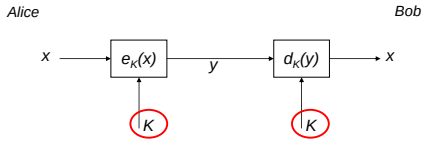
Public Key

Instructor: Dr. Wei (Lisa) Li
Department of Computer Science, GSU

Symmetric Cryptography Review



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 (1)

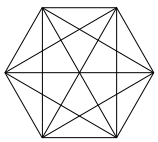
- Symmetric algorithms, e.g., AES or 3DES, are very secure, fast & widespread **but**:
- **Key distribution problem:** The secret key must be transported securely
- **Number of keys:** In a network, each pair of users requires an individual key

→ n users in the network require n(n-1)/2 keys, each user stores (n-1) keys

Example:

6 users (nodes)

6*5/2 = 15 keys (edges)



Symmetric Cryptography: Shortcomings (2)

- Alice or Bob can **cheat each other**, because they have/share the same 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**

Asymmetric Cryptography: Idea

New Idea:

Use the *good old mailbox principle:*

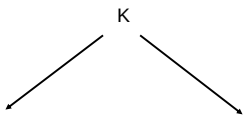
Everyone can drop a letter

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

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

Asymmetric (Public-Key) Cryptography

Principle: "Split up" the key



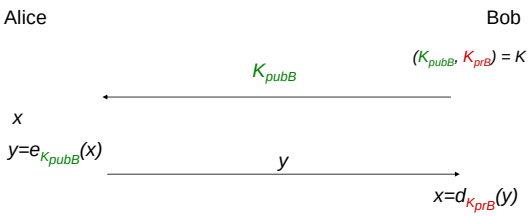
→ During the key generation, a key pair K_{pub} and K_{pr} is computed

Asymmetric Cryptography: Analogy

Safe with public lock and private lock:

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

Basic Protocol for Public-Key Encryption



9

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 and DSA 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!)

10

Basic Key Transport Protocol

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

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

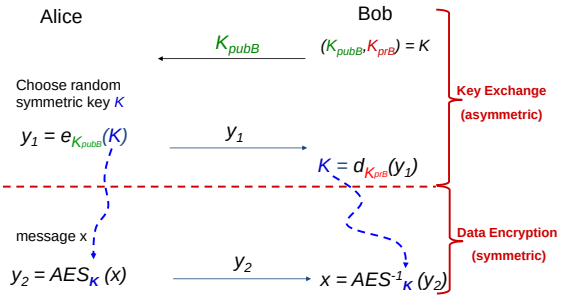
2. **Encryption** of data is done using (fast) symmetric ciphers, e.g., **block ciphers or stream ciphers**

11

How to perform symmetric encryption, e.g., AES, in a public-key encryption protocol?

12

Example: Hybrid protocol with AES as the symmetric cipher



How to build Public-Key Algorithms (1)

Definition 6.1.1 One-way function
A function $f()$ is a one-way function if:

1. $y = f(x)$ *is computationally easy, and*
2. $x = f^{-1}(y)$ *is computationally infeasible.*

- $y = f(x)$ should be sufficiently fast for practical applications
- $x = f^{-1}(y)$ should be computationally intensive for security

How to build Public-Key Algorithms (2)

Asymmetric schemes are based on a ``**one-way function**'' ***f()***:

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

One way functions are based on **mathematically hard problems**. Three main families:

- **Factoring integers** (RSA, ...):
  Given a composite integer n, find its prime factors
  (Multiply two primes: easy)
- **Discrete Logarithm** (Diffie-Hellman, Elgamal, DSA, …):
  Given a, y and m, find x such that $a^x = y \bmod m$
  (Exponentiation a^x: easy)
- **Elliptic Curves (EC)** (ECDH, ECDSA): Generalization of discrete logarithm

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

Key Lengths and Security Levels

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

- The exact complexity of RSA (factoring) and DL (Index-Calculus) is difficult to estimate
- The existence of quantum computers would probably be the end for ECC, RSA & DL (at least 2-3 decades away, some people doubt that QC will ever exist, but now, Intel has created 49- and 17-qubit superconducting test chips for quantum computing)

Further Discussion

- **Authenticity of Public Keys:** how do we really know that a certain public key belongs to a certain person?
- **Implementation of Public-Key Encryption:** how to efficiently implement a public-key encryption?

Essential Number Theory for Public-Key Algorithms

Euclidean Algorithm (1)

- Compute the **greatest common divisor *gcd* (r_0, r_1)** of two integers r_0 and r_1
- gcd is **easy for small numbers**:
  1. factor r_0 and r_1
  2. gcd = highest common factor
- Example:

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

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

  $2 \cdot 3 = 6 = gcd\,(30,84)$
- **But:** Factoring is complicated (and often infeasible) for large numbers

Euclidean Algorithm (2)

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 !

Eudlidean Algorithm (3)

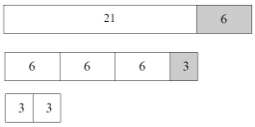
Euclidean Algorithm
Input: positive integers r_0 and r_1 with $r_0 > r_1$
Output: $\gcd(r_0, r_1)$
Initialization: $i = 1$
Algorithm:

1 DO
1.1 $i = i+1$
1.2 $r_i = r_{i-2} \bmod r_{i-1}$
WHILE $r_i \neq 0$
2 RETURN
$\gcd(r_0, r_1) = r_{i-1}$

Remark of WHILE loop:
gcd (r_0, r_1) = gcd (r_0 - r_1, r_1) = gcd (r_0 - 2r_1, r_1)
= ...= gcd (r_0 -m r_1, r_1)
→ gcd (r_0, r_1) = gcd (r_0 mod r_1, r_1)
→ gcd (r_0, r_1) = gcd (r_{i-1}, 0) = r_{i-1}

Example 1: Eudlidean Algorithm

Example 1: *gcd (r_0, r_1)* for r_0 = 27 and r_1 = 21



gcd(27, 21) = gcd(1 · 21+6, 21) = gcd(21, 6)
gcd(21, 6) = gcd(3 · 6+3, 6) = gcd(6, 3)
gcd(6, 3) = gcd(2 · 3+0, 3) = gcd(3, 0) = 3

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

Example 2: Eudlidean Algorithm

Example 2: *gcd (r_0, r_1)* for r_0 = 973 and r_1 = 301. The gcd can be computed as follows.

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

Note: Reduce the problem of finding the gcd of two larger numbers to the of the gcd of two smaller numbers.

Extended Euclidean Algorithm (1)

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

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

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

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

Extended Euclidean Algorithm (2)

Extended Euclidean Algorithm (EEA)
Input: positive integers r_0 and r_1 with $r_0 > r_1$
Output: $\gcd(r_0, r_1)$, as well as s and t such that $\gcd(r_0, r_1) = s \cdot r_0 + t \cdot r_1$.
Initialization:
$s_0 = 1 \quad t_0 = 0$
$s_1 = 0 \quad t_1 = 1$
$i = 1$
Algorithm:

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 $r_i \neq 0$
2 RETURN
$\gcd(r_0, r_1) = r_{i-1}$
$s = s_{i-1}$
$t = t_{i-1}$

Remark of WHILE loop:

gcd (r_0, r_1) = gcd (r_0 mod r_1, r_1)

→ $r_2 = r_0 \bmod r_1$, $r_0 = q_1 r_1 + r_2$

→ $r_{i-2} = q_{i-1} r_{i-1} + r_i$

→ $r_i = r_{i-2} - q_{i-1} r_{i-1} = [s_i] r_0 + [t_i] r_1$

Example: EEA

- Calculate the modular Inverse of 12 mod 67:
- From magic table follows
- Hence **28 is the inverse** of 12 mod 67.

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

- Check: 28 · 12 = 336 ≡ 1 mod 67 ✓

Euler's Phi Function (1)

- *New problem, important for public-key systems, e.g., RSA:*
  Given the set of the *m* integers {0, 1, 2, …, *m* -1},
  How many numbers in the set are **relatively prime to *m*** ?
- Answer: **Euler's Phi function *Φ(m)***
- **Example** for the sets {0,1,2,3,4,5} (*m*=6) and {0,1,2,3,4} (*m*=5)

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

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

→ 1 and 5 relatively prime to *m*=6, → ***Φ*(5) = 4**
hence ***Φ*(6) = 2**

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

Euler's Phi Function (2)

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

- Phi especially easy for $e_i = 1$, e.g., $m = p \cdot q$ → $\Phi(m) = (p-1) \cdot (q-1)$
- **Example** m = 899 = 29 · 31:
  ***Φ*(899)** = (29-1) · (31-1) = 28 · 30 **= 840**
- **Note:** Finding *Φ(m)* is computationally easy **if factorization of *m* is known** (otherwise the calculation of *Φ(m)* becomes computationally infeasible for large numbers)

Fermat's Little Theorem

- Given a **prime *p*** and an **integer *a***: $a^p \equiv a \pmod{p}$
- Can be rewritten as: $a^{p-1} \equiv 1 \pmod{p}$
- **Use: Find modular inverse**, if p is prime. Rewrite to $a \cdot a^{p-2} \equiv 1 \pmod{p}$
- Comparing with definition of the modular inverse: $a \cdot a^{-1} \equiv 1 \bmod m$
  → $a^{-1} \equiv a^{p-2} \pmod{p}$ is the modular inverse modulo a prime *p*

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

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

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

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

Euler's Theorem

- Generalization of Fermat's little theorem to **any integer modulus**
- Given two **relatively prime integers *a*** and ***m*** **:** $a^{\Phi(m)} \equiv 1 \pmod{m}$
- **Example**: *m*=12, *a*=5
  1. Calculate Euler's Phi Function
     $\Phi(12) = \Phi(2^2 \cdot 3) = (2^2 - 2^1)(3^1 - 3^0) = (4-2)(3-1) = 4$
  2. Verify Euler's Theorem
     $5^{\Phi(12)} = 5^4 = 25^2 = 625 \equiv 1 \bmod 12$
- Fermat's little theorem = special case of Euler's Theorem
- For a prime ***p***: $\Phi(p) = (p^1 - p^0) = p - 1$
  → Fermat: $a^{\Phi(p)} = a^{p-1} \equiv 1 \pmod{p}$

RSA Cryptosystem

- Martin Hellman and Whitfield Diffie published their landmark public-key paper in 1976
- Ronald Rivest, Adi Shamir and Leonard Adleman proposed the asymmetric RSA cryptosystem in1977
- Until now, RSA is the most widely use asymmetric cryptosystem although elliptic curve cryptography (ECC) becomes increasingly popular
- RSA is mainly used for two applications
  - Transport of (i.e., symmetric) keys
  - 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* being large primes
- Encryption and decryption are simply exponentiations in the ring

Definition

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

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

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

where x, y in Z_n.

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

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

Key Generation (1)

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

Algorithm: RSA Key Generation

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

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

Key Generation (2)

Algorithm: RSA Key Generation

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

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

Remarks:

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

Example: RSA with small numbers

ALICE		BOB
Message $x = 4$		1. Choose $p = 3$ and $q = 11$
		2. Compute $n = p * q = 33$
		3. $\Phi(n) = (3-1) * (11-1) = 20$
		4. Choose $e = 3$
		5. $d \equiv e^{-1} \equiv 7 \bmod 20$
	$K_{pub} = (33,3)$ ←	
$y = x^e \equiv 4^3 \equiv 31 \bmod 33$		
	y = 31 →	
		$y^d = 31^7 \equiv 4 = x \bmod 33$

RSA Implementation

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

Attacks and Countermeasures

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

37

Discrete Logarithm

Discrete Logarithm Problem

39

Preliminary (1)

Definition 8.2.1 Group

A group *is a set of elements G together with an operation ∘ which combines two elements of G. A group has the following properties.*

1. *The group operation ∘ is* closed. *That is, for all* $a, b, \in G$, *it holds that* $a \circ b = c \in G$.
2. *The group operation is* associative. *That is,* $a \circ (b \circ c) = (a \circ b) \circ c$ *for all* $a, b, c \in G$.
3. *There is an element* $1 \in G$, *called the* neutral element *(or* identity element*), such that* $a \circ 1 = 1 \circ a = a$ *for all* $a \in G$.
4. *For each* $a \in G$ *there exists an element* $a^{-1} \in G$, *called the* inverse *of a, such that* $a \circ a^{-1} = a^{-1} \circ a = 1$.
5. *A group G is* abelian (or commutative) *if, furthermore,* $a \circ b = b \circ a$ *for all* $a, b \in G$.

E.g., Z_p^*: set of positive integers smaller than *p* which are relatively prime to *p*

40

Preliminary (2)

Definition 8.2.3 Order of an element
The order $ord(a)$ *of an element a of a group* $(G, \circ)$ *is the smallest positive integer k such that*

$$a^k = \underbrace{a \circ a \circ \ldots \circ a}_{k \text{ times}} = 1,$$

where 1 *is the identity element of G.*

Definition 8.2.4 Cyclic Group
A group G which contains an element α *with maximum order* $ord(\alpha) = |G|$ *is said to be* cyclic. *Elements with maximum order are called* primitive elements *or* generators.

Cyclic Group: Example

Example 8.6. We want to check whether $a = 2$ happens to be a primitive element of $\mathbb{Z}_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Note that the cardinality of the group is $|\mathbb{Z}_{11}^*| = 10$. Let's look at all the elements that are generated by powers of the element $a = 2$:

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

From the last result it follows that

$$\text{ord}(a) = 10 = |\mathbb{Z}_{11}^*|.$$

Discrete Logarithm Problem

Discrete Logarithm Problem (DLP) in Z_p^*

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

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

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

Generalized Discrete Logarithm Problem

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

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

Diffie–Hellman Key Exchange: Overview

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

Diffie–Hellman Key Exchange: Set-up

Set-up Steps:

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: Process

Alice		Bob
Choose random private key $k_{prA}=a \in \{1,2,\ldots,p-1\}$		Choose random private key $k_{prB}=b \in \{1,2,\ldots,p-1\}$
Compute corresponding public key $k_{pubA}= A = \alpha^a \bmod p$	A →	
	← B	Compute correspondig public key $k_{pubB}= B = \alpha^b \bmod p$
Compute common secret $k_{AB} = B^a = (\alpha^a)^b \bmod p$		Compute common secret $k_{AB} = A^b = (\alpha^b)^a \bmod p$
We can now use the joint key k_{AB} for encryption, e.g., with AES		
$y = AES_{kAB}(x)$	y →	$x = AES^{-1}_{kAB}(y)$

Diffie–Hellman Key Exchange: Example

Domain parameters p=29, α=2

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

Proof of correctness:
Alice computes: $B^a = (\alpha^b)^a \bmod p$
Bob computes: $A^b = (\alpha^a)^b \bmod p$
i.e., Alice and Bob compute the same key k_{AB}!

Attacks against DLP

Summary of records for computing discrete logarithms in Z_p^*

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

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

Security of Classical Diffie–Hellman Key Exchange (1)

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

Security of Classical Diffie–Hellman Key Exchange (2)

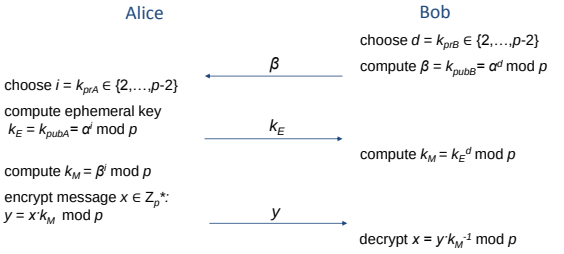
- The only known way to solve the DHP is to solve the DLP, i.e.
  1. Compute $a = log_\alpha A \bmod p$
  2. *Compute* $k_{AB} = B^a = \alpha^{ba} = \bmod p$

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

Elgamal Encryption Scheme: Overview

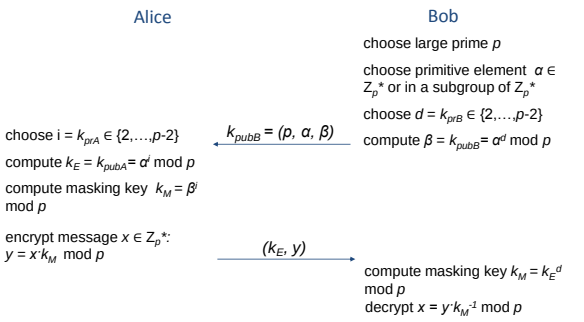
- Proposed by Taher Elgamal in 1985
- Can be viewed as an extension of the DHKE protocol
- Based on the intractability of the discrete logarithm problem and the Diffie–Hellman problem

Elgamal Encryption Scheme: Principle

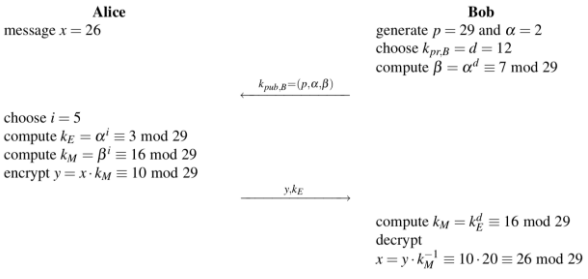


This looks very similar to the DHKE! The actual Elgamal protocol re-orders the computations which helps to save one communication (cf. next slide)

Elgamal Encryption Scheme: Protocol



Elgamal Encryption Scheme: Example



Computational Aspects

- Key Generation
  - Generation of prime p
  - p has to of size of at least 1024 bits
  - Prime-finding algorithms
- Encryption
  - Requires two modular exponentiations and a modular multiplictation
  - All operands have a bitlength of $\log_2 p$
  - Efficient execution requires methods such as the square-and-multiply algorithm
- Decryption
  - Requires one modular exponentiation and one modulare inversion

Security

- Passive attacks
  - Attacker eavesdrops p, α, $\beta = \alpha^d$, $k_E = \alpha^i$, $y = x \cdot \beta^i$ and wants to recover x
  - Problem relies on the DLP
- Active attacks
  - If the public keys are not authentic, an attacker could send an incorrect public key
  - An attack is also possible if the secret exponent i is being used more than once