalid

Proof of DSA:

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

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

$$\Leftrightarrow k_E \equiv s^{-1} \text{SHA}(x) + d \cdot s^{-1} r \pmod q$$

$$\Leftrightarrow k_E \equiv u_1 + d \cdot u_2 \pmod q$$

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

$$\Leftrightarrow \alpha^{k_E} \pmod p \equiv \alpha^{u_1 + d \cdot u_2} \pmod p$$

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

$$\Leftrightarrow \alpha^{k_E} \pmod p \equiv \alpha^{u_1} \beta^{u_2} \pmod p$$

We now reduce both sides of the equation modulo q :

$$\Leftrightarrow (\alpha^{k_E} \pmod p) \pmod q \equiv (\alpha^{u_1} \beta^{u_2} \pmod p) \pmod q$$

Since $r \equiv \alpha^{k_E} \pmod p \pmod q$ and $v \equiv (\alpha^{u_1} \beta^{u_2} \pmod p) \pmod q$, this expression is identical to:

$$\Leftrightarrow r \equiv v$$

■ Example

Alice

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

$$\leftarrow (x, (r, s)) = (x, 20, 5)$$

Verify:

$$w \equiv 5^{-1} \equiv 6 \pmod{29}$$

$$u_1 \equiv 6 \cdot 26 \equiv 11 \pmod{29}$$

$$u_2 \equiv 6 \cdot 20 \equiv 4 \pmod{29}$$

$$v = (3^{11} \cdot 4^4 \pmod{59}) \pmod{29} = 20$$

$$v \equiv r \pmod{29} \rightarrow \text{valid signature}$$

Bob

Key generation:

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

Sign:

Compute hash of message $H(x) = 26$

1. Choose ephemeral key $k_E = 10$
2. $r = (3^{10} \pmod{59}) \equiv 20 \pmod{29}$
3. $s = (26 + 7 \cdot 20) \cdot 3 \equiv 5 \pmod{29}$

■ 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

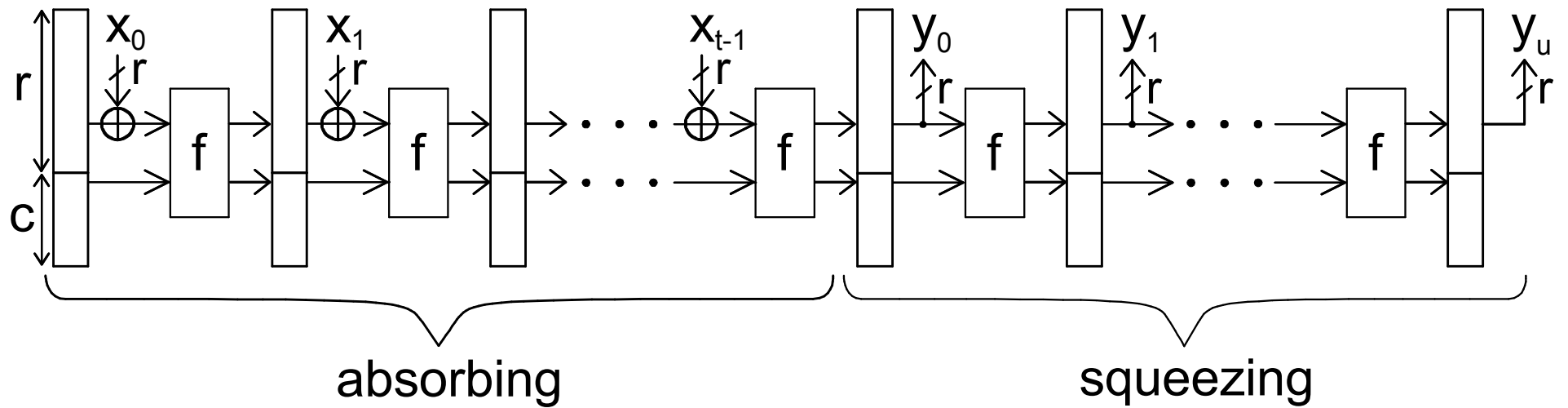


Fig. 1.3 The internal structure of Keccak

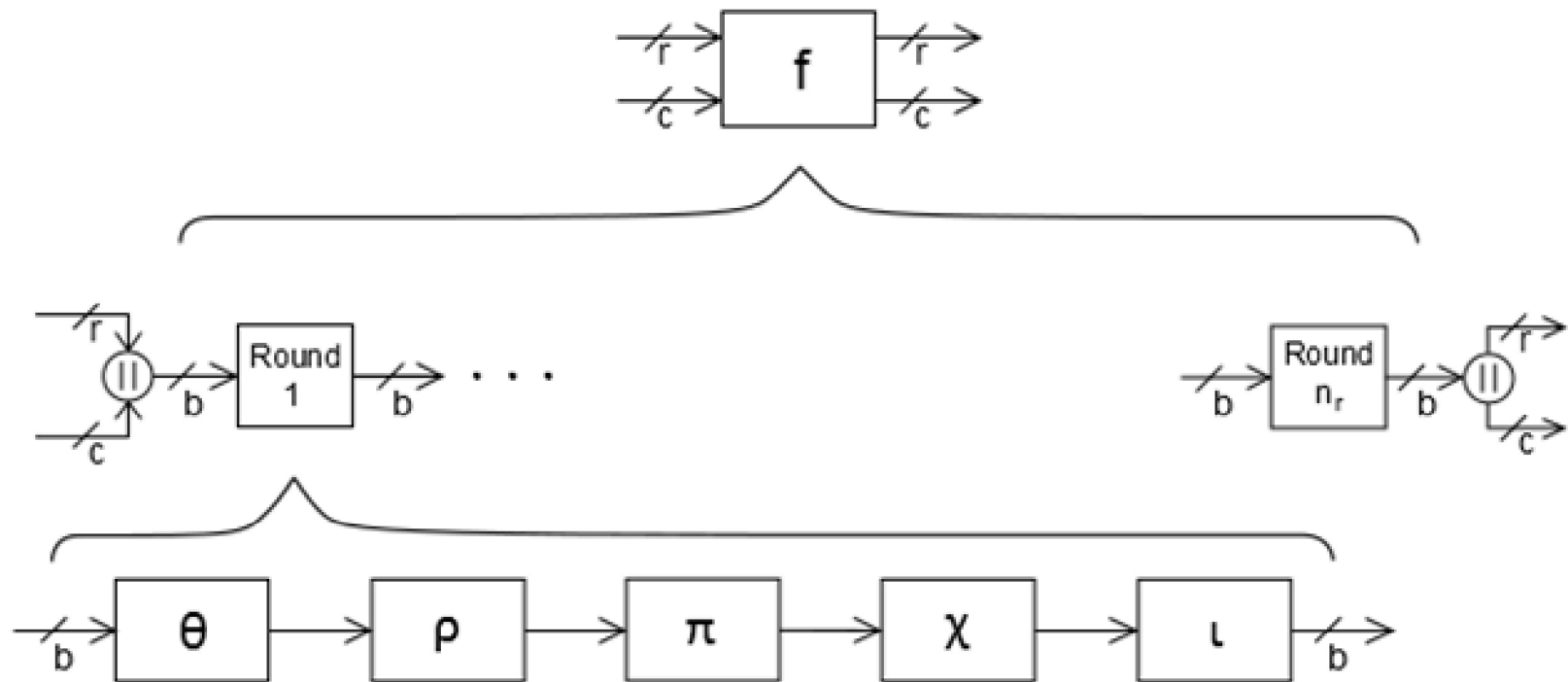


Fig. 1.4 The state of Keccak

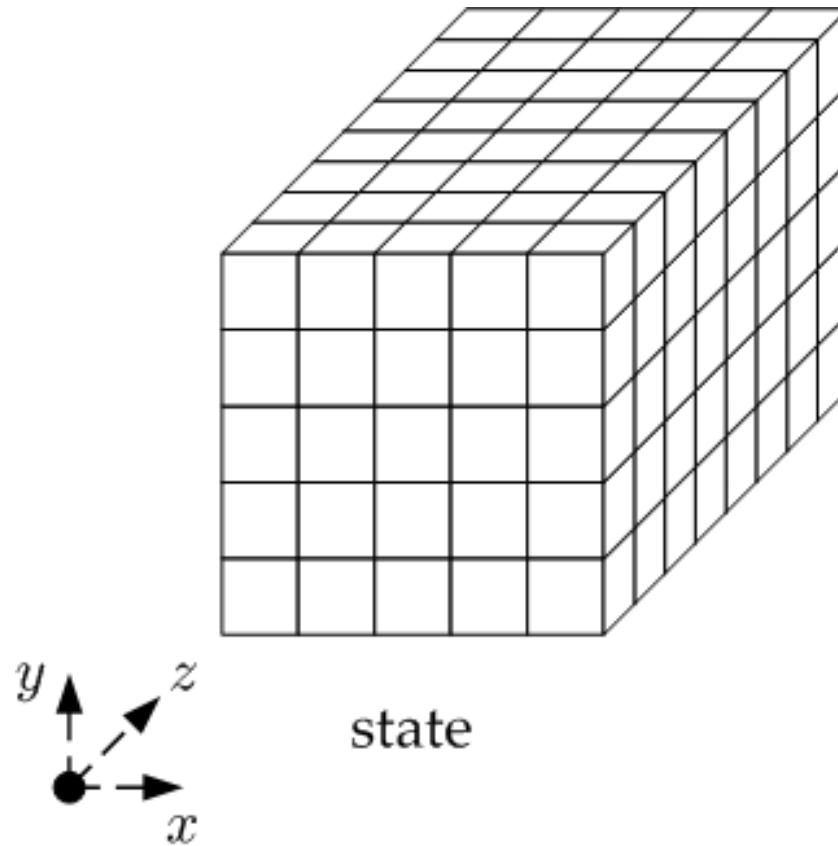


Fig. 1.5 The Theta Step of Keccak – visually

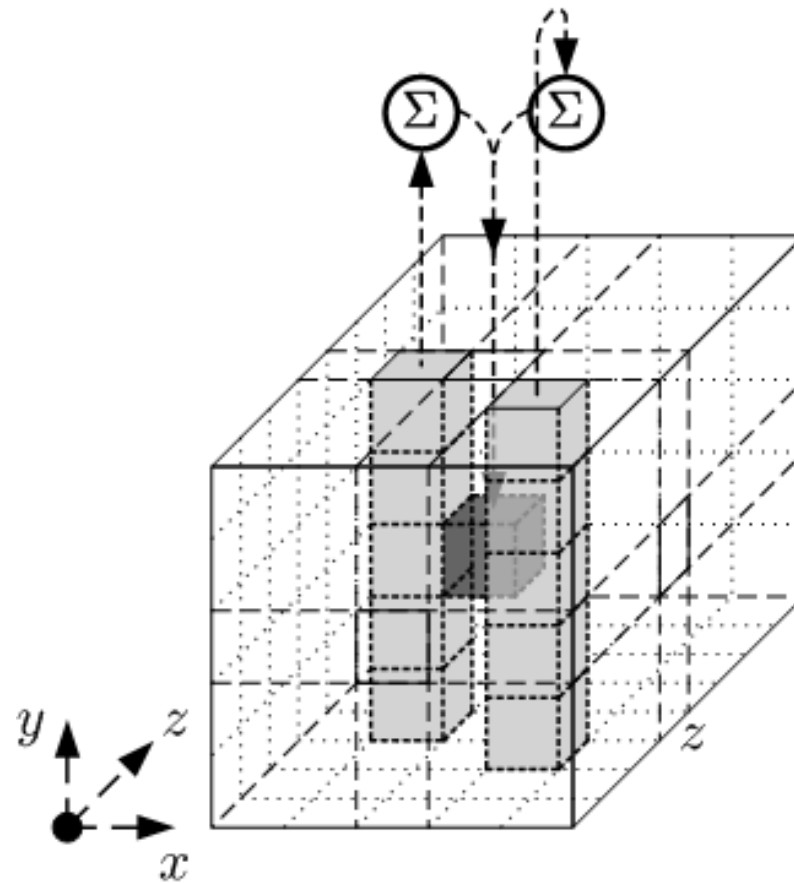


Fig. 1.5 The Theta Step of Keccak – pseudo code

- Input: state array $A[x,y]$
- Output: manipulated state array $A[x,y]$
- $C[x] = A[x,0] \oplus A[x,1] \oplus A[x,2] \oplus A[x,3] \oplus A[x,4]$ $x = 0 \dots 4$
- $D[x] = C[x-1] \oplus \text{rot}(C[x+1], 1)$ $x = 0 \dots 4$
- $A[x,y] = A[x,y] \oplus D[x]$ $x,y = 0 \dots 4$

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

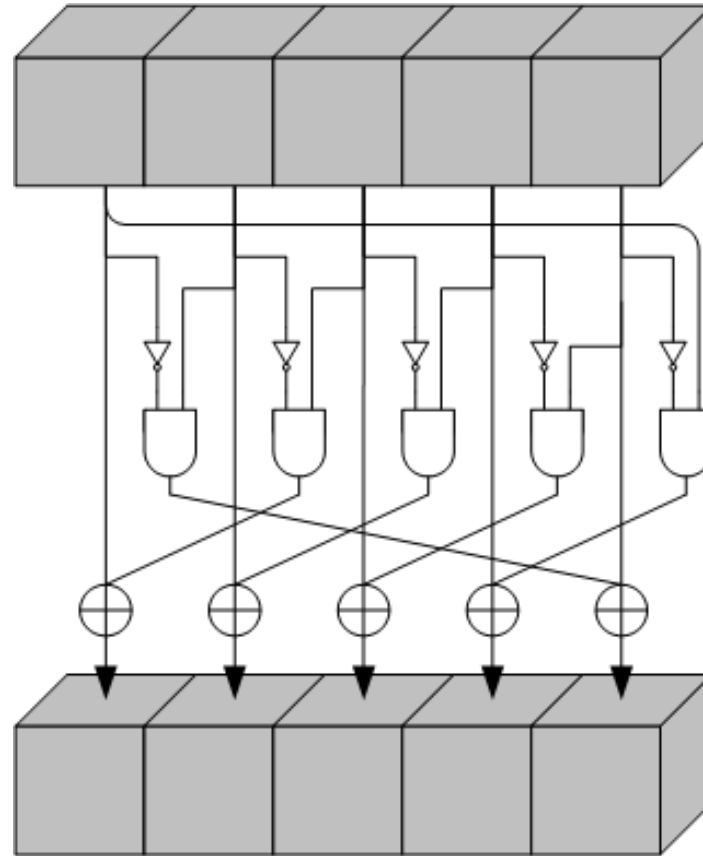


Table 1.4 The round constants of Keccak

RC[0] = 0x0000000000000001	RC[12] = 0x000000008000808B
RC[1] = 0x0000000000000802	RC[13] = 0x800000000000008B
RC[2] = 0x800000000000080A	RC[14] = 0x8000000000000809
RC[3] = 0x8000000080008000	RC[15] = 0x8000000000008003
RC[4] = 0x000000000000080B	RC[16] = 0x8000000000008002
RC[5] = 0x0000000080000001	RC[17] = 0x8000000000000080
RC[6] = 0x8000000080008081	RC[18] = 0x000000000000800A
RC[7] = 0x8000000000000809	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

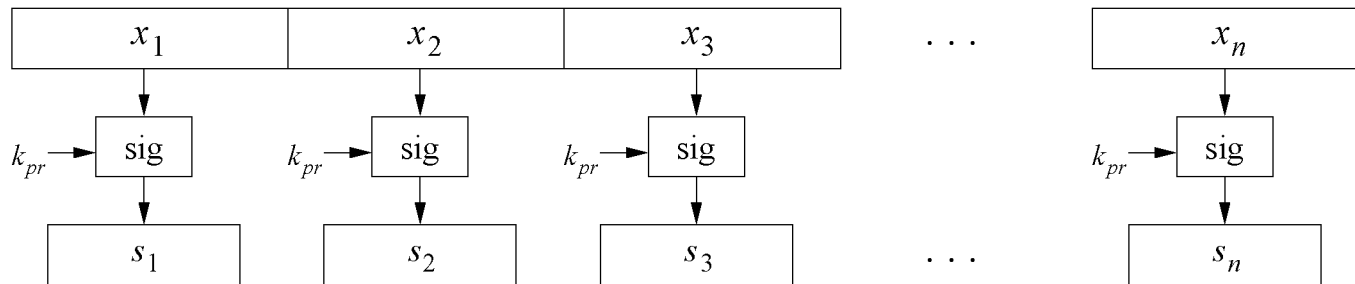
Content of this Chapter

- 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*”.

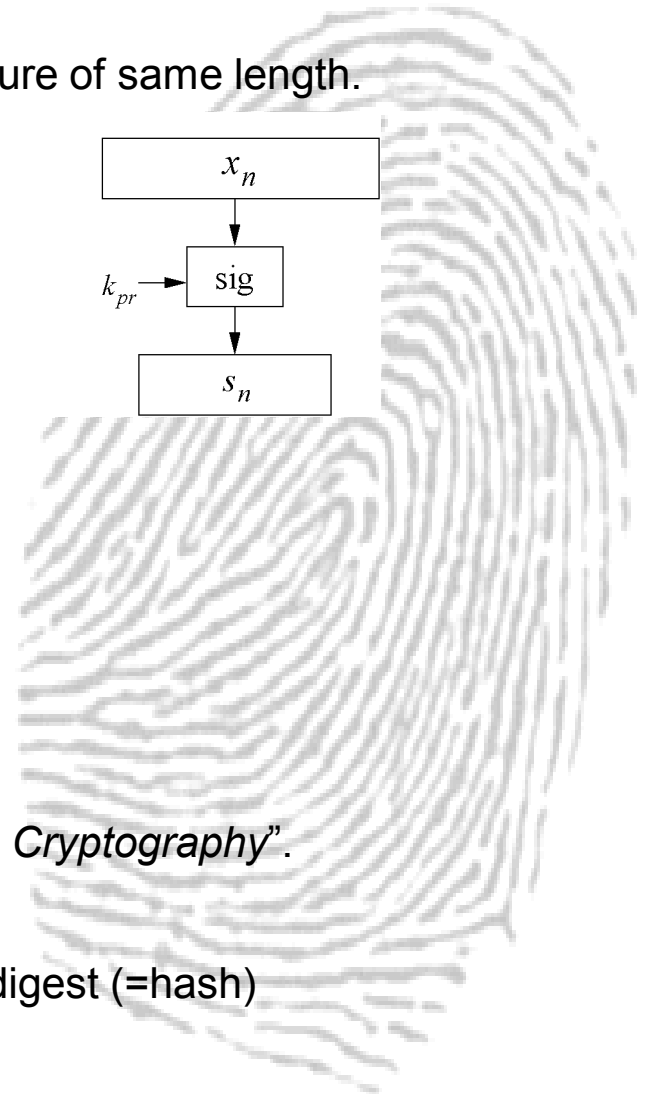
Solution:

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

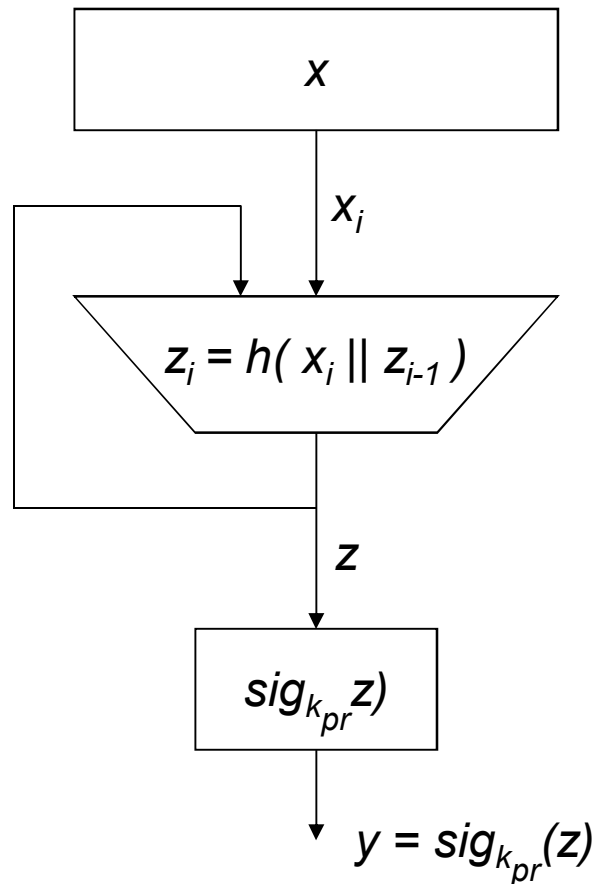
Also secure, but much faster

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:

Alice

Bob

K_{pub}



$$z = h(x)$$

$$s = \text{sig}_{K_{pr}}(z)$$

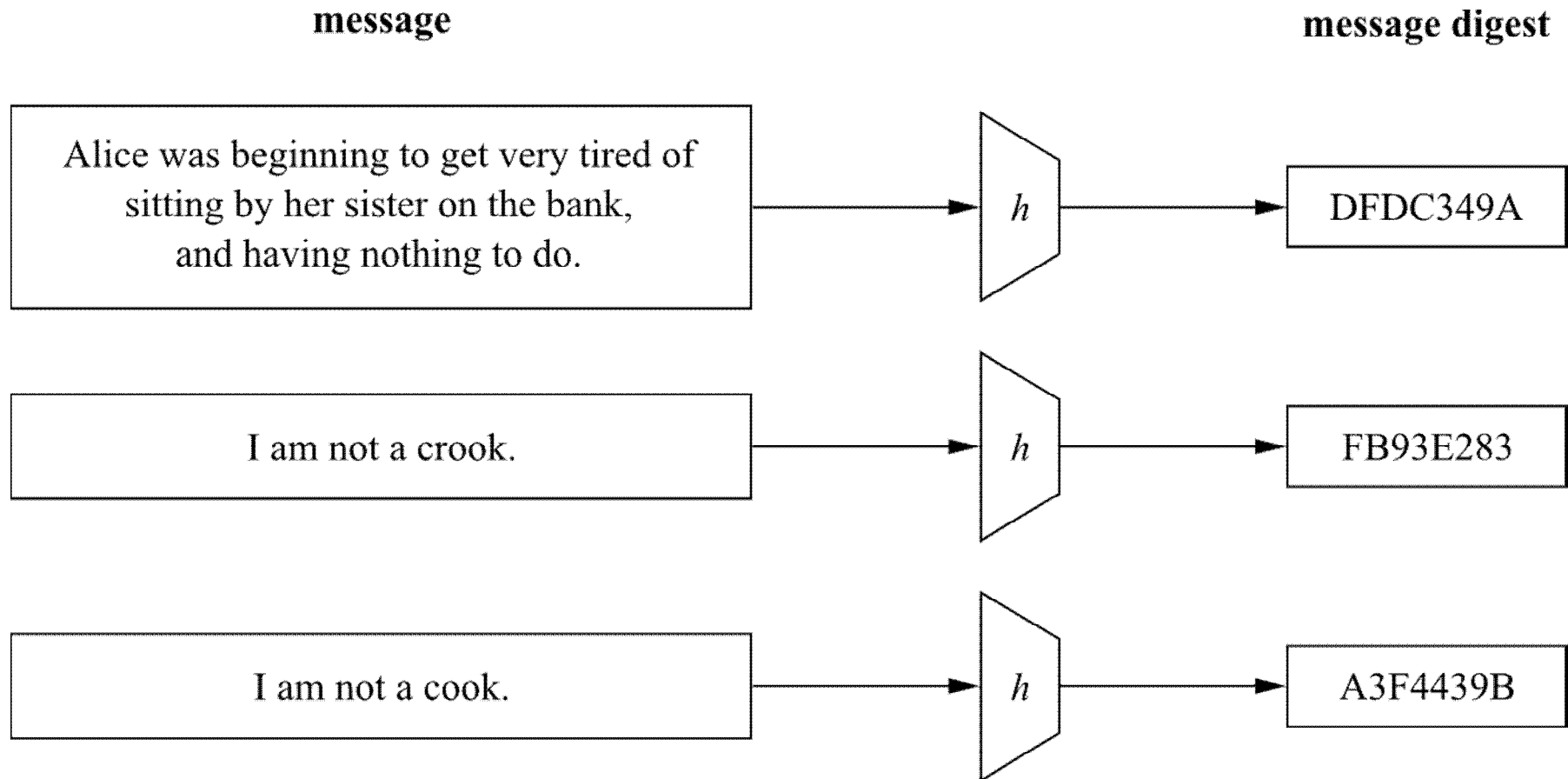
(x, s)



$$z' = h(x)$$

$$\text{ver}_{K_{pub}}(s, z') = \text{true/false}$$

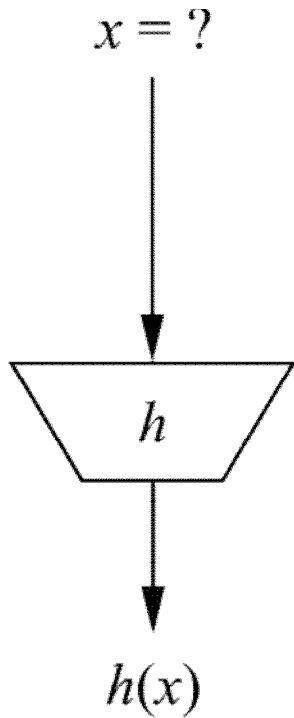
■ Principal input–output behavior of hash functions



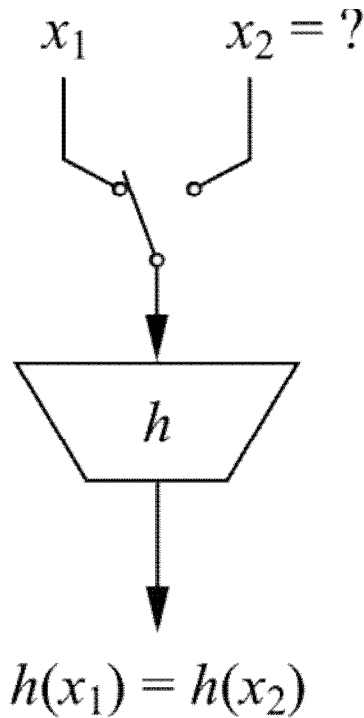
Content of this Chapter

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

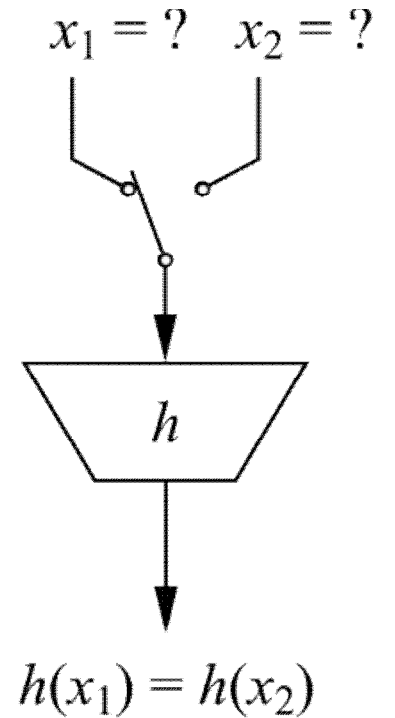
■ The three security properties of hash functions



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

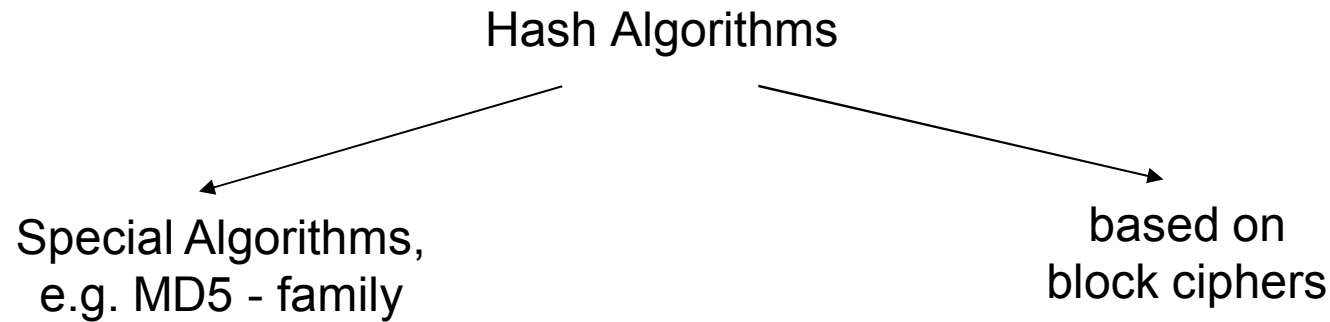
It turns out that collision resistance causes most problems

- 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

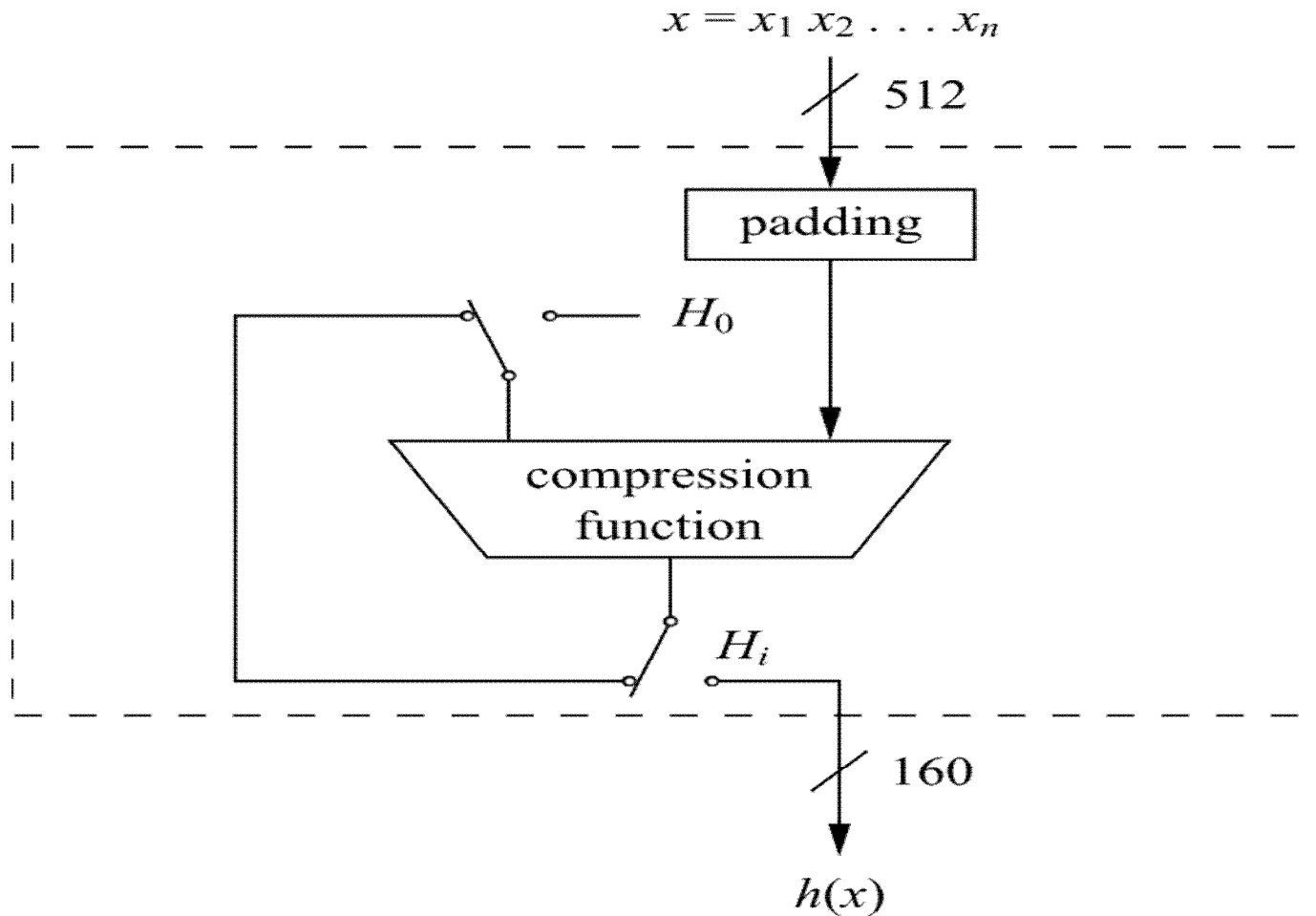
- Why we need hash functions
- How does it work
- Security properties
- Algorithms
- **Example: The Secure Hash Algorithm SHA-1**

■ SHA-1

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

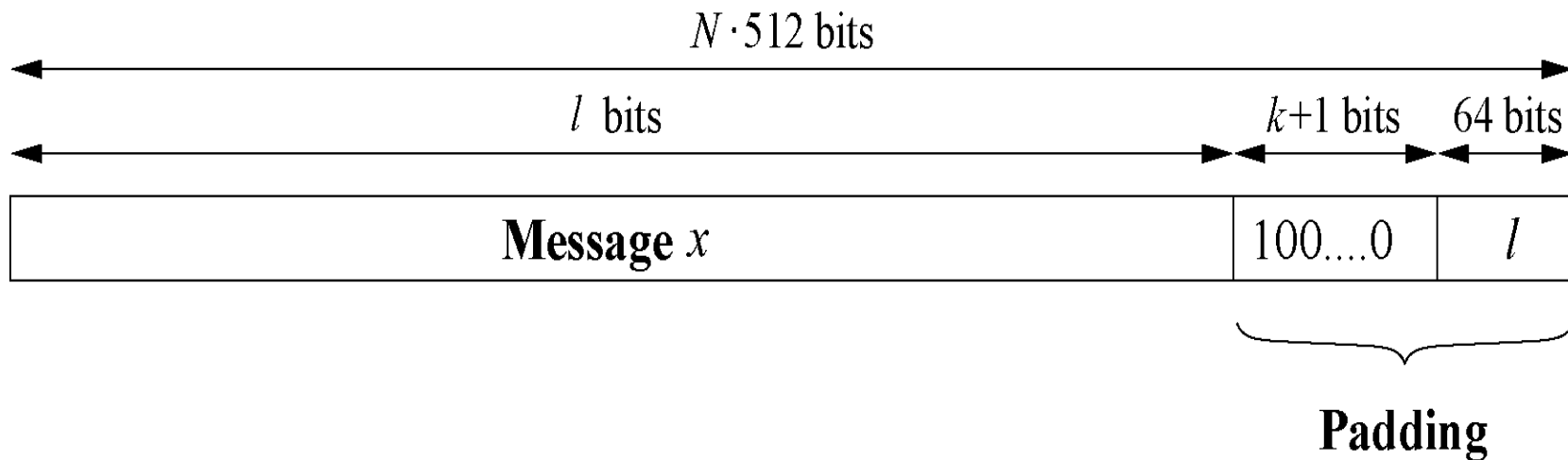
■ SHA-1 High Level Diagramm

- 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.
- $k \equiv 512 - 64 - 1 - 1 = 448 - (l + 1) \pmod{512}$.



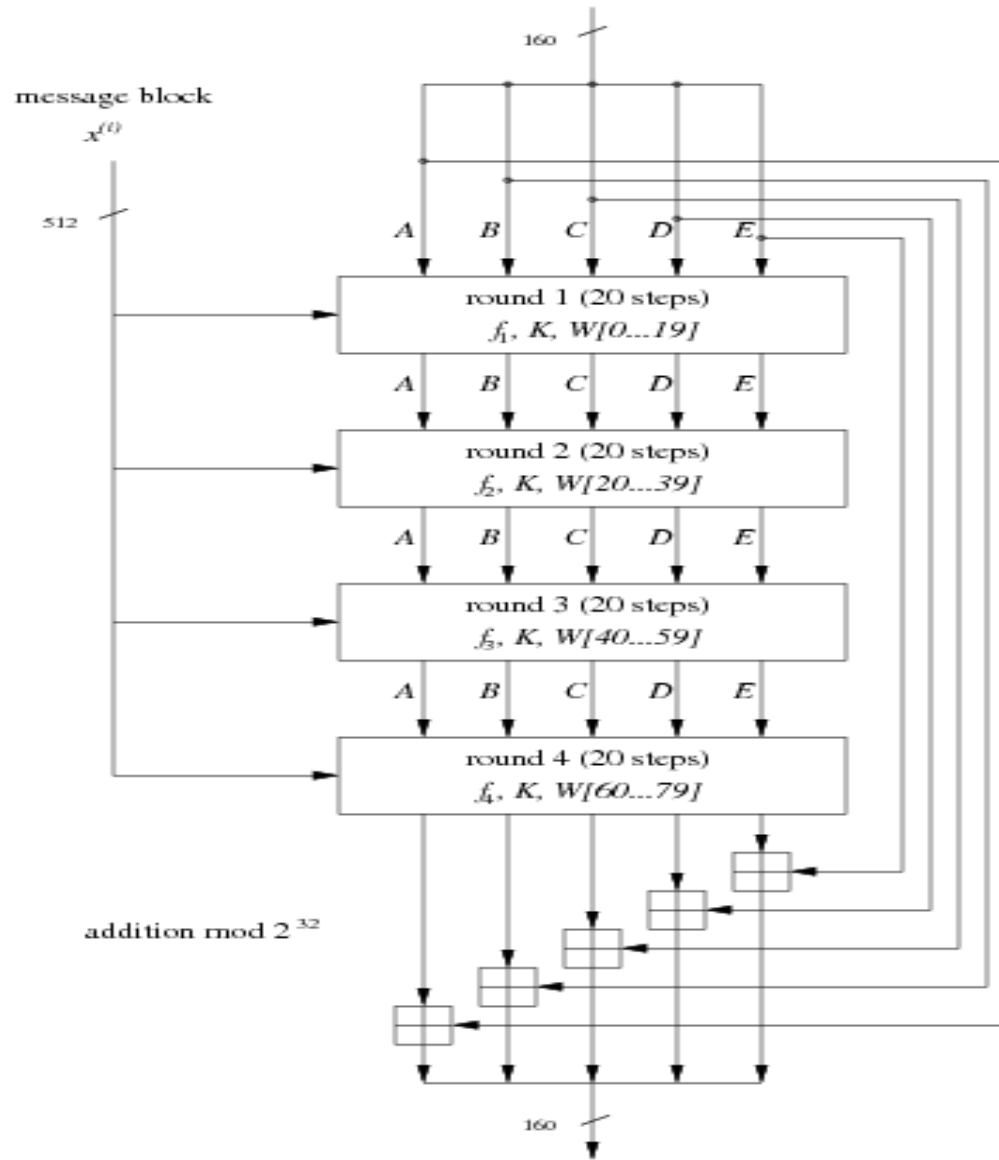
■ SHA-1: Hash Computation

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

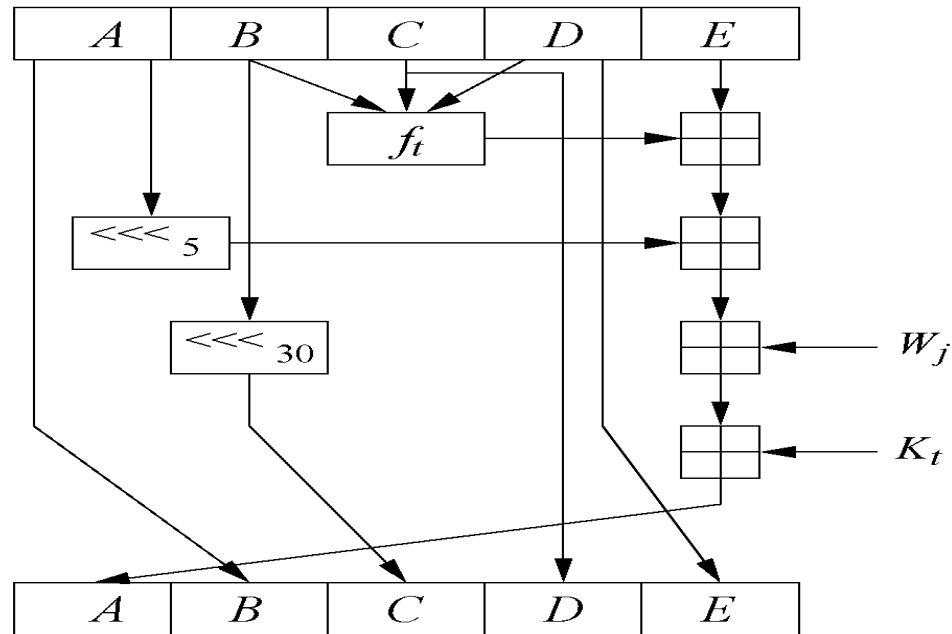
SHA-1 uses:

- A message schedule which computes a 32-bit word W_0, W_1, \dots, W_{79} 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	$K=5A827999$	$f(B,C,D)=(B \wedge C) \vee (\bar{B} \wedge D)$
2	20...39	$K=6ED9EBA1$	$f(B,C,D)=B \oplus C \oplus D$
3	40...59	$K=8F1BBCDC$	$f(B,C,D)=(B \oplus C) \vee (B \oplus D) \vee (C \oplus D)$
4	60...79	$K=CA62C1D6$	$f(B,C,D)=B \oplus C \oplus D$

■ 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**

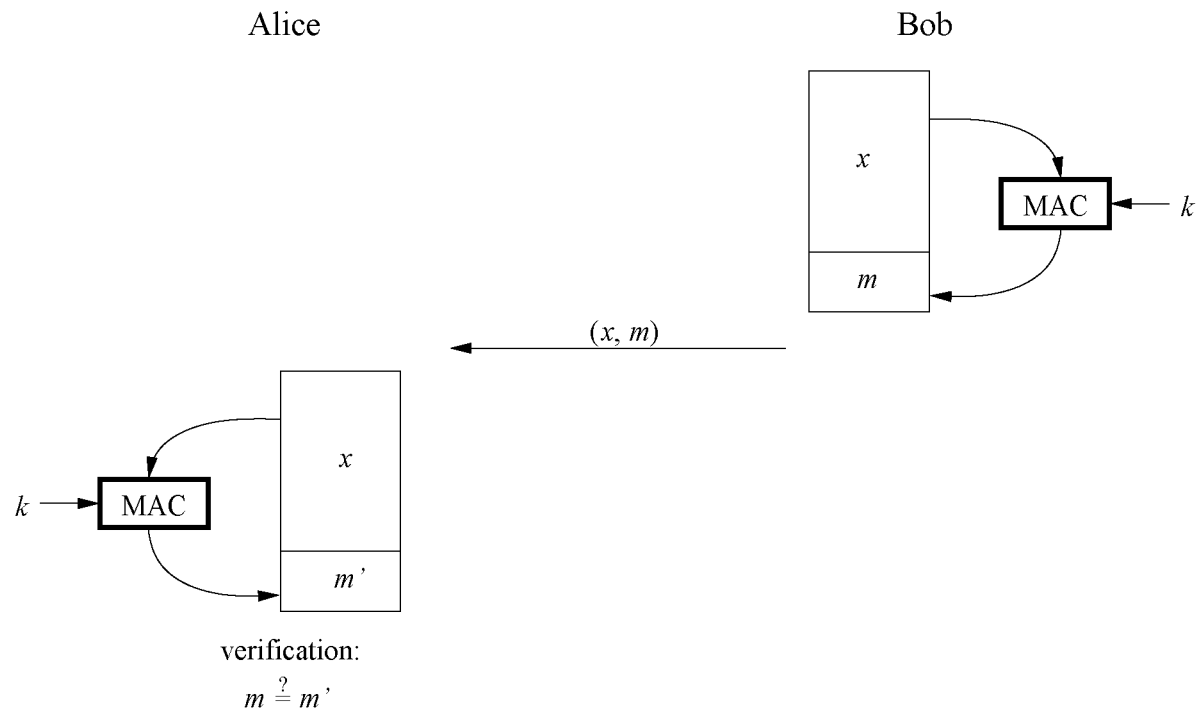
- Overview over many Hash Functions with Spezifikationen:
 - http://ehash.iaik.tugraz.at/wiki/The_Hash_Function_Zoo
- Birthday Paradox: Wikipedia has a nice explanation
 - http://en.wikipedia.org/wiki/Birthday_problem
- SHA Standards
 - SHA1+2: <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf>
 - 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 provide message 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.

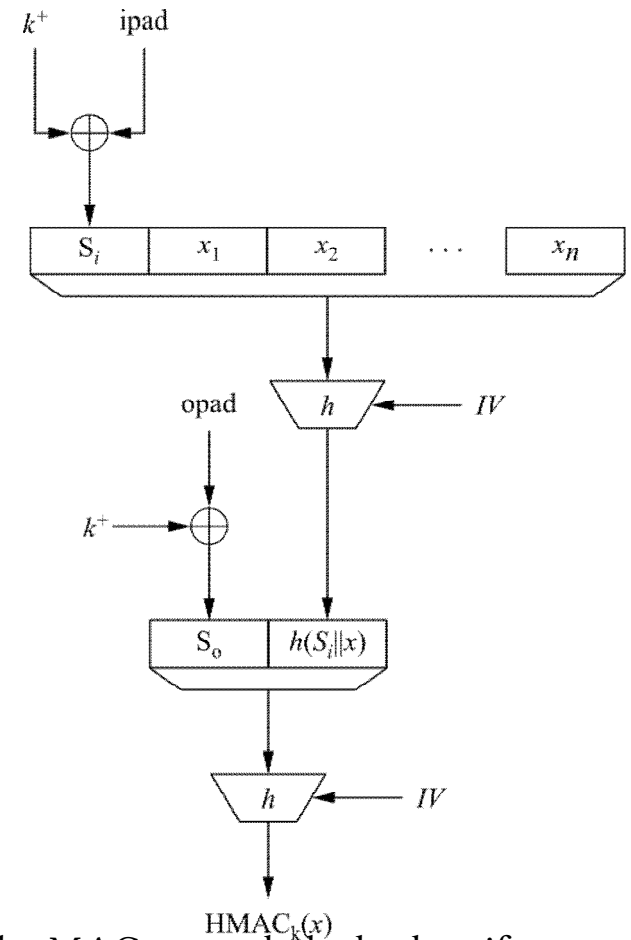
■ MACs from Hash Functions

- 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 $x = (x_1, x_2, \dots, x_n, x_{n+1})$, 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 $m = h(x\|k) = h(x_0\|k)$
- 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

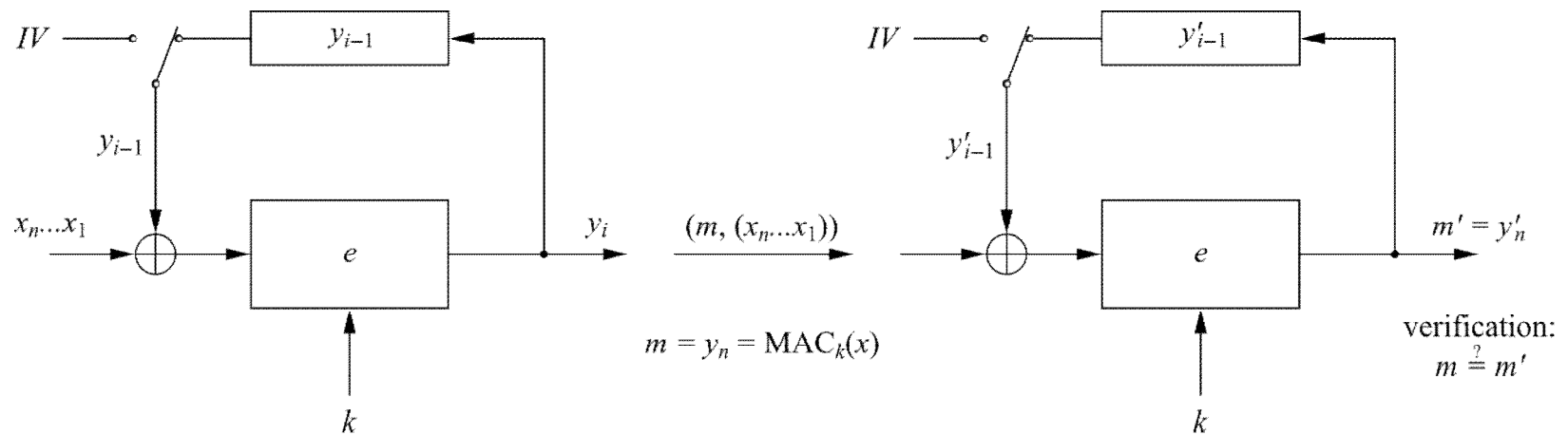
- k^+ is expanded key k
- expanded key k^+ is XORed with the inner pad
- $\text{ipad} = 00110110, 00110110, \dots, 00110110$
- $\text{opad} = 01011100, 01011100, \dots, 01011100$
- $\text{HMAC}_k(x) = h[(k^+ \oplus \text{opad}) \| h[(k^+ \oplus \text{ipad}) \| x]]$



- 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

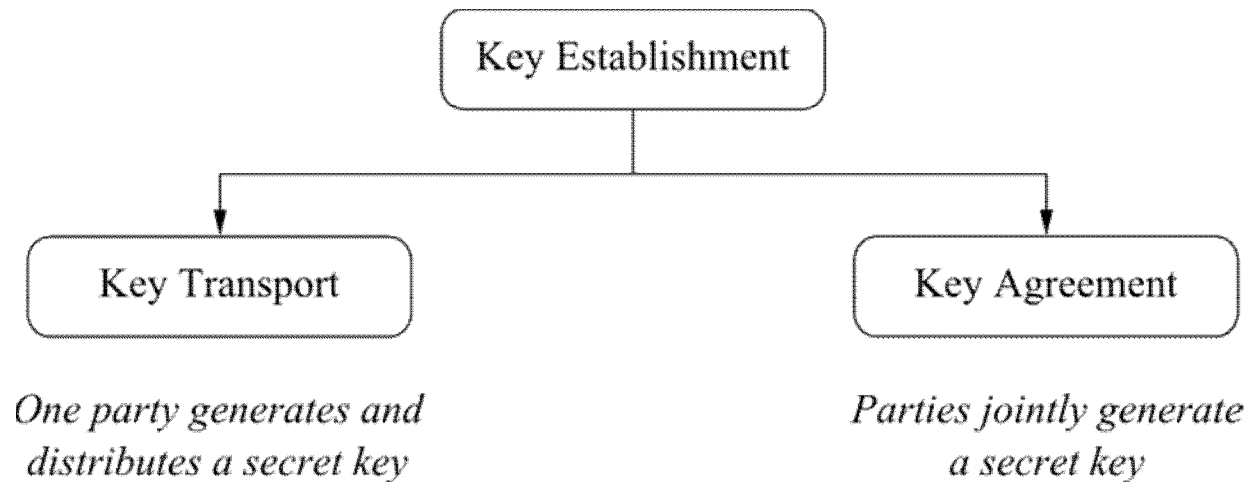
■ Lessons Learned

- 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

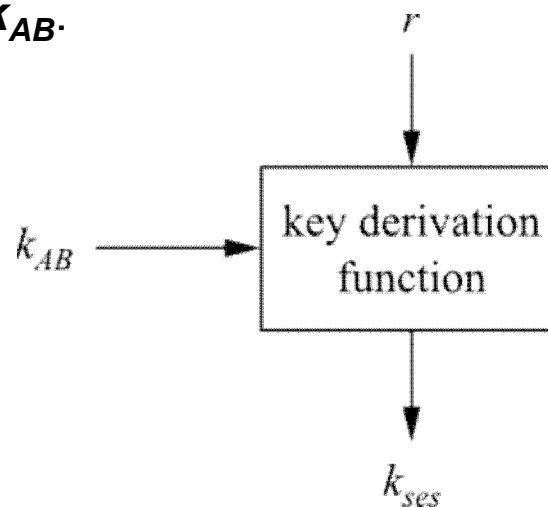
It is often desirable to frequently change the key in a cryptographic system.

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

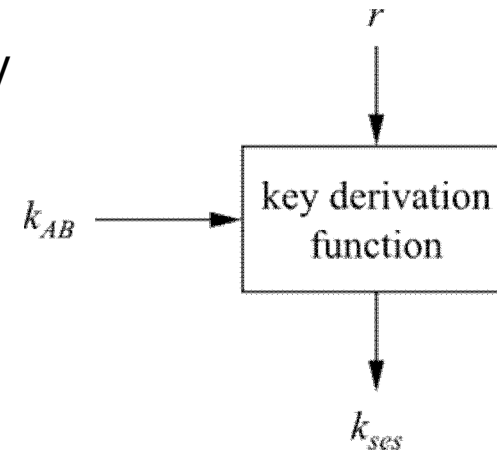
- In order to achieve key freshness, we need to generate new keys frequently.
- Rather than performing a full key establishment every time (which is costly in terms of computation and/or communication), we can **derive multiple session keys k_{ses} from a given key k_{AB}** .



- 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

$$K_{ses} = e_{k_{AB}}(r)$$

derive session key

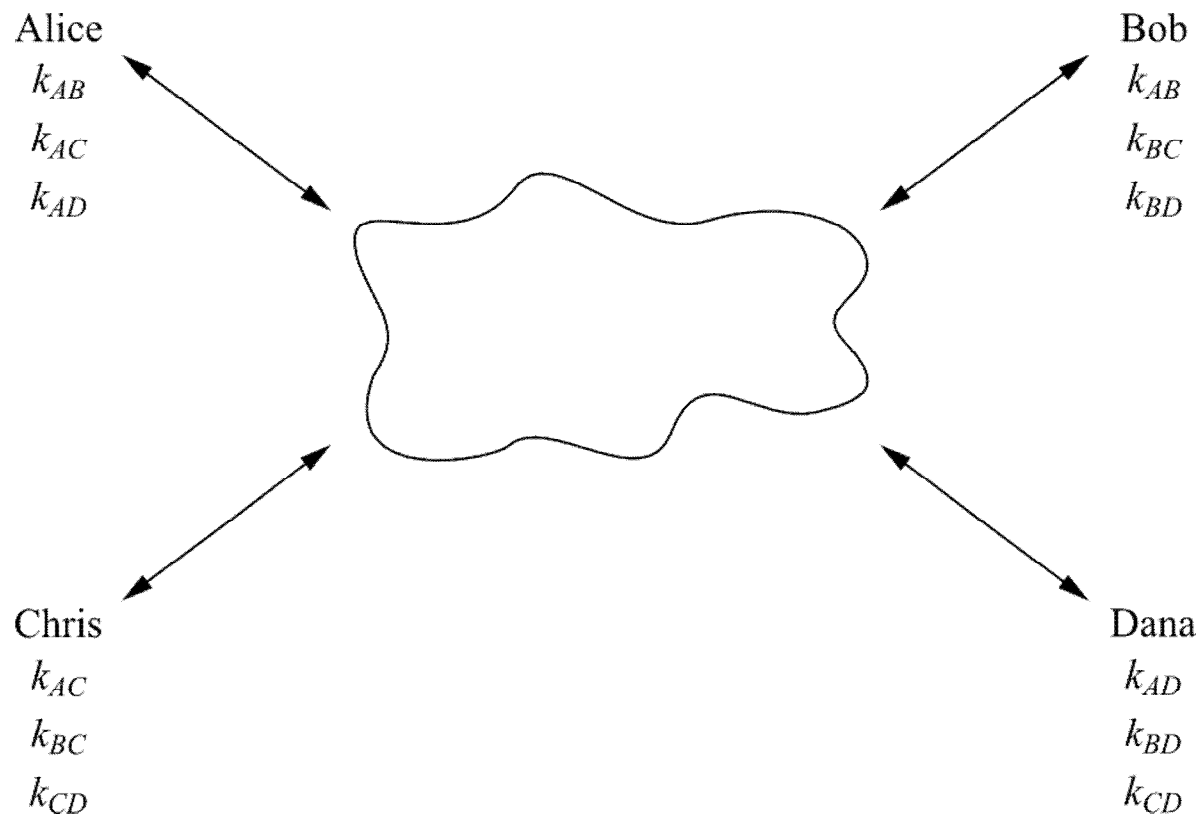
$$K_{ses} = e_{k_{AB}}(r)$$

■ 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

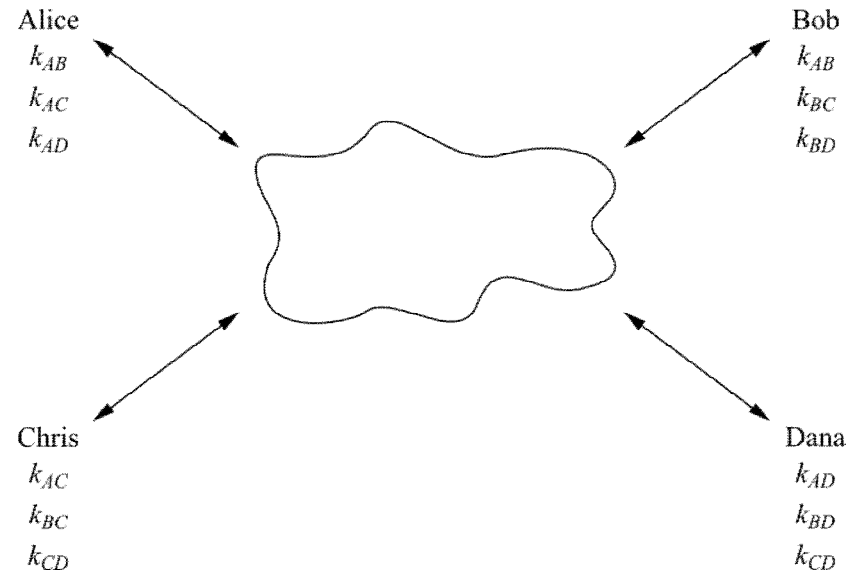
- Simple situation: Network with n users. Every user wants to communicate securely with every of the other $n-1$ users.
- Naïve approach: Every pair of users obtains an individual key pair



■ The n^2 Key Distribution Problem

Shortcomings

- There are $n(n-1) \approx n^2$ keys in the system
 - There are $n(n-1)/2$ 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 users
- ⇒ Only works for small networks which are relatively static



Example: mid-size company with 750 employees

- $750 \times 749 = 561,750$ 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

- 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

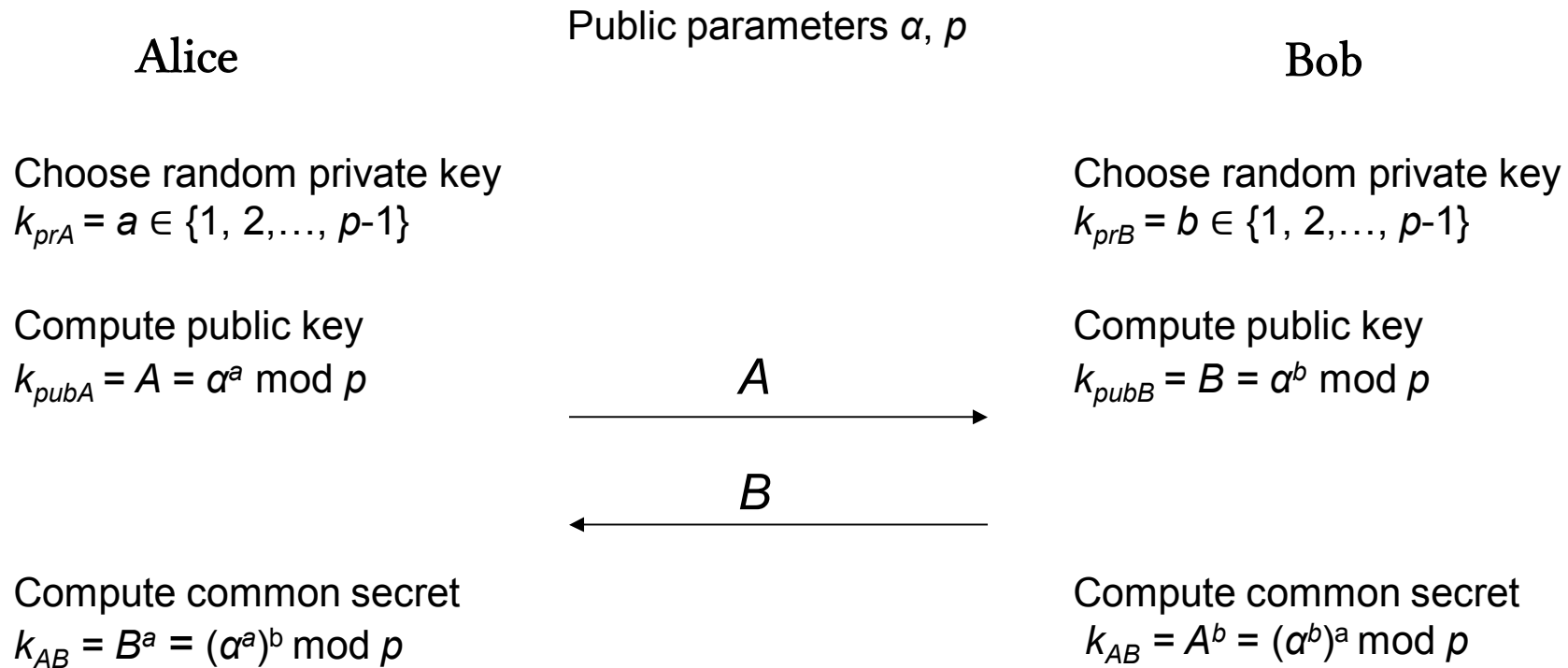
Remaining problems:

- **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

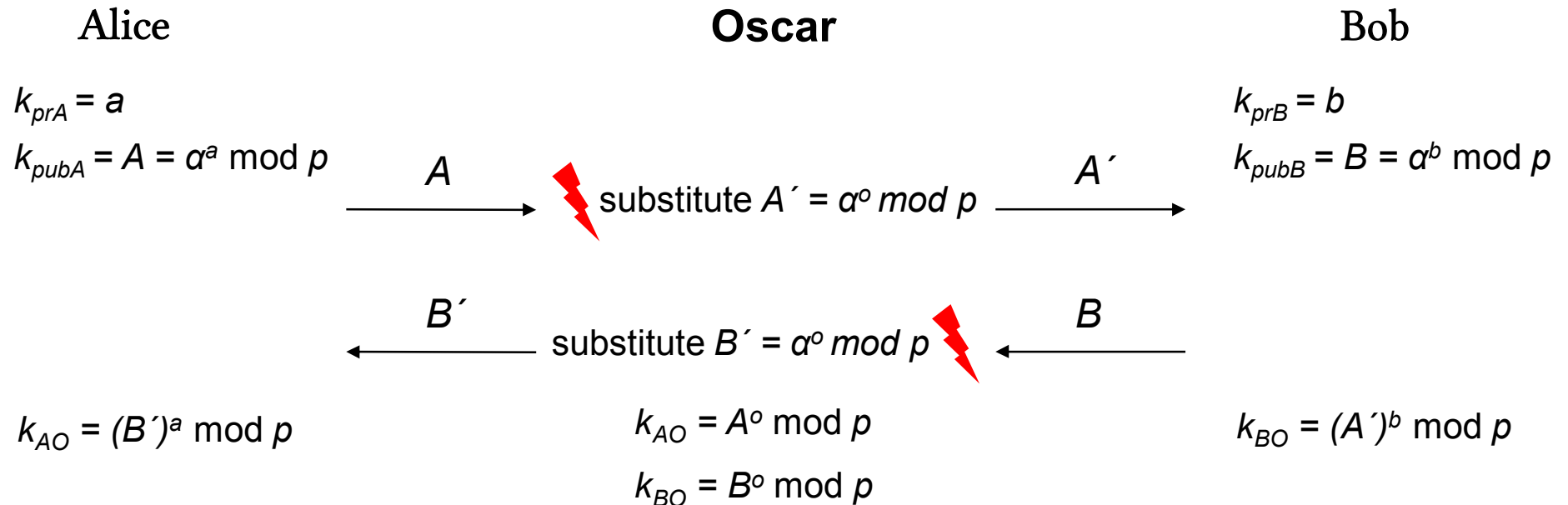
- Introduction
- The n^2 Key Distribution Problem
- Symmetric Key Distribution
- **Asymmetric Key Distribution**
 - **Man-in-the-Middle Attack**
 - Certificates
 - Public-Key Infrastructure

■ 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 communication, 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 communicating with each other !
- The attack efficiently performs 2 DH key-exchanges: Oscar-Alice and Oscar-Bob
- Here is why the attack works:

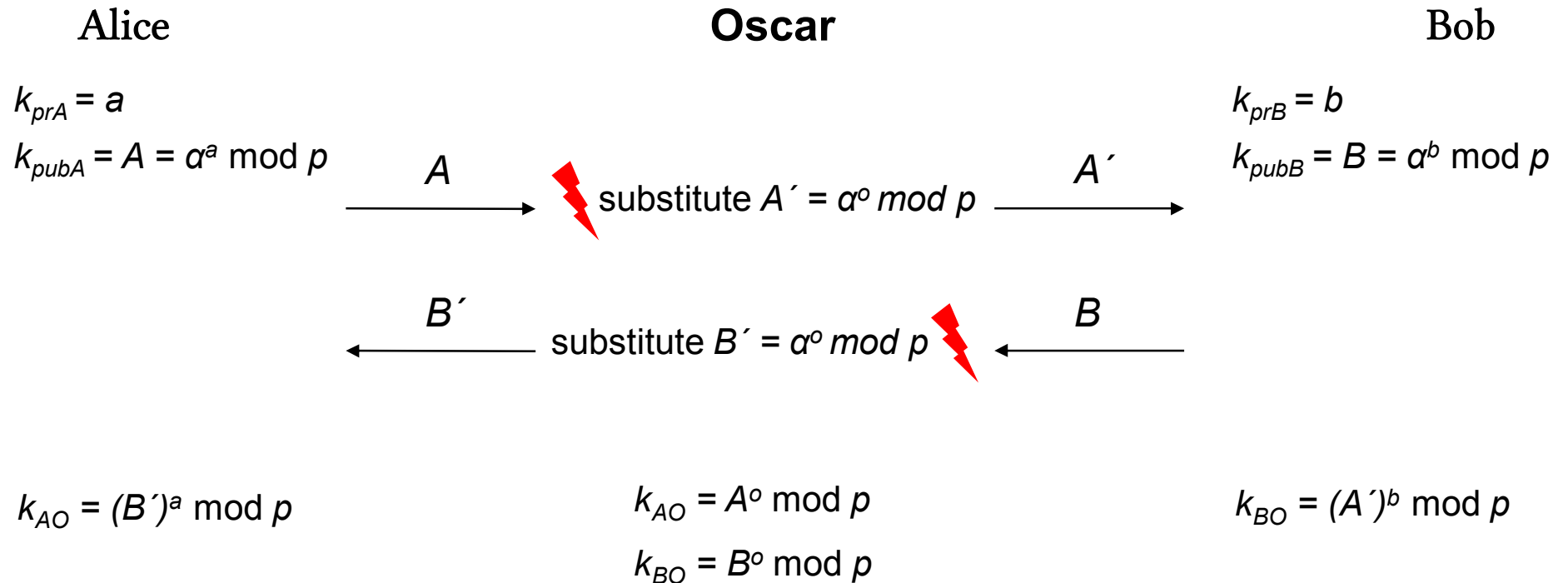
Alice computes: $k_{AO} = (B')^a = (\alpha^0)^a$

Bob computes: $k_{BO} = (A')^b = (\alpha^0)^b$

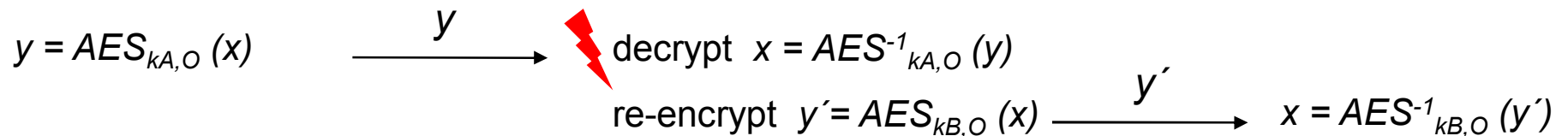
Oscar computes: $k_{AO} = A^0 = (\alpha^a)^0$

Oscar computes: $k_{BO} = B^0 = (\alpha^b)^0$

■ 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*

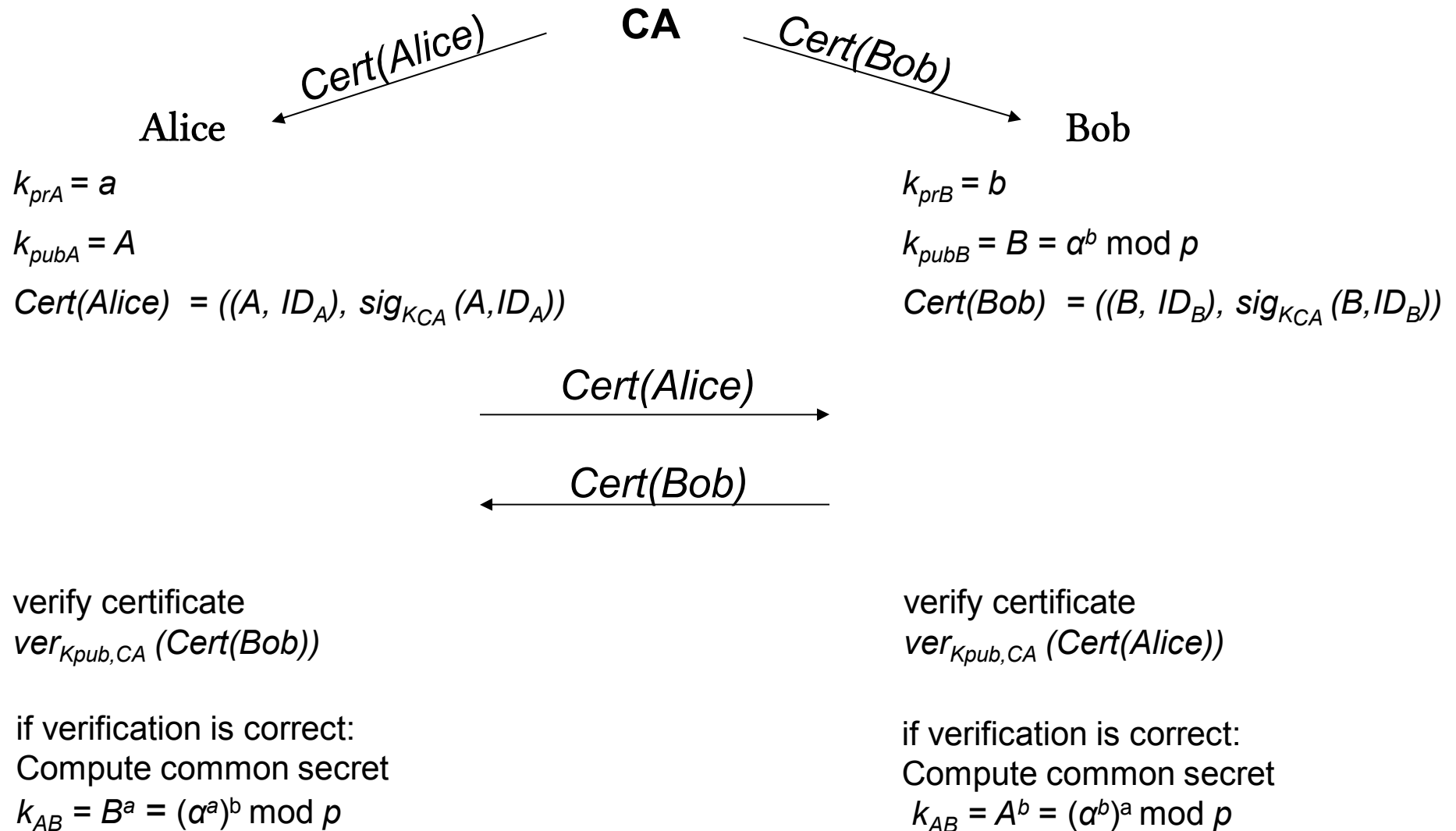
certificate = public key + ID(user) + digital signature over public key and ID

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

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

- 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



■ Certificates

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

■ Public-Key Infrastructure

Definition: The entire system that is formed by CAs together with the necessary support mechanisms is called a public-key infrastructure (PKI).

■ 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 certificate (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*

Serial Number
Certificate Algorithm: - Algorithm - Parameters
Issuer
Period of Validity: - Not Before Date - Not After Date
Subject
Subject’s Public Key: - Algorithm - Parameters - Public Key
Signature

■ 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 revoked, 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*