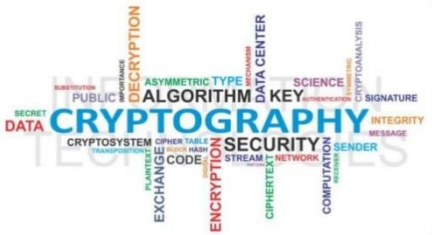
Cryptography & Security

5. Public-key cryptography

Aureliu Zgureanu,
PhD, Associate Professor

Content of this part



- Symmetric Cryptography Revisited
- Principles of Asymmetric Cryptography
- Practical Aspects of Public-Key Cryptography
- Important Public-Key Algorithms
- Essential Number Theory for Public-Key Algorithms
- Important Public-Key Algorithms
- RSA Algorithms
- Diffie–Hellman Key Exchange
- The Discrete Logarithm Problem
- Elliptic Curve Cryptography

Classification Of Ciphers

Cryptographic Ciphers

Classical Ciphers

Modern Ciphers

Are not ciphers!

Substitution Ciphers

Transpositional Ciphers

Secret Cryptography

Public Cryptography

Hash Functions

Single Substitution

Multiple Substituion

Rail Fence Cipher

Block Ciphers

Stream Cipher

Monoalphabetical Ciphers

Polyalphabetical Ciphers

Polygraphic Ciphers

Route Cipher

Synchronize

Self- Synchronize

Atbash Cipher

Vigenere Cipher

Hill Cipher

Columnar Cipher

IDEA

DES

3DES

AES

Caesar Cipher

Gronsfeld Cipher

Double Transpositional Cipher

Integer Factorization Cryptosystems

Discrete Log Cryptosystems

Elliptic Curves Cryptosystems

Affine Cipher

Beaufort Cipher

Myszkowiski Cipher

RSA

ElGamal

Auto Key Cipher

Disrupted Cipher

Rabin

Cramer-Shoup

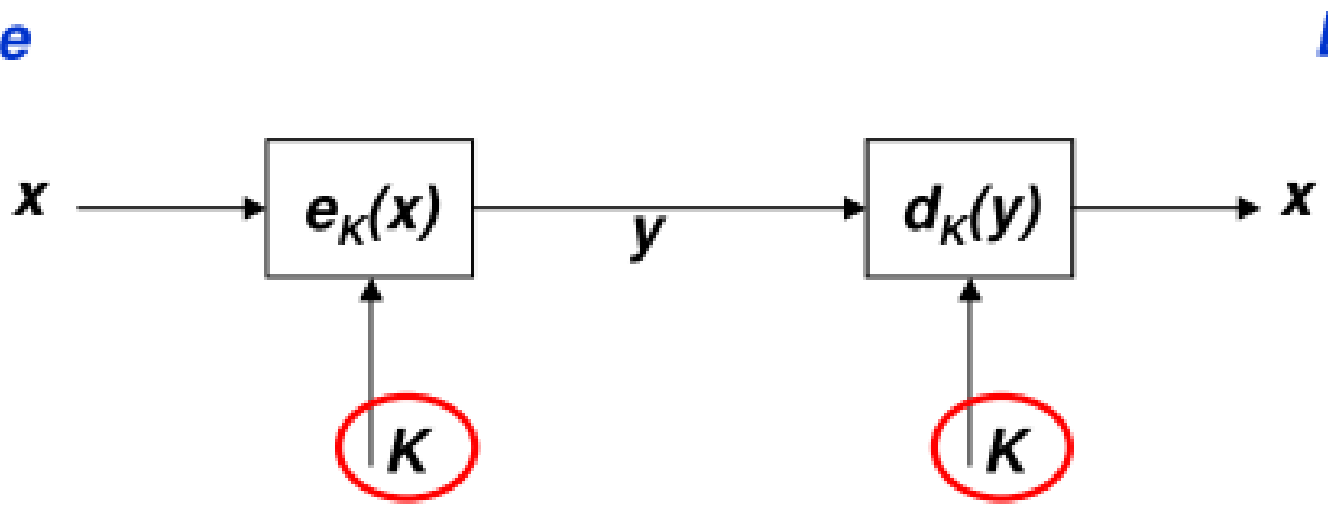
Running Key Cipher

Grilles Cipher

Schmidt-Samoa

Massey-Omura

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

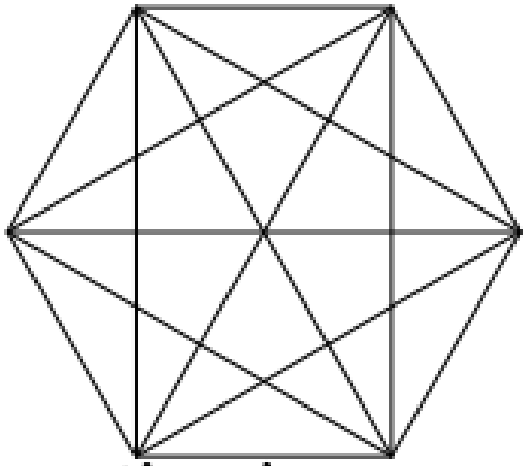
Symmetric Cryptography: Shortcomings

- 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"

Idea behind Asymmetric Cryptography

New Idea:

Use the „good old mailbox“ principle:

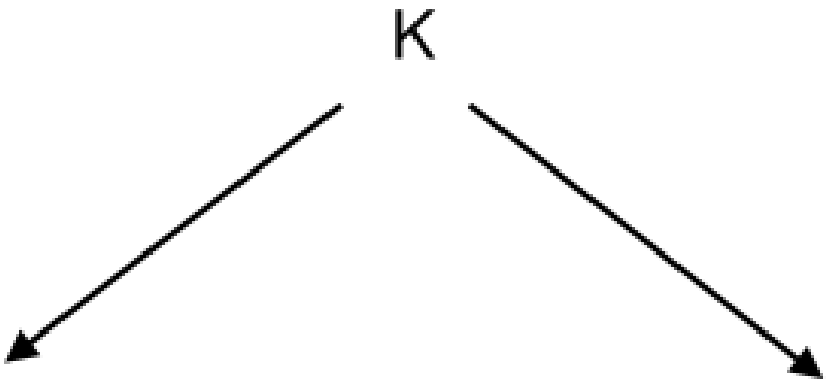
Everyone can drop a letter

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

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

Asymmetric (Public-Key) Cryptography

Principle: "Split up" the key



Public Key (K_{pub})
(Encrypt)

Secret Key (K_{pr})
(Decrypt)

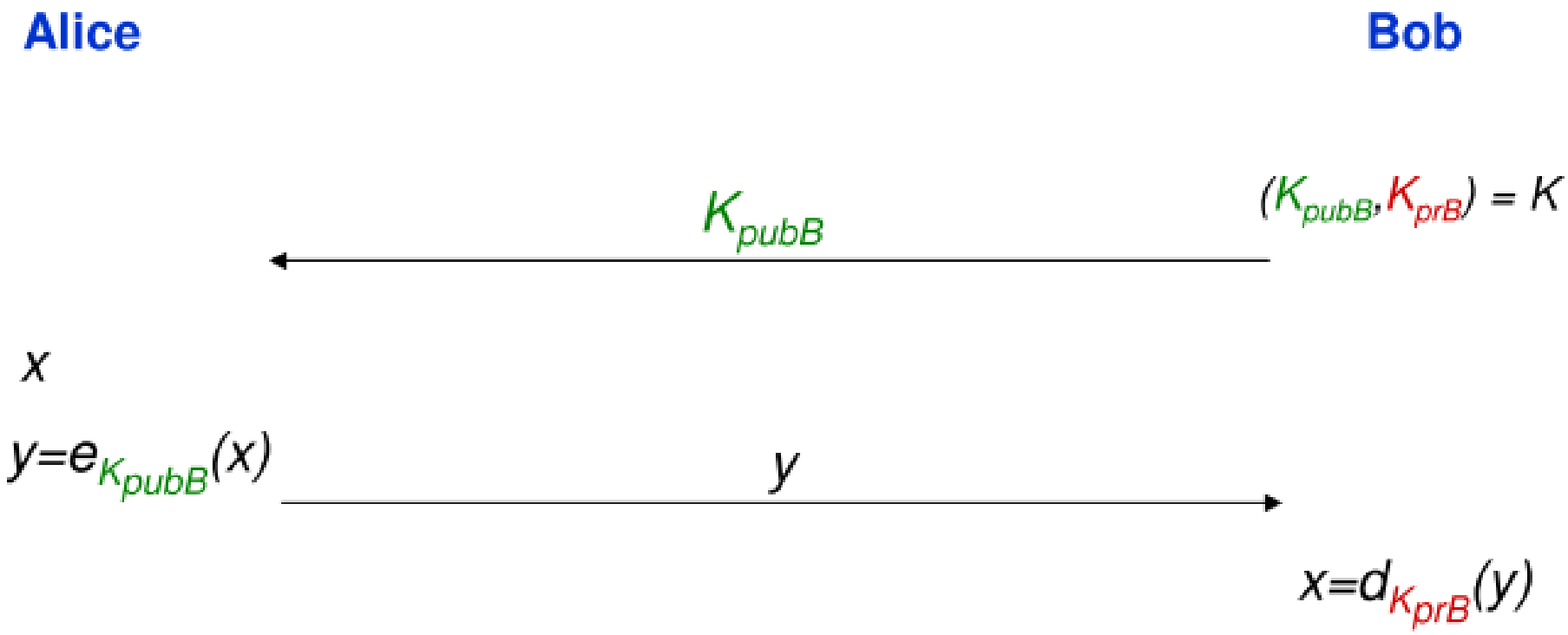
During the key generation, a key pair K_{pub} and K_{pr} is computed

Asymmetric Cryptography: Analogy

Safe with public lock and private lock:

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

Basic Protocol for Public-Key Encryption



→ Key Distribution Problem solved *

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

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

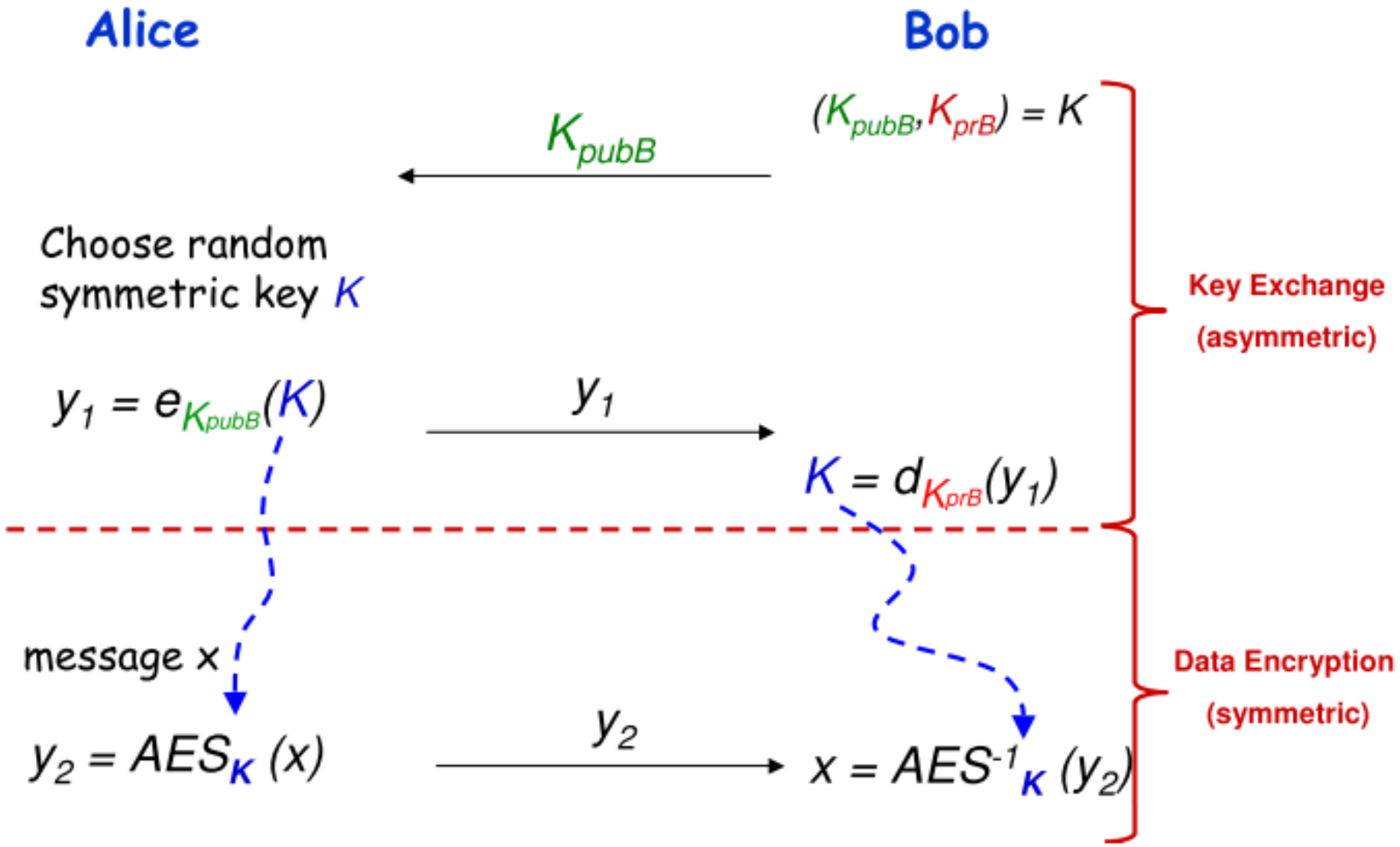
In practice: Hybrid systems, incorporating asymmetric and symmetric algorithms

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



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 very hard

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)
128 Bit	256 Bit	≈ 4096 Bit	Long term security (without quantum computers)

- The exact complexity of RSA (factoring) and DL (Index-Calculus) is difficult to estimate
- The development of quantum computers would probably be the end for ECC, RSA & DL

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:

$$r_0 = 84 = \textcircled{2} \cdot 2 \cdot \textcircled{3} \cdot 7$$
$$r_1 = 30 = \textcircled{2} \cdot \textcircled{3} \cdot 5$$

→ The gcd is the product of all common prime factors:
$2 \cdot 3 = 6 = gcd\,(30,84)$

- But: Factoring is very complicated for large numbers

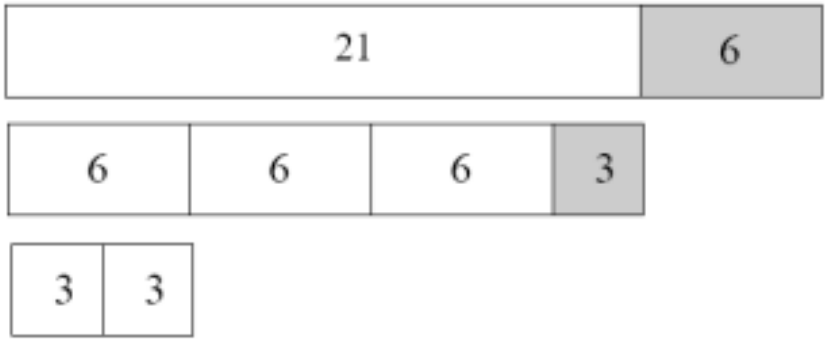
Euclidean Algorithm 2/2

- 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 $r_1 = 21$



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

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

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

Very efficient method even for long numbers: complexity grows *linearly* with number of bits

$973 = 3 \cdot 301 + 70$	$\gcd(973,301) = \gcd(301,70)$
$301 = 4 \cdot 70 + 21$	$\gcd(301,70) = \gcd(70,21)$
$70 = 3 \cdot 21 + 7$	$\gcd(70,21) = \gcd(21,7)$
$21 = 3 \cdot 7 + 0$	$\gcd(21,7) = \gcd(7,0) = 7$

Extended Euclidean Algorithm (EEA)

- Extend the Euclidean algorithm to find modular inverse of $r_1 \bmod r_0$
- EEA computes s, t, and the gcd: $\gcd(r_0, r_1) = s \cdot r_0 + t \cdot r_1$
- $gcd(r_0, r_1) = 1$ in order for the inverse to exist

$$s \cdot r_0 + t \cdot r_1 \equiv 1 \equiv \gcd(r_0, r_1)$$

- Reduce the equation mod r_0 :

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

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

- t is the inverse of $r_1 \bmod r_0$
- EEA uses recursive formulae to calculate s and t in each step
  - Express current remainder r_i as $r_i = s_i r_0 + t_i r_1$.
  - Last iteration:

$$r_l = \gcd(r_0, r_1) = s_l r_0 + t_l r_1 = s r_0 + t r_1$$

Extended Euclidean Algorithm - Example

gcd(973,301)=7

r_0=973; r_1=301

s=13; t=-42

$973 = 3 \cdot 301 + 70$	$70 = [1]r_0 + [-3]r_1$
$301 = 4 \cdot 70 + 21$	$21 = 301 - 4 \cdot 70$ $= r_1 - 4(1r_0 - 3r_1)$ $= [-4]r_0 + [13]r_1$
$70 = 3 \cdot 21 + 7$	$7 = 70 - 3 \cdot 21$ $= (1r_0 - 3r_1) - 3(-4r_0 + 13r_1)$ $= [13]r_0 + [-42]r_1$
$21 = 3 \cdot 7 + 0$	

$$\gcd(973,301) = 7 = [13]973 + [-42]301 = 12649 - 12642$$

EEA can be expressed using recursive formulae for s_i, t_i

$$s_0 = 1 \quad t_0 = 0$$
$$s_1 = 0 \quad t_1 = 1$$

1 DO
1.1 $i = i + 1$
1.2 $r_i = r_{i-2} \bmod r_{i-1}$
1.3 $q_{i-1} = (r_{i-2} - r_i)/r_{i-1}$
1.4 $s_i = s_{i-2} - q_{i-1} \cdot s_{i-1}$
1.5 $t_i = t_{i-2} - q_{i-1} \cdot t_{i-1}$
WHILE $r_i \neq 0$

EEA – calculating a modular inverse

Example: Calculate the modular inverse of 12 mod 67:

- Using EEA we obtain $-5 \cdot 67 + 28 \cdot 12 = 1$
- Hence 28 is the inverse of 12 mod 67.

gcd(67,12)=gcd(12,7)=gcd(5,2)=gcd(2,1)

67=12•5+7 ⇒7=(1)67+(-5)12

12=7+5 ⇒5=12-7=(-1)67+(6)12

7=5+2 ⇒2=7-5=(2)67+(-11)12

5=2•2+1 ⇒1=5-2•2=(-5)67+(28)12

$s_0 = 1 \quad t_0 = 0$

$s_1 = 0 \quad t_1 = 1$

1 DO

1.1 $i = i + 1$

1.2 $r_i = r_{i-2} \bmod r_{i-1}$

1.3 $q_{i-1} = (r_{i-2} - r_i)/r_{i-1}$

1.4 $s_i = s_{i-2} - q_{i-1} \cdot s_{i-1}$

1.5 $t_i = t_{i-2} - q_{i-1} \cdot t_{i-1}$

WHILE $r_i \neq 0$

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**

r_1 mod r_0 =12 mod 67

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

EEA – calculating a modular inverse and GCD scheme (another view)

i	s	t	r	q
1	1	0	$r_1 = n$	
2	0	1	$r_2 = a$	$q_2 = \left[\frac{r_1}{r_2}\right]$
3	$s_i = s_{i-2} - s_{i-1} \cdot s_{i-1}$	$t_i = t_{i-2} - q_{i-1} \cdot t_{i-1}$	$r_i = r_{i-2} \, mod \, r_{i-1}$	$q_3 = \left[\frac{r_2}{r_3}\right]$
…	…	…	…	…
k	$s = s_{k-2} - q_{k-1} \cdot s_{k-1}$	$t_k = t_{k-2} - q_{k-1} \cdot t_{k-1}$	1	$q_k = \left[\frac{r_{k-1}}{r_k}\right]$
	GCD	a^{-1}	Calculating $a^{-1} mod \, n$	

Euler's Phi Function 1/2

- *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 $\Phi(m)$ *(totient function)*
- Example: sets {0,1,…,5} (*m*=6), and {0,1,…,4} (*m*=5)

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

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

→ $\Phi(5) = 4$; $\Phi(6) = 2$

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

Euler's Phi Function 2/2

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

$$\Phi(m) = \prod_{i=1}^{n} (p_i^{e_i} - p_i^{e_i - 1})$$

- Phi especially easy for $e_i = 1$, e.g., $m = p \cdot q \Rightarrow \Phi(m) = (p-1) \cdot (q-1)$
- Examples: $m=6=3 \cdot 2 \Rightarrow \Phi(6) = (3-1)(2-1)=2$

  $m = 899 = 29 \cdot 31$:
  $\Phi(899) = (29-1) \cdot (31-1) = 28 \cdot 30 = 840$
- Note: Finding $\Phi(m)$ is computationally easy if factorization of m is known
  (otherwise the calculation of $\Phi(m)$ is computationally very hard 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 $a^{p-1} \equiv 1 \pmod{p}$

$$a \cdot a^{p-2} \equiv 1 \pmod{p}$$

- Use: Find modular inverse, if p is prime.
- Comparing with definition of the modular inverse

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

the modular inverse modulo a prime p *is*

$$a^{-1} \equiv a^{p-2} \pmod{p}$$

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

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

- Fermat's Little Theorem works only modulo a prime p

Euler's Theorem

- Generalization of Fermat's little theorem to any integer modulus
- Given two relatively prime integers *a* and *m*:
- Example: *m*=18, *a*=5

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

1. Calculate Euler's Phi Function

$$\Phi(18) = \Phi(3^2 \cdot 2) = (3^2 - 3^1)(2^1 - 2^0) = 6$$

2. Verify Euler's Theorem

$$5^{\Phi(18)} = 5^6 = 25^3 = 7^3 \bmod 18 \Rightarrow 7^3 = 343 = 18 \cdot 19 + 1 = 1 \bmod 18$$

- 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.

Content of this Chapter

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

The RSA Cryptosystem

- Hellman and Diffie published their landmark public-key paper in 1976
- Rivest, Shamir and Adleman proposed the asymmetric RSA cryptosystem in 1977
- 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
  - Transmission of (i.e., symmetric) keys
  - Digital signatures

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 large primes
- Encryption and decryption are 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 integers (≥ 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

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

Example: RSA with small numbers

ALICE

Message $x = 9$

BOB

1. Choose $p = 7$ and $q = 11$
2. Compute $n = p * q = 77$
3. $\Phi(n) = (7-1) * (11-1) = 60$
4. Choose $e = 7$; $gcd(7,60)=1$
5. $d \equiv e^{-1} \equiv 43 \bmod 60$

$7 \times 43 = 301 = 1 \bmod 60$

(use EEA to get d)

$K_{pub} = (77,7)$ ←

$y = x^e = 9^7 = 4782969 \equiv 37 \bmod 77$

$y = 37$ →

$y^d = 37^{43} \equiv 9 \bmod 77$

Proof

- Show that $x = y^d \bmod n = (x^e)^d \bmod n$
- Case 1: $\gcd(x,n)=1$
- 1. Since $d \cdot e \equiv 1 \bmod \Phi(n)$ we can write $d \cdot e = 1 + t \cdot \Phi(n)$
- 2. $(x^e)^d \equiv x^{ed} \equiv x^{1+t\Phi(n)} \equiv (x^{\Phi(n)})^t \cdot x \bmod n$
- 3. Based on Euler's theorem if $\gcd(x,n)=1$ then $1 \equiv x^{\Phi(n)} \bmod n$
- 4. Thus, $(x^{\Phi(n)})^t \equiv 1^t \bmod n \equiv 1 \bmod n$
- Case 2: $\gcd(x,n) \neq 1$ then $x=a \cdot p$ (or $a \cdot q$)
- 1. and 2. above
- 3a. $x^{\Phi(q)} = x^{q-1} = 1 \bmod q$ since $\gcd(x,q)=1$ (Euler's theorem)
- 3b. $(x^{\Phi(n)})^t = ((x^{q-1})^{p-1})^t = 1 \bmod q$ thus $(x^{\Phi(n)})^t = 1+b \cdot q$
- 4. $(x^{\Phi(n)})^t \cdot x = (1+b \cdot q) \cdot x = x+x \cdot b \cdot q = x+ a \cdot p \cdot b \cdot q = x \bmod n$

Implementation aspects

- RSA uses only one arithmetic operation (modular exponentiation) which makes it conceptually simple
- But, 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 implementation algorithm
- 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

Input: Exponent H, base element x, Modulus n

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

1. Determine binary representation $H = (h_t, h_{t-1}, \ldots, h_0)_2$
2. FOR $i = t\text{-}1$ 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 rep. 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 $x^{26} = x^{11010}$
- 6 operations instead of 25 multiplications

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 or 4096 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
- Some important tricks:
  - Short public exponent *e*
  - Chinese Remainder Theorem (CRT)
  - Exponentiation with pre-computation *(not covered here)*

https://en.wikipedia.org/wiki/Chinese_remainder_theorem#:~:text=In%20mathematics%2C%20the%20Chinese%20remainder,are%20pairwise%20coprime%20(no%20two

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

- Small Hamming weight → https://en.wikipedia.org/wiki/Hamming_weight#:~:text=The%20Hamming%20weight%20of%20a,string%20of%20the%20same%20length.
- Commonly used trick (e.g., SSL/TLS, etc.): makes RSA the fastest asymmetric scheme with regard to encryption

Fast decryption with CRT

- Choosing a small private key d results in security weakness
  - 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 accelerate exponentiation with the private key d
- Based on the CRT we can replace the computation of

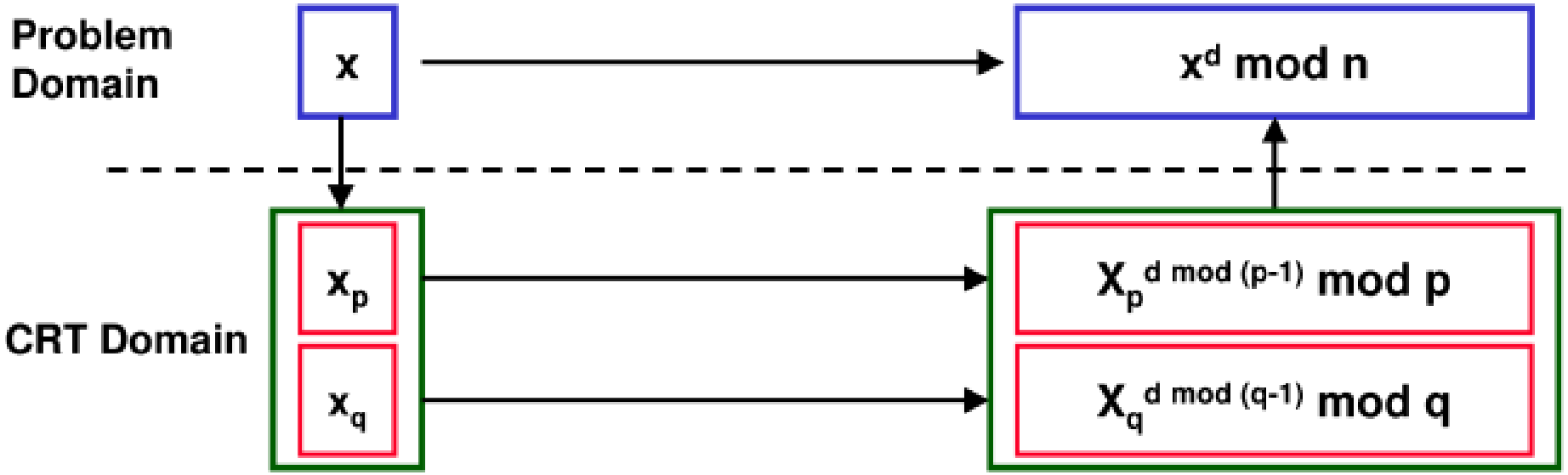
$$x^{d \bmod \Phi(n)} \bmod n$$

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-based exponentiation 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

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

CRT: Step 2 – Exponentiation

- Given d_p and d_q such that

  $d_p \equiv d \bmod (p-1)$ and $d_q \equiv d \bmod (q-1)$

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

  $y_p \equiv x_p^{d_p} \bmod p$ and $y_q \equiv x_q^{d_q} \bmod q$

- 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

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

$$y \equiv [\, q * q^{-1} \bmod p \,] * y_p + [\, p * p^{-1} \bmod q \,] * y_q \bmod n$$

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

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

can be precomputed and stored

Example

p=13, q=11, n=pq=143; $\Phi(n)$ = (p-1)(q-1)=120=8•3•5

Select e=77 as gcd(77,120)=1; d=53 mod 120 as
77 • 53=1 mod 120

Encryption: Given x=101 then y= $x^e = 101^{77} = 101^{1001101}$ =
$((((101^2)^2)^2 101)^2 101)^2)^2 101$ mod 143 = 95 mod 143

Decryption:

$$x_p = y^{d \bmod (p-1)} \bmod p = 95^{53 \bmod 12} \bmod 13 = 4^5 \bmod 13 = 10 \bmod 13$$

$$x_q = y^{d \bmod (q-1)} \bmod q = 95^{53 \bmod 10} \bmod 11 = 7^3 \bmod 11 = 2 \bmod 11$$

$c_p \equiv q^{-1} \bmod p \equiv 11^{-1} \bmod 13 = 6 \bmod 13$ since $11*6 \equiv 1 \bmod 13$

$c_q \equiv p^{-1} \bmod q \equiv 13^{-1} \bmod 11 = 6 \bmod 11$ since $13*6 \equiv 1 \bmod 11$

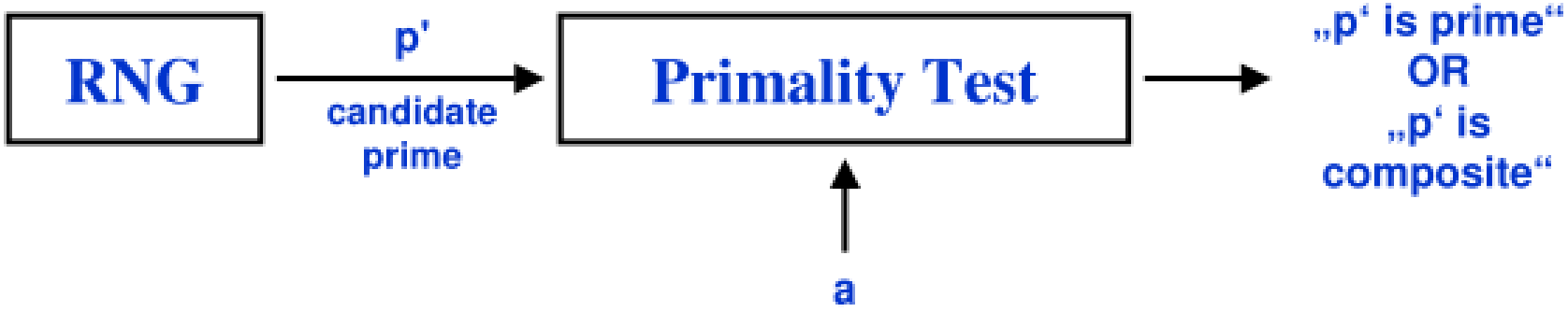
x=11*6*10+13*6*2=66*10+78*2 mod 143=88+13 mod 143=101

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, p and q are about $t/2$ bits long
- The complexity is determined by the two exponentiations (using the square-and-multiply algorithm) in the CRT domain. The operands are only $t/2$ bits long:
  - # 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 exponentiations, but since the operands have half the bit length compared to regular exponent, each operation (i.e., multiplying and squaring) is 4 times faster
- Hence CRT is 4 times faster than straightforward exponentiation

Finding Large Primes

- Generating keys for RSA requires finding two large primes *p* and *q* such that $n = p * q$ is sufficiently large
- Size of *p* and *q* is typically half 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 hard
- 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

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

Output: *p'* is composite or 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 („Carmichael numbers") this test often returns „*p* is likely a prime" – although they are composite
- Therefore, the Miller-Rabin Test is preferred
- Example: 561=3*11*17; $a^{560} \equiv 1 \bmod 561$ for all gcd(a,561)=1

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

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

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

1. FOR *i = 1* TO *s*
2. choose random $a \in \{2,3, ..., p'-2\}$
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 („Carmichael numbers") this test often returns „*p* is likely a prime" – although they are composite
- Therefore, the Miller-Rabin Test is preferred
- Example: 561=3*11*17; $a^{560} \equiv 1 \bmod 561$ for all gcd(a,561)=1

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

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

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

Otherwise it is probably a prime.

- This theorem can be turned into an 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"

FOR $i = 1$ TO s
  choose random $a \in \{2, 3, \ldots, p'-2\}$
  $z \equiv a^r \bmod p'$
  IF $z \neq 1$ AND $z \neq p'-1$ THEN
    FOR $j = 1$ TO $u-1$
      $z \equiv z^2 \bmod p'$
      IF $z = 1$ THEN
        RETURN „p' is composite"
    IF $z \neq p'-1$ THEN
      RETURN „p' is composite"
RETURN „p' is likely a prime"

Bit lengths of p	Security parameter s
250	11
300	9
400	6
500	5
600	3

Attacks and Countermeasures

- 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

Analytical Attacks: (1) Mathematical attacks

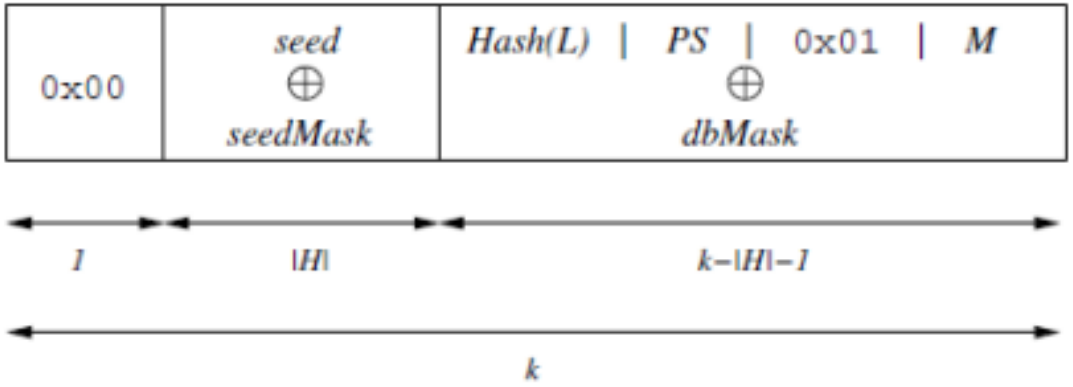
- The best known attack is factoring of n to obtain $\Phi(n)$
- Can be prevented using a sufficiently large n - current factoring record is 664 bits. Thus, n should have between 1024 and 3072 bits

Analytical Attacks

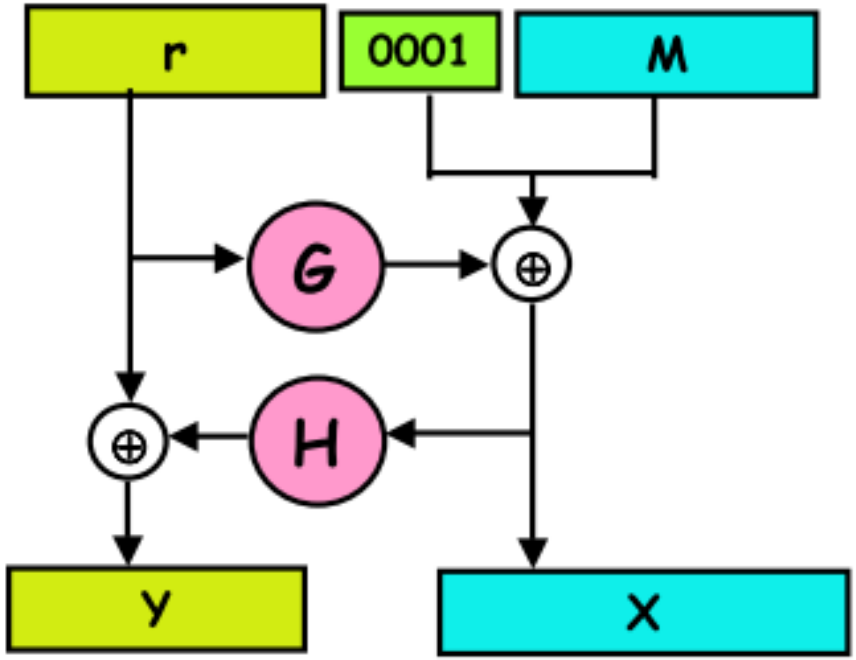
(2) Protocol attacks

- Exploit the malleability of RSA, e.g., a ciphertext can be modified without knowing the private key – Oscar can replace ciphertext y by $y \cdot s^e$ with an integer s resulting in the decrypted message $x \cdot s$
- RSA is deterministic: identical plaintexts ⇒ same ciphertext
- x=0,1,-1 produce y=0,1,-1
- Can be prevented by proper padding (using random numbers)
- Standard padding: Optimal Asymmetric Encryption Padding (OAEP)
- Force all messages to be encrypted to have the maximum length, i.e., k bytes where k is the length of the modulus n in bytes
- Include a random byte string

Optimal Asymmetric Encryption Padding

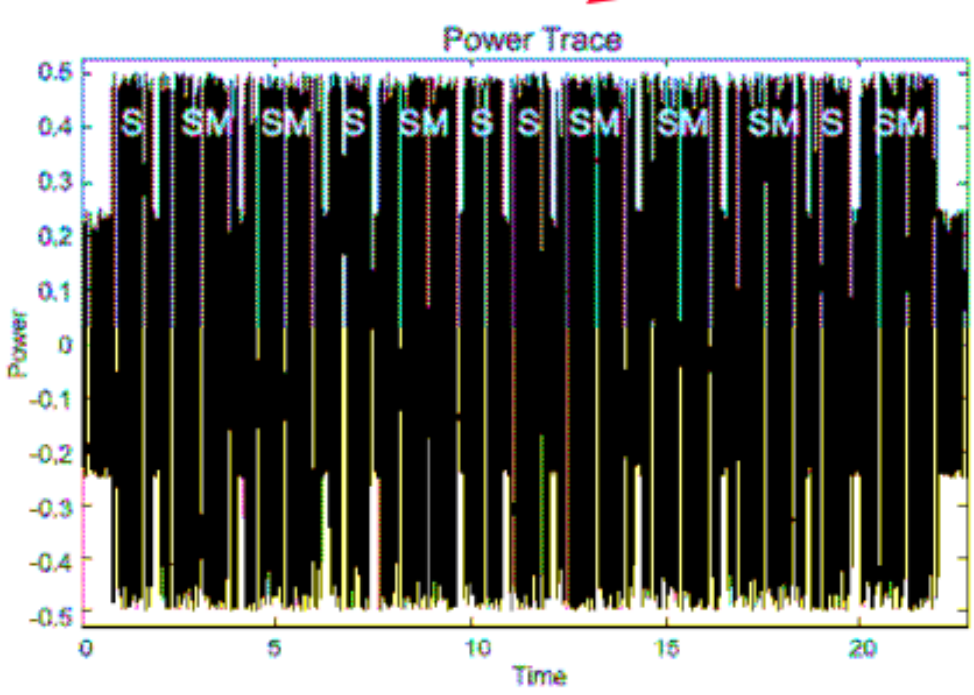


- Hash(L) | zero bytes | 01 | M
  where L = optional message label
- Hash(L) has a fixed length
- G expands the bits of the random number to the required number
- Decode:
- Recover r=Y ⊕ H(x)
- Data = X ⊕ G(r)



Implementation Attacks

- 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



Lessons Learned

- RSA is still 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 almost 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 naive 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

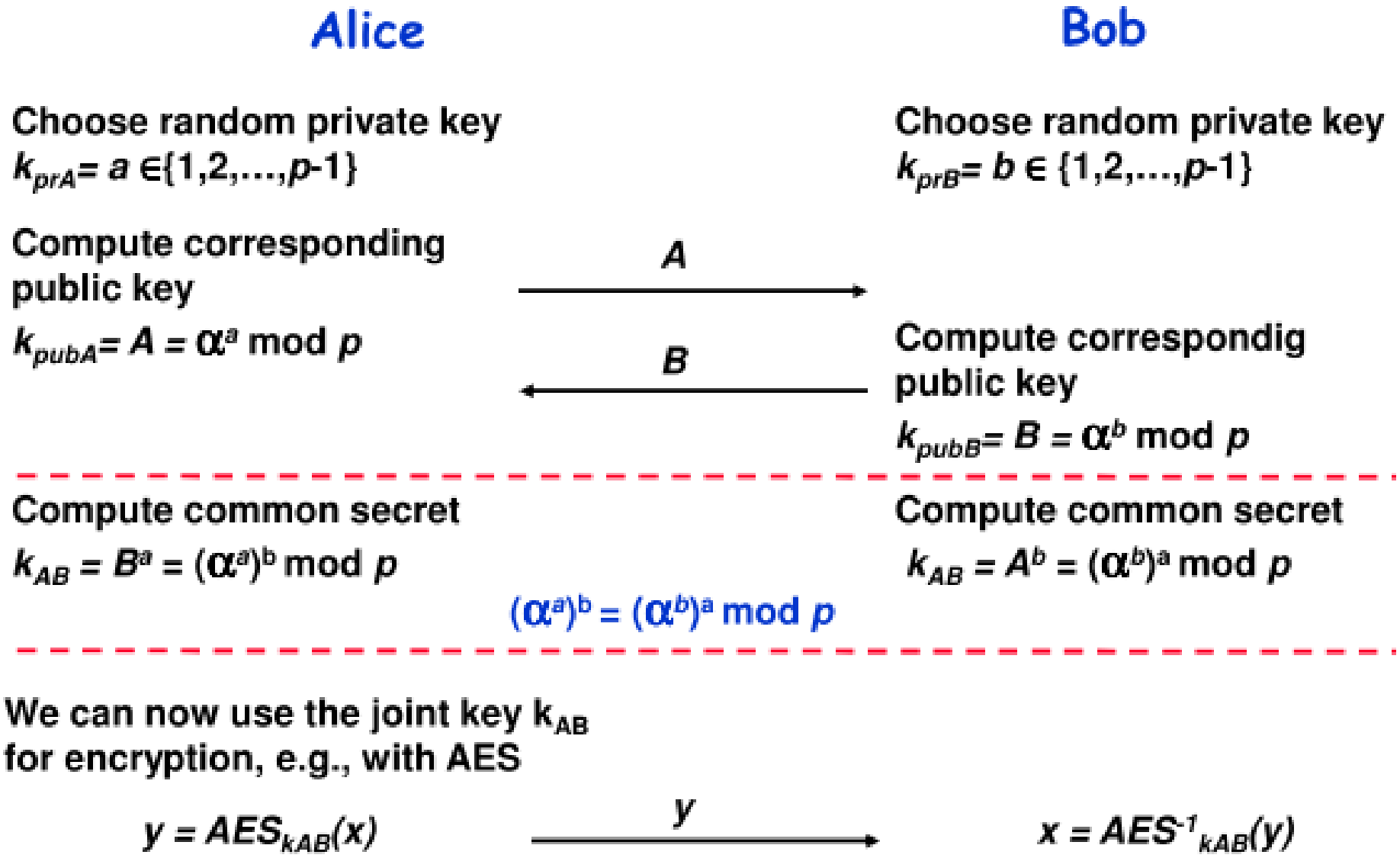
- Proposed in 1976 by Diffie and Hellman
- Widely used, e.g. in Secure Shell (SSH), Transport Layer Security (TLS), and Internet Protocol Security (IPSec)
- The Diffie–Hellman Key Exchange (DHKE) is a key exchange protocol and not used for encryption

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

Setup:

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



Diffie-Hellman Key Exchange: Example

Domain parameters $p=29$, $\alpha=2$

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

Alice and Bob compute the same key k_{AB}

p is large so k_{AB} is large too

Use the 128 MSBs (for AES) or hash to 128 bits

The Discrete Logarithm (DLP) Problem in Z_p^*

Given: 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 $1 \le x \le p-1$ such that $\alpha^x \equiv \beta \bmod p$
- DLP: find $x = \log_\alpha \beta \bmod p$
- Example: Compute x for $5^x \equiv 41 \bmod 47$

- $5^{46} \equiv 1 \bmod 47$
- Ord(5)=46

Cyclic groups and primitive elements

* The group $Z_p^* = \{0,1,\ldots,p-1\}$ for p prime is commutative under multiplication mod p
* The order of an element α, ord(α), of a group with the operation ° is the smallest integer k such that $\alpha^k = \underbrace{\alpha \circ \alpha \circ \ldots \circ \alpha}_{k \text{ times}} = 1 \bmod p$

- Example: a=3 in Z_{11}^*
- Ord(3)=5

$$
\begin{aligned}
a^1 &= 3 \\
a^2 &= a \cdot a = 3 \cdot 3 = 9 \\
a^3 &= a^2 \cdot a = 9 \cdot 3 = 27 \equiv 5 \bmod 11 \\
a^4 &= a^3 \cdot a = 5 \cdot 3 = 15 \equiv 4 \bmod 11 \\
a^5 &= a^4 \cdot a = 4 \cdot 3 = 12 \equiv 1 \bmod 11
\end{aligned}
$$

- The cardinality |G| of the group $G=Z_p^*$ is the number of elements in G such that gcd(i,p)=1, i.e., $|G|=\Phi(p)=p-1$
- A group G that contains an element α such that ord(α)=|G| (cardinality of G) is called cyclic; such α is called a primitive element (or generator) of G
- Example: a=2 in Z_{11}^*
- Ord(2)=10
- All elements of Z_{11}^* are generated

$$
\begin{aligned}
a &= 2 & a^6 &\equiv 9 \bmod 11 \\
a^2 &= 4 & a^7 &\equiv 7 \bmod 11 \\
a^3 &= 8 & a^8 &\equiv 3 \bmod 11 \\
a^4 &\equiv 5 \bmod 11 & a^9 &\equiv 6 \bmod 11 \\
a^5 &\equiv 10 \bmod 11 & a^{10} &\equiv 1 \bmod 11
\end{aligned}
$$

Cyclic groups and primitive elements

- For every element a in a cyclic group G, ord(a) divides |G| and $a^{|G|}=1$
- Example: Z_{11}^*

$$\begin{aligned} \text{ord}(1) &= 1 & \text{ord}(6) &= 10 \\ \text{ord}(2) &= 10 & \text{ord}(7) &= 10 \\ \text{ord}(3) &= 5 & \text{ord}(8) &= 10 \\ \text{ord}(4) &= 5 & \text{ord}(9) &= 5 \\ \text{ord}(5) &= 5 & \text{ord}(10) &= 2 \end{aligned}$$

- The number of primitive elements in G is $\Phi(|G|)$
- Example: for Z_{11}^* $\Phi(10)=(5-1)(2-1)=4$
- If |G| is prime all elements $a \neq 1$ are primitive

The Generalized Discrete Logarithm Problem

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

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

Two discrete logarithm problems are used in cryptography

1. The multiplicative group of the prime field Z_p *or a subgroup of it, e.g.,* the classical DHKE, ElGamal encryption and the Digital Signature Algorithm (DSA).
2. The cyclic group formed by an elliptic curve (see `later`)

Attacks against the Discrete Logarithm

- 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 $\beta = \alpha \circ \alpha \circ \ldots \circ \alpha = \alpha^x$
- Algorithms for computing discrete logarithms
  - Generic algorithms: Work for any cyclic group
    - Brute-Force Search ($\approx (p-1)/2$ computations)
    - Shanks' Baby-Step-Giant-Step Method
    - Pollard's Rho Method - Pohlig-Hellman Method
  - Non-generic Algorithms: Work only for specific groups, in particular in Z_p: The Index Calculus Method
- 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 DL – Shanks' method

- Write $x = \log_\alpha \beta \bmod p$ as $x = x_g\, m + x_b$ for $0 \le x_g, x_b < m$ where $m = \lceil \sqrt{|G|} \rceil$

$$\beta = \alpha^x = \alpha^{x_g m + x_b} \qquad \beta \cdot (\alpha^{-m})^{x_g} = \alpha^{x_b}$$

- Search for x_g and x_b separately:
  (1) Baby-step: compute and store all values of $\alpha^{\{x_b\}}$
  (2) Giant-step: check for all x_g if for some stored entry from (1)

$$\beta \cdot (\alpha^{-m})^{x_g} \stackrel{?}{=} \alpha^{x_b}$$

- For a group G or order 2^80 only 2^40 values are needed, i.e., p should have at least 160 bits

Attacks against DL – Polard's Rho method

- Probabilistic algorithm based on the birthday paradox
- Generate random elements of the form $\alpha^i \beta^j$, store the values of i and j
- Continue until we find a collision, i.e., $\alpha^{i_1} \cdot \beta^{j_1} = \alpha^{i_2} \cdot \beta^{j_2}$
- Substituting $\beta = \alpha^x$, yields $i_1 + x j_1 = i_2 + x j_2 \bmod |G|$
- Then $$x \equiv \frac{i_2 - i_1}{j_1 - j_2} \bmod |G|$$
- The attacks complexity is of the order of $\sqrt{|G|}$
- This is also the best known algoritm to calculate the discrete logarithm in elliptic curve groups. Therefore, elliptic curve groups should have at least a size of 2^{160}

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

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^* 2022: *1024-2048* bits

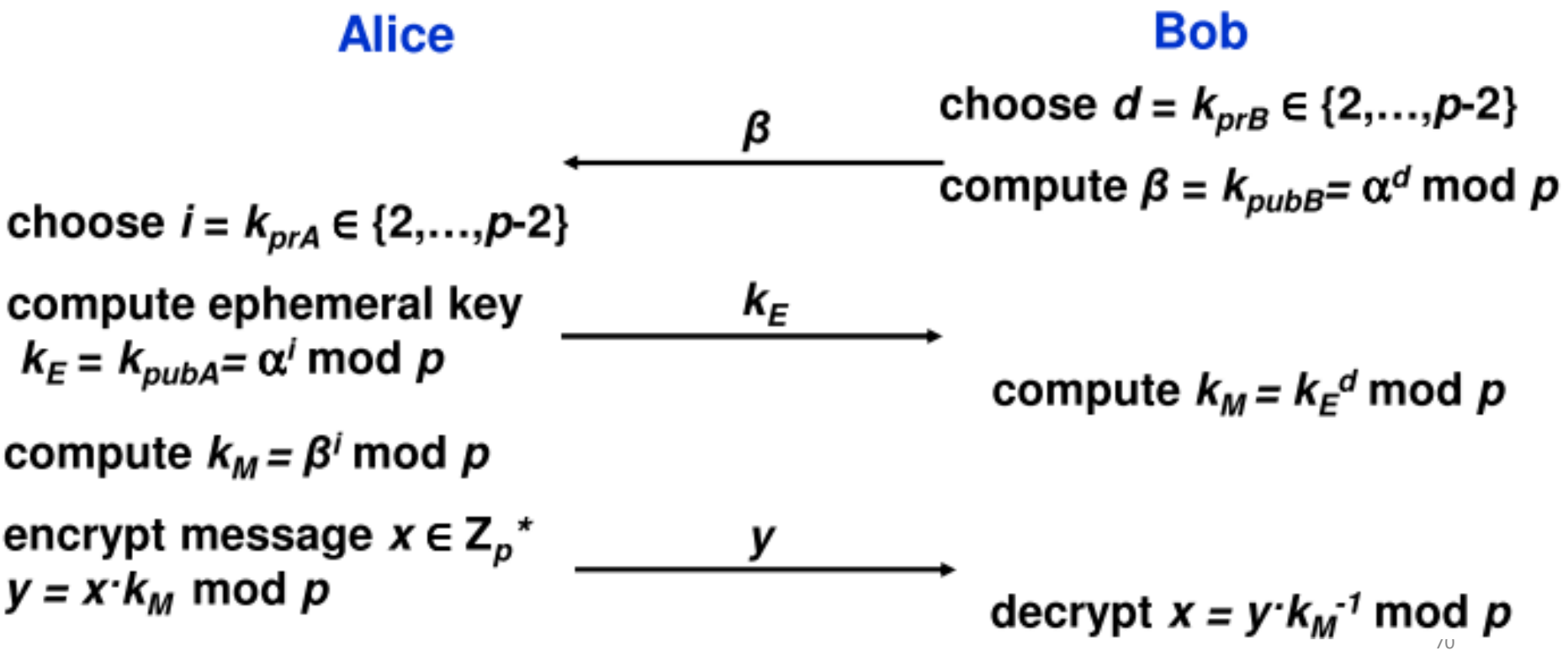
Security of the classical DH Key Exchange

- 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 known 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 $k_{AB} = B^a = \alpha^{ba} \bmod p$

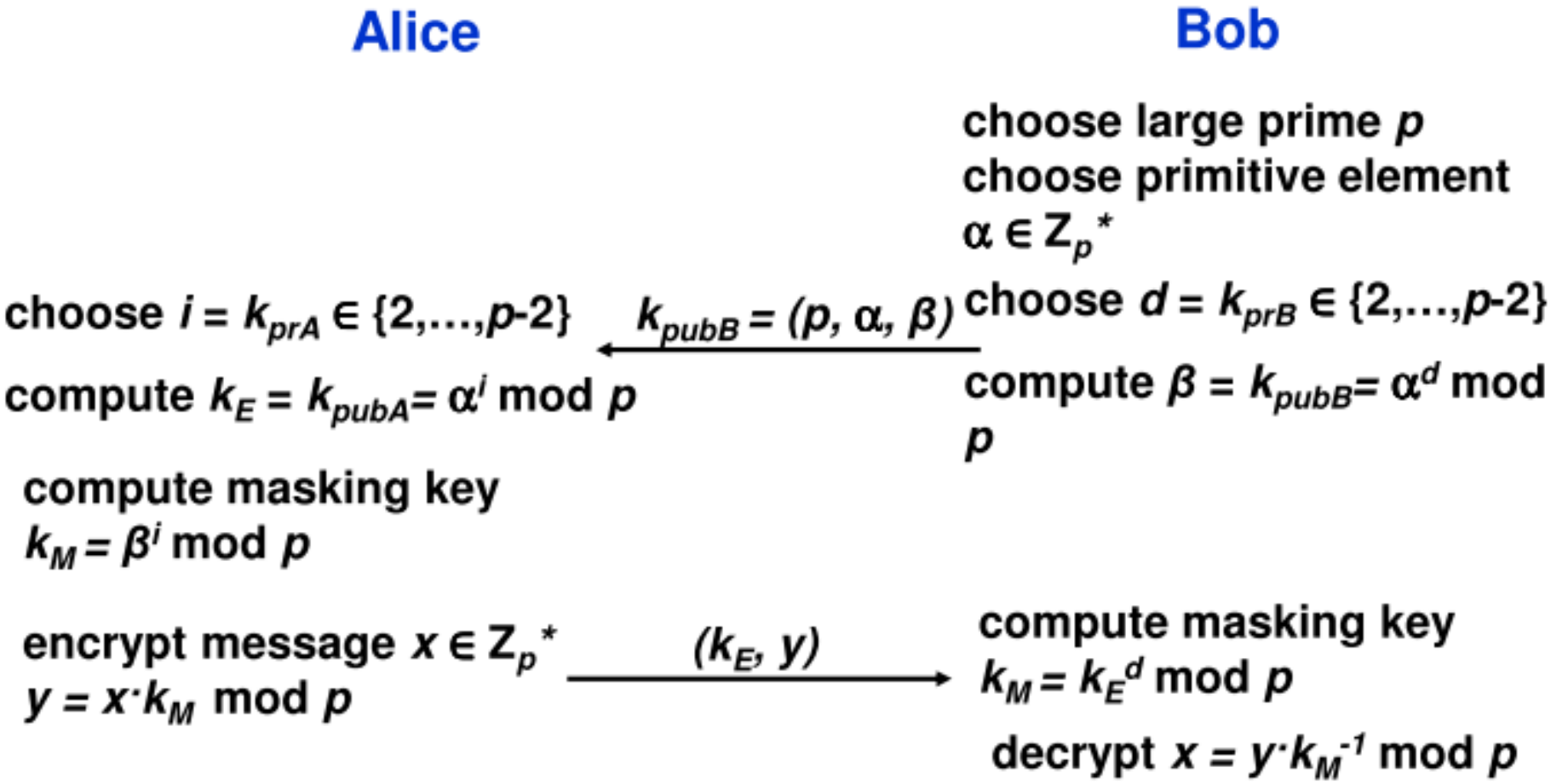
  It is conjectured that the DHP and the DLP are equivalent, i.e., solving DHP implies solving DLP
- To prevent attacks, i.e., to prevent that the DLP can be solved, choose $p \geq 2^{1024}$

The ElGamal Encryption Scheme: Principle

- Proposed by ElGamal in 1985 - can be viewed as an extension of the DHKE protocol
- Based on the intractability of the discrete logarithm and the Diffie–Hellman problems
- Select p and α



The ElGamal Encryption Protocol



The ElGamal protocol re-orders the computations to save one communication

The ElGamal Protocol - Example

Alice

Message $x=12$

choose $i = k_{prA} = 2$

$k_E = k_{pubA} = 3^2 = 9 \bmod 31$

masking key $k_M = 26^2 = 25 \bmod 31$

encrypt $x=12$:

$y = 12 \cdot 25 \bmod 31 = 21 \bmod 31$

$\gcd(31,25)=\gcd(25,6)=\gcd(6,1)=1$

$6=31 \cdot 1-25 \cdot 1$; $\{25 = 6 \cdot 4+1 = 31 \cdot 4 - 25 \cdot 4+1\}$

$\Rightarrow 1=25 \cdot 5 - 31 \cdot 4 \Rightarrow 25^{-1} = 5 \bmod 31$

Bob

$p=31$; *primitive* element $\alpha=3$

choose $d = k_{prB} = 5$

$\beta = k_{pubB} = \alpha^d = 3^5 = 26 \bmod 31$

$k_{pubB} = (31,3,26)$ ←

(9,21) →

masking key

$k_M = 9^5 = 25 \bmod 31$

decrypt $x = y \cdot k_M^{-1} \bmod p$

$= 21 \cdot 25^{-1} \bmod 31$

$= 21 \cdot 5 = 12 \bmod 31$

Computational Aspects

- Key Generation
  - Generation of prime p: has to have at least 1024 bits
- Encryption
  - Requires two modular exponentiations and a modular multiplication
  - All operands have a bit length of $\log_2 p$
  - Efficient execution requires methods such as the square-and-multiply algorithm
- Decryption
  - Requires one modular exponentiation and one inversion
  - Inversion can be computed from the ephemeral key:

$$k_M^{-1} \bmod p = (k_E^d)^{-1} \bmod p = (k_E^d)^{-1} k_E^{p-1} \bmod p = k_E^{p-d-1} \bmod p$$

(*Since* $k_E^{p-1} = 1 \bmod p$ according to Fermat's little theorem)

Example: for $k_E = 9$, $k_E^{p-d-1} \bmod p = 9^{31-5-1} = 9^{25} \bmod 31 = 5 \bmod 31$

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

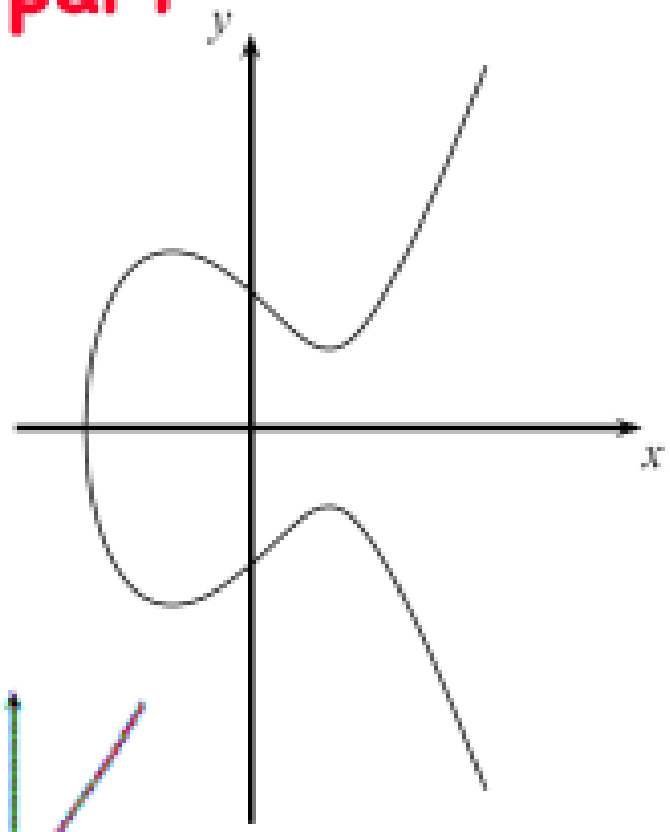
- Active attacks
  - If the public keys are not authenticated, an attacker could send an incorrect public key (cf. Chapter 13)
  - An Attack is also possible if the secret exponent i is being used more than once (if you know/guess 1st message you can decrypt the 2nd)
  - ElGammal protocol is malleable: Oscar can replace (k_E, y) by $(k_E, s \cdot y)$ and the receiver would get $s \cdot x$

Lessons Learned

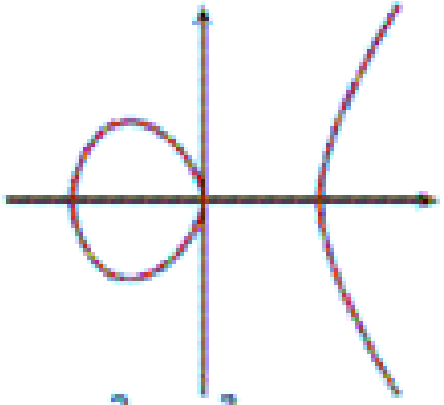
- The Diffie–Hellman protocol is widely used for key exchange.
- The discrete logarithm problem is one of the most important one-way functions in modern asymmetric cryptography. Many public-key algorithms are based on it.
- For the Diffie–Hellman protocol in Z_p^*, *the prime p should be at least 1024 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 mask to encrypt a message.
- ElGamal is a probabilistic encryption scheme, i.e., encrypting two identical messages does not yield identical ciphertexts.

Content of this part

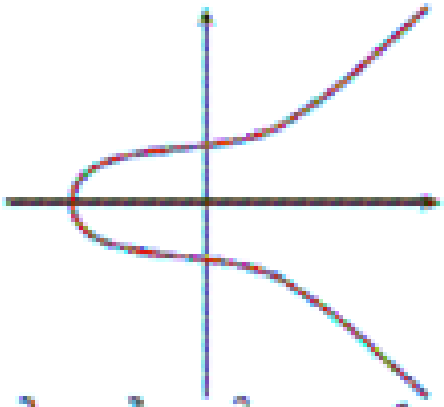
- Introduction
- Computations on Elliptic Curves
- The Elliptic Curve Diffie-Hellman Protocol
- Security Aspects
- Implementation in Software and Hardware



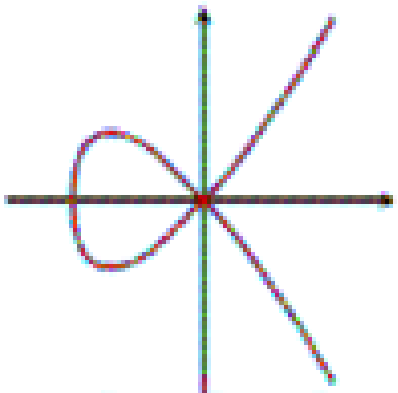
$y^2 = x^3 - 3x + 3$



$y^2 = x^3 - x$



$y^2 = x^3 + x^2 + x + 1$



$y^2 = x^3 + x^2$
singularity at (0, 0)

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 size is difficult (storage space) for small embedded devices
- **Goal:**
  Smaller field sizes that can provide equivalent security
- **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

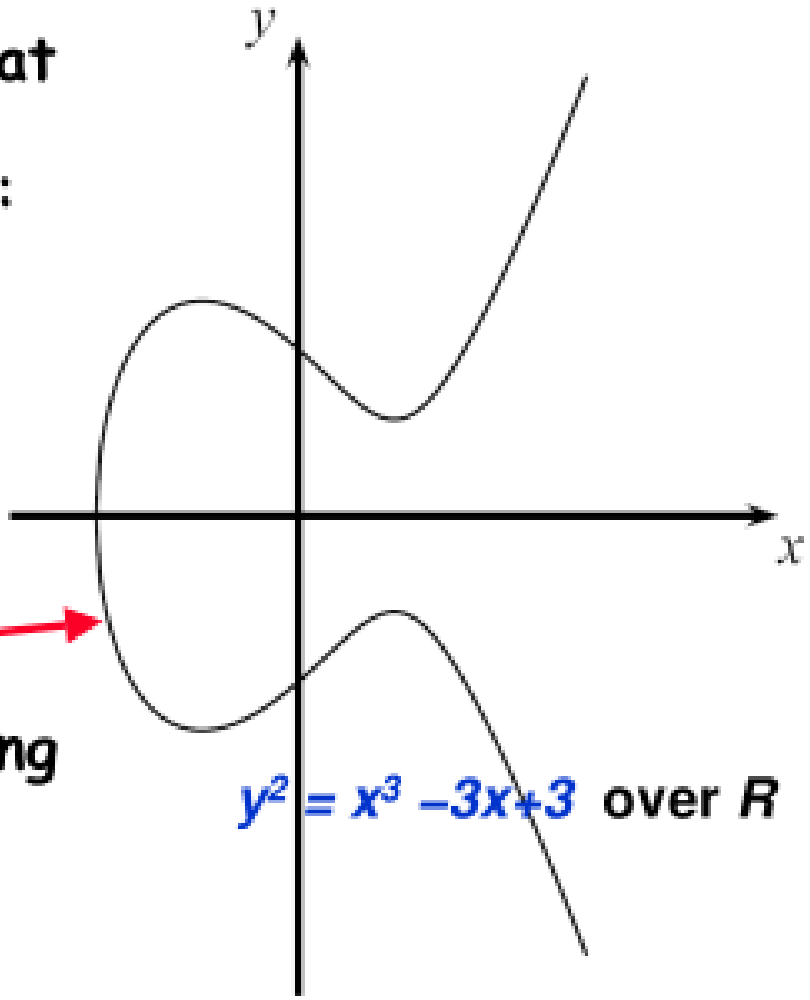
Elliptic Curves over R

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



Elliptic Curves over the field Z_p

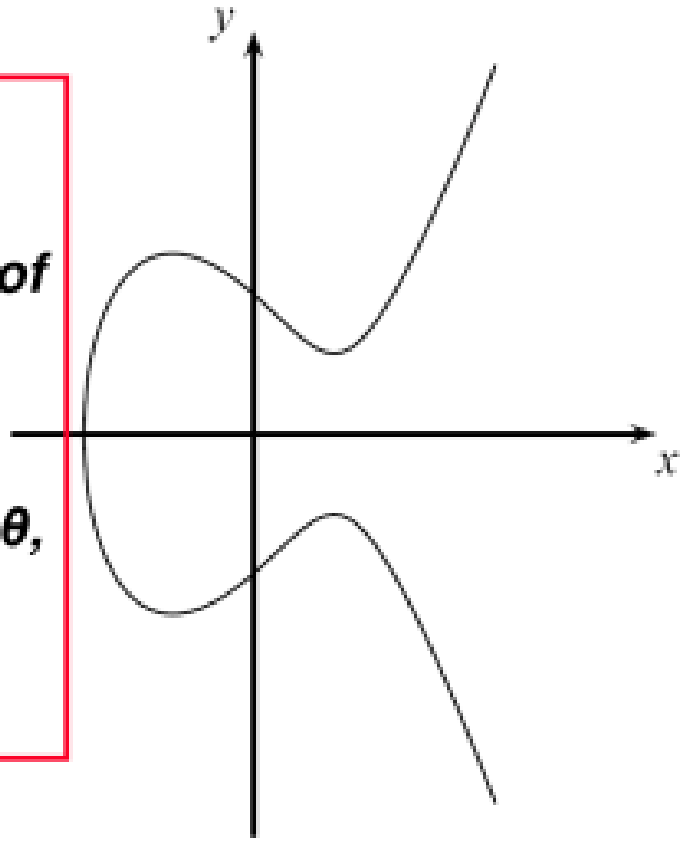
- In cryptography, we are interested in elliptic curves module a prime *p*:

Definition: ***Elliptic Curves over prime fields***

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

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

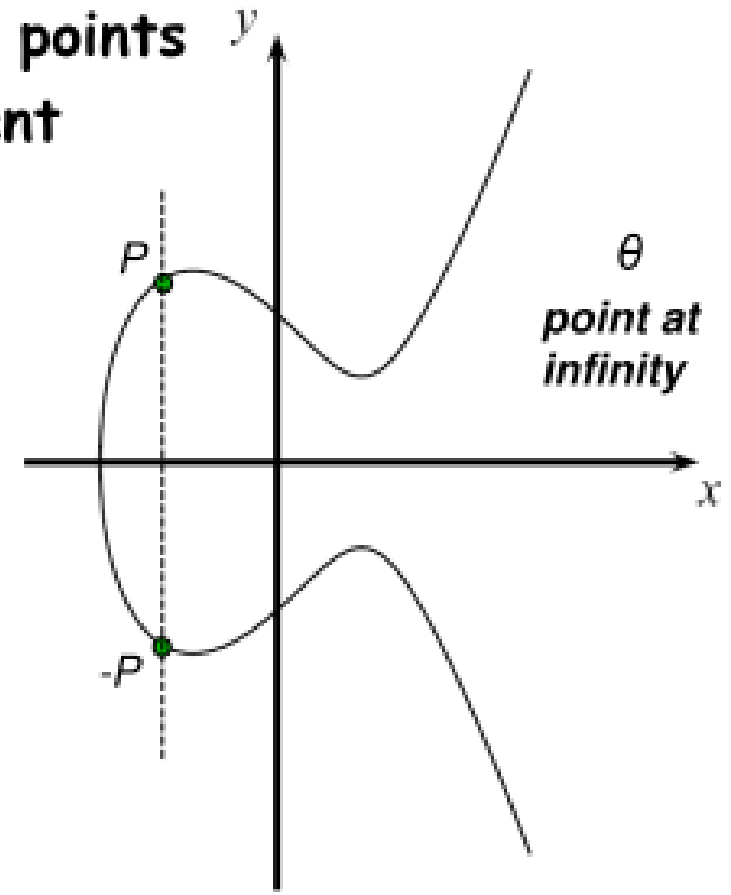
together with an imaginary point at infinity θ, where $a,b \in Z_p$ and $4a^3+27b^2 \neq 0 \bmod p$ (to avoid self-intersections of the curve)



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

Computations on Elliptic Curves

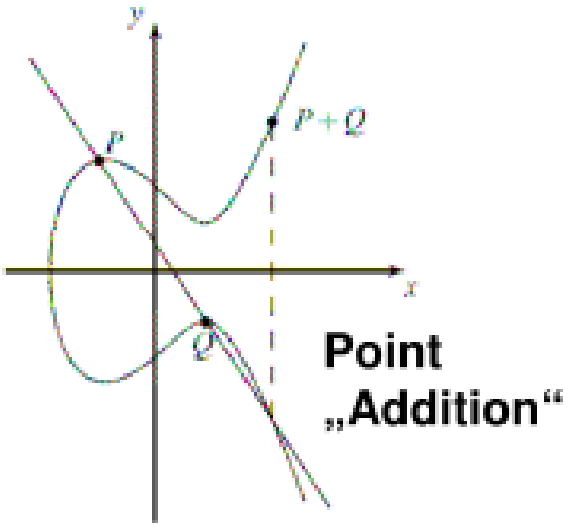
- Special considerations are required to convert elliptic curves into a group of points
- In any group, a special identity element is required, i.e., given $P \in E$: $P + \theta = P = \theta + P$
- This identity point, denoted by θ is not on the curve and is added to the group definition
- Up to two *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 $-P = (x, -y)$
- $P+(-P)= \theta$



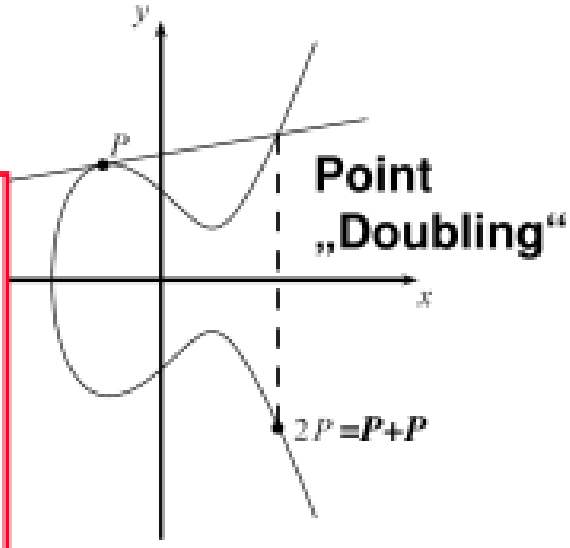
$y^2 = x^3 + ax + b \bmod p$

Computations on Elliptic Curves (ctd.)

- Generating a *group of points* on elliptic curves based on „point addition" operation $P+Q = W$, *i.e.*, $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$
- Geometric Interpretation of point addition operation
  - *Draw straight line through* $P=(x_1, y_1)$ *and* $Q=(x_2, y_2)$; *if* $P=Q$ *use tangent line instead*
  - *Mirror 3rd intersection point of drawn line with the elliptic curve with respect to 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

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

Abelian (commutative) group

- $P + \theta = P$
- If $P = (x_1, y_1)$, $-P = (x_1, -y_1)$
- $P + (-P) = \theta$ (not on the curve)
- $P+Q = Q+P$
- $P+(Q+W)=(P+Q)+W$

Computations on Elliptic Curves – Example 1

- Example: Given $E: y^2 = x^3+2x+2 \bmod 17$ and point $P=(5,1)$
  Goal: Compute $2P = P+P = (5,1)+(5,1)= (x_3, y_3)$

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

$$x_3 = s^2 - x_1 - x_2 = 13^2 - 5 - 5 = 159 \equiv 6 \bmod 17$$
$$y_3 = s(x_1-x_3) - y_1 = 13(5 - 6) - 1 = -14 \equiv 3 \bmod 17$$

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

Verify that $(6,3)$ is a point on the curve:

$3^2 = 9$, $6^3 + 12 + 2 = 36\cdot 6 + 14 \equiv 2\cdot 6 + 14 = 26 \equiv 9 \bmod 17$

$P+2P=(5,1)+(6,3)=(10,6)$ since

$$s = \frac{y_2 - y_1}{x_2 - x_1} \bmod p = (3-1)/(6-5) = 2\ (1)^{-1} \equiv 2 \bmod 17$$

$x_3 = s^2 - x_1 - x_2 = 4 - 11 = -7 \equiv 10 \bmod 17$;

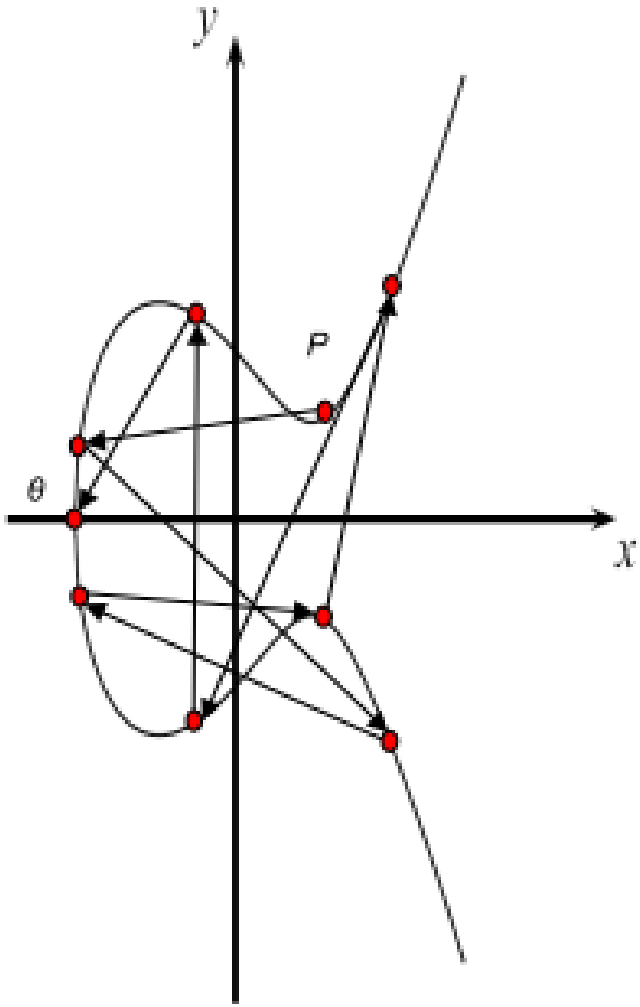
$y_3 = s(x_1 - x_3) - y_1 = -10 - 1 = -11 \equiv 6 \bmod 17$

Example 1 (ctd.)

- The points on an elliptic curve and the point 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.



Example 2

- Given $E: y^2 = x^3+x+3 \bmod 7$
- There are 6 points on this curve: Θ, (4,1), (6,6), (5,0), (6,1), (4,6)
- Addition table:

+	$\mathcal{O}$	(4, 1)	(6, 6)	(5, 0)	(6, 1)	(4, 6)
$\mathcal{O}$	$\mathcal{O}$	(4, 1)	(6, 6)	(5, 0)	(6, 1)	(4, 6)
(4, 1)	(4, 1)	(6, 6)	(5, 0)	(6, 1)	(4, 6)	$\mathcal{O}$
(6, 6)	(6, 6)	(5, 0)	(6, 1)	(4, 6)	$\mathcal{O}$	(4, 1)
(5, 0)	(5, 0)	(6, 1)	(4, 6)	$\mathcal{O}$	(4, 1)	(6, 6)
(6, 1)	(6, 1)	(4, 6)	$\mathcal{O}$	(4, 1)	(6, 6)	(5, 0)
(4, 6)	(4, 6)	$\mathcal{O}$	(4, 1)	(6, 6)	(5, 0)	(6, 1)

- P=(4,1): 2P=(6,6),3P=(5,0),4P=(6,1),5P=(4,6);6P=Θ
- P=(4,6): 2P=(6,1),3P=(5,0),4P=(6,6),5P=(4,1);6P=Θ
- P=(5,0): 2P= Θ; P=(6,1): 2P=(6,6),3P=Θ

Number of Points on an Elliptic Curve

- How many points can be on an arbitrary elliptic curve?
  - Example 1, $E: y^2 = x^3+2x+2 \bmod 17$ 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

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 $1 \le d \le \#E$ such that

$$\underbrace{P + P + \ldots + P}_{d \text{ times}} = dxP = T$$

- 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 *dxP* 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: $T = d\,P$

Initialization:

$T = P$

Algorithm:

FOR $i = t-1$ DOWNTO 0

  $T = T + T \bmod n$

  IF $d_i = 1$

    $T = T + P \bmod n$

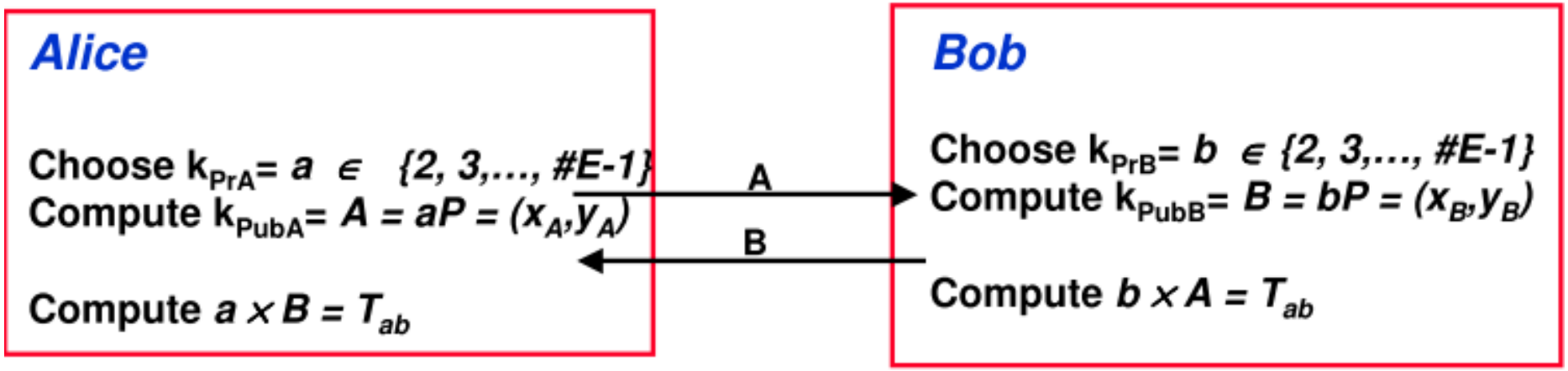
RETURN (T)

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

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

Elliptic Curve Diffie-Hellman Key Exchange (ECDH)

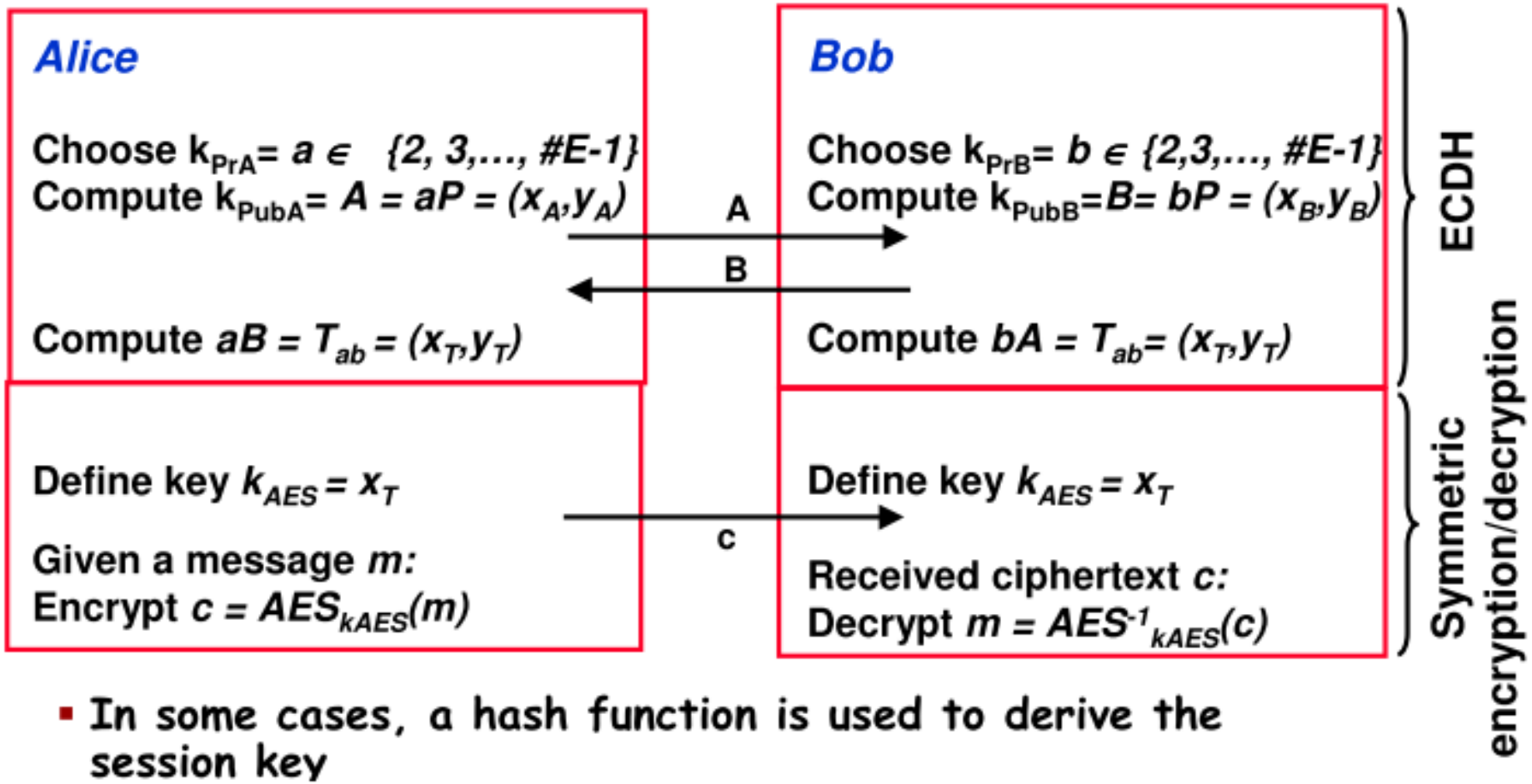
- Given a prime p, an elliptic curve E and a primitive point $P=(x_P, y_P)$
- The Elliptic Curve Diffie-Hellman Key Exchange protocol:



- 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

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



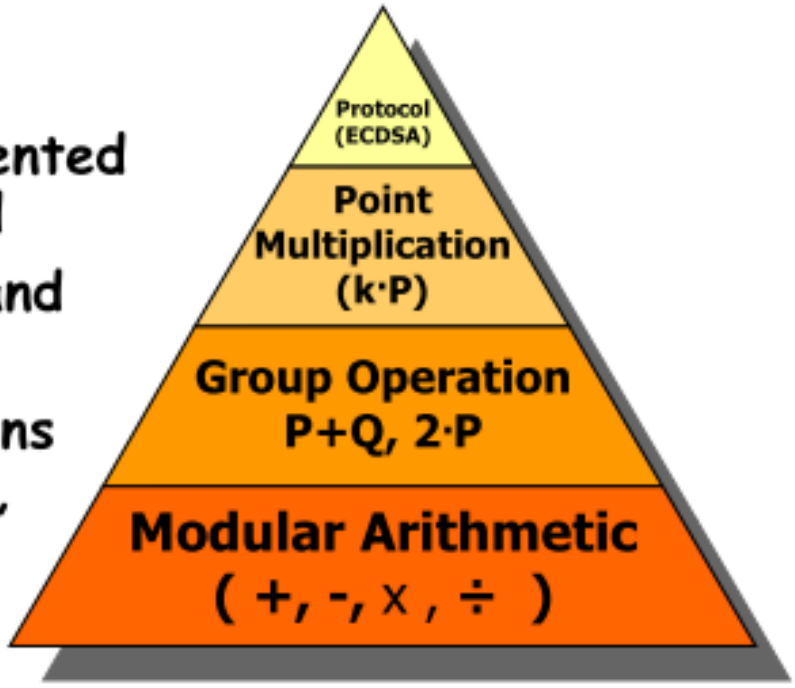
- In some cases, a hash function is used to derive the session key

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

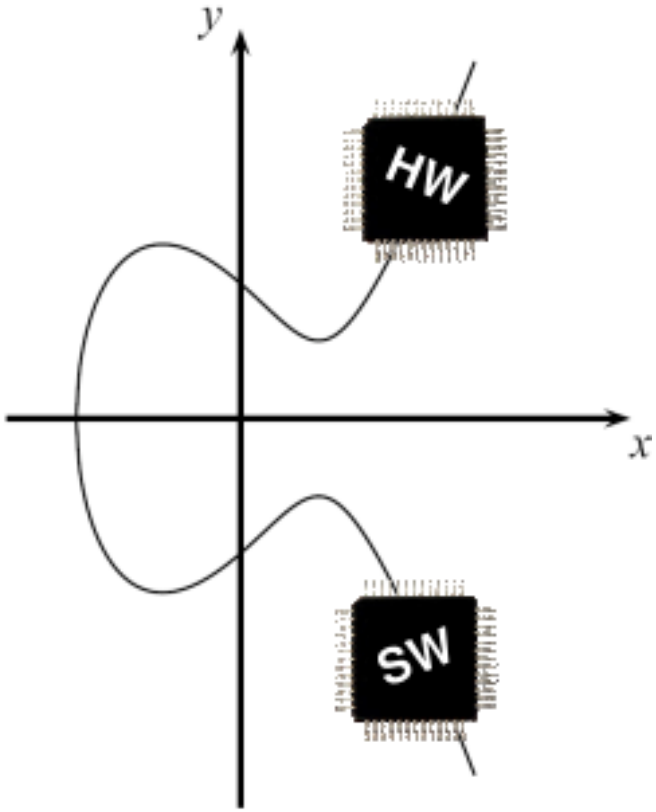
Implementations in Hardware and Software

- Elliptic curve computations usually regarded as consisting of four layers:
  - Basic modular arithmetic operations
  - 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 hundreds microseconds on reconfigurable hardware
  - Dedicated chips for ECC can compute a point multiplication even in a few tens microseconds



Lessons Learned

- Elliptic Curve Cryptography (ECC) is based on the discrete logarithm problem
- 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), resulting 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

Thank You!


DECRYPTION
ASYMMETRIC TYPE
PUBLIC
ALGORITHM
KEY
SCIENCE
DATA CENTER
SIGNATURE
SECRET
DATA
CRYPTOGRAPHY
INTEGRITY
MESSAGE
CRYPTOSYSTEM
CIPHER
SECURITY
SENDER
EXCHANGE
CODE
STREAM
NETWORK
ENCRYPTION
CIPHERTEXT
COMPUTATION
CRYPTOANALYSIS

Questions???