50.042 FCS Summer 2024
Lecture 11 – Key Establishment

Felix LOH
Singapore University of Technology and Design

SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN

With selected materials adapted from: *Understanding Cryptography: A Textbook for Students and Practitioners, by C. Paar and J. Pelzl*

Preamble

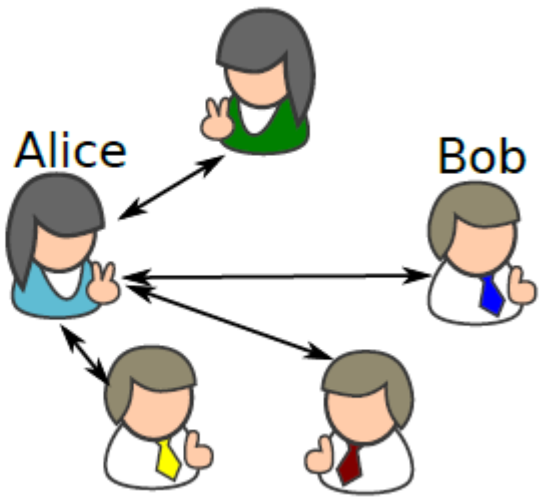
- So far, we have discussed a several ciphers in detail:
  - Shift ciphers
  - Substitution ciphers
  - DES, AES
  - Block ciphers
- These ciphers all assume that the keys have been securely transmitted between the parties involved (like Alice and Bob)
- In reality, ensuring that the keys are transmitted securely over an *insecure* channel is a non-trivial task
- In this lecture, we will discuss how the keys can be securely distributed

Key establishment

- The task of key establishment is one of the most important parts of a security system
- It deals with establishing a shared secret between two or more parties
- Key establishment is strongly related to the issue of identification
  - The attacker Oscar/Eve may attempt to join the key establishment protocol with the aim of impersonating either Alice or Bob
  - To prevent such attacks, each legitimate member must be assured of the identity of the other members

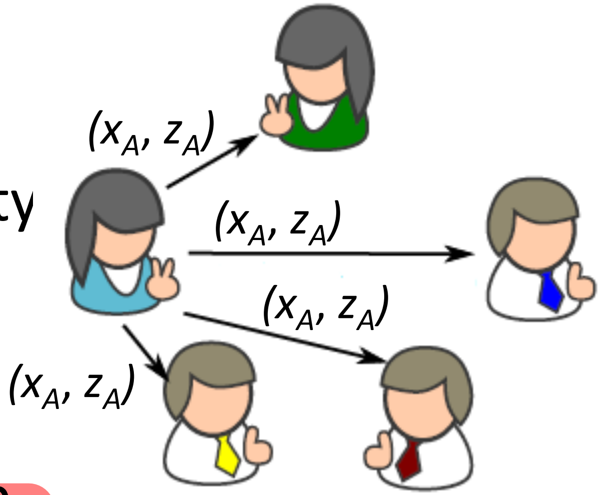
Key establishment: example and motivation

- Suppose we want to implement a secure chat application for students at SUTD (including students Alice and Bob)
  - Chat application needs to satisfy the confidentiality and integrity requirements
- Assume that it uses some wireless channel
- Users are identified by their student ID
  - Chat application uses the student IDs to determine the source and destination of a message
- How can we implement this chat application in a secure way?
  - What kind of infrastructure is required?
  - How many (and what kind of) messages need to be exchanged?
  - What happens when a new student enrols at SUTD?



Key establishment: example and motivation

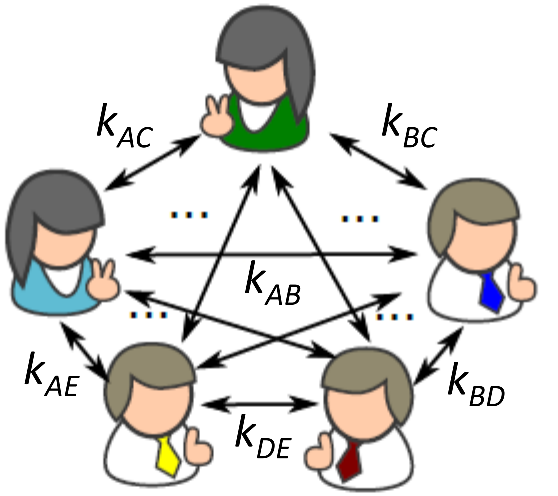
- A simple but insecure way to implement the application:
- Alice wants to send a plaintext message x_A to Bob…
  - She computes the message digest or hash of x_A, i.e. $z_A = h(x_A)$, then broadcasts the message and hash pair (x_A, z_A) over the wireless channel
  - Bob receives the message and hash pair (x_A, z_A), then he computes $z_B = h(x_A)$ to check that $z_A = z_B$ and verify the integrity of x_A
- The major problem with this method:
  - No confidentiality whatsoever – all other users on the chat application can read the message x_A that's only meant for Bob
- We need some way to distribute secret keys amongst the users so that we can encrypt messages



Key establishment: example and motivation

- A simple secure method – use pre-shared keys:
- One shared secret key for each pair of users
- Assume that the keys have been distributed via some secure channel
- Alice wants to send a plaintext message x_A to Bob…
  - She retrieves her shared secret key with Bob, k_{AB}
  - She computes the MAC, i.e. $m_A = MAC_{k_{AB}}(x_A)$ → MAC with secret key
  - She uses a cipher (say AES with key k_{AB}) to encrypt x_A, obtaining the ciphertext y_A, then broadcasts the ciphertext and MAC pair (y_A, m_A) over the wireless channel
  - Bob receives (y_A, m_A), decrypts y_A to obtain x_A, then he computes $m_B = MAC_{k_{AB}}(x_A)$ to check that $m_A = m_B$ and verify the integrity of x_A

MESSAGE AUTHENTICATION CODE

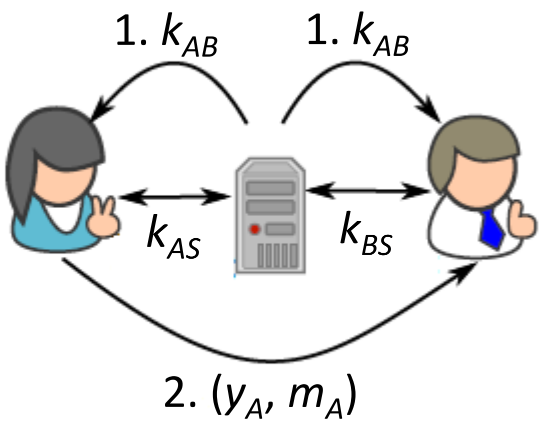


Pre-shared keys: comments

- This method can be a logistical nightmare, particularly for a large number of users
- Each user needs to store a key for every other user (recall that we need one shared key for each pair of users)
- The total number of keys needed for *n* users of the application is $n \cdot (n+1) / 2$
  - For example, with 1000 users, we have 500500 keys in total
- When a new user joins, every other existing user needs to obtain and store an additional new key

Key establishment: example and motivation

- A better secure method – use a centralized server for session key distribution (i.e. a KDC – key distribution center):
- Each user has a pre-shared secret key (e.g. k_{AS}) with the server ('S'), distributed via some secure channel
  - These keys are long-term keys; different from the session keys
- Alice wants to send a plaintext message x_A to Bob…
  - Using her pre-shared key k_{AS}, Alice securely requests the server for a new session key k_{AB} (this is also a secret key)
  - The server then distributes the encrypted session key k_{AB} to both Alice and Bob (they decrypt the session key by using their respective pre-shared secret keys)
  - Alice uses the session key k_{AB} to compute the MAC for x_A and to encrypt x_A with AES, then broadcasts the ciphertext and MAC pair (y_A, m_A) over the wireless channel
  - Bob decrypts the ciphertext y_A and verifies its integrity



Key distribution center (KDC): comments

- The total number of long-term keys needed for n users is n (much better and more feasible than the other method)
- The KDC can create a short-term shared session key and send this key to the appropriate pair of users
- The *Kerberos* protocol is one real-life example of such a scheme

- Some shortcomings:
  - There is a single point of failure (the server)
  - If the server is compromised, the long-term secret keys can be revealed
  - There is some communication overhead
  - Still needs a secure channel to distribute the pre-shared keys (but this is generally a one-time procedure)

Key establishment over an insecure channel

- From our brief overview of the above two secure methods, we can see that it is possible to provide both confidentiality and integrity using the concepts we learned so far
- Both methods require a secure channel to distribute the secret keys
- It would be nice if we can somehow distribute the secret keys over an insecure medium or channel…

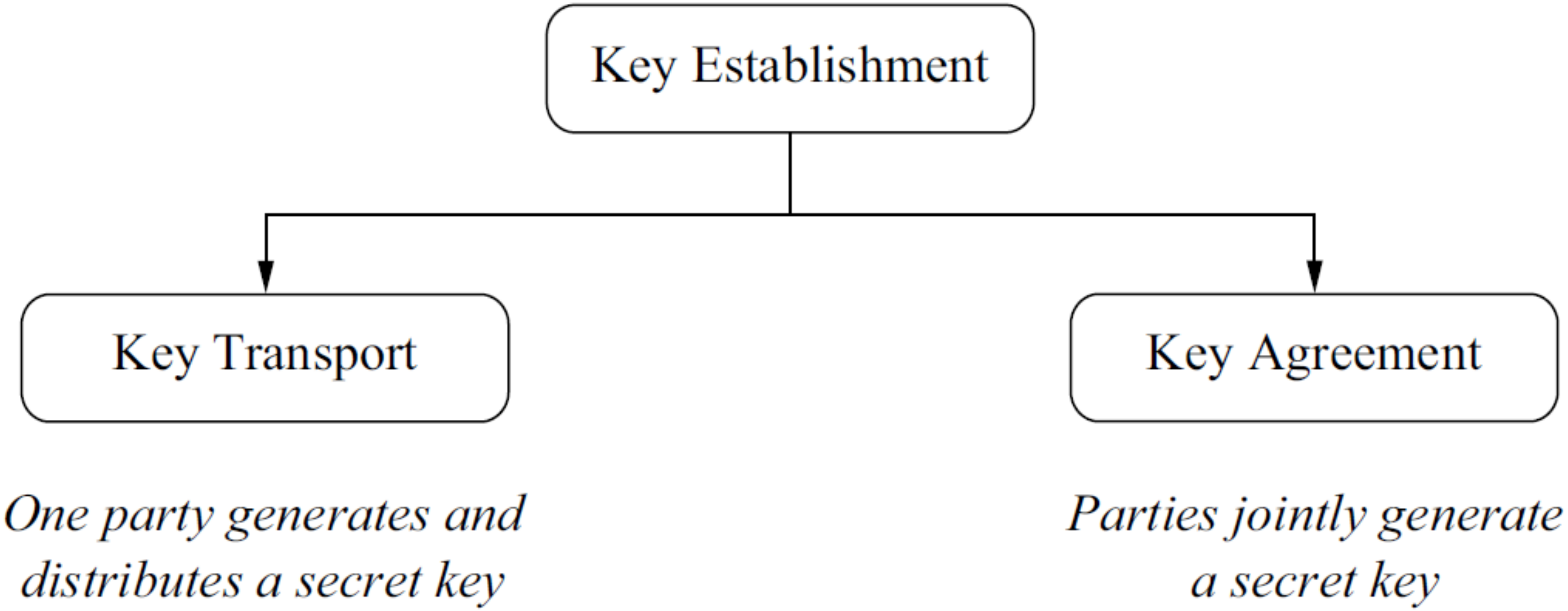
Key establishment over an insecure channel

- Fortunately, we can achieve that goal using asymmetric (public-key) algorithms, like RSA and Diffie-Hellman key exchange (DHKE)
  - In this lecture, we'll discuss DHKE in more detail
  - RSA will be covered in the next lecture

- Note: we also need to *authenticate* the channel when distributing the keys – this means verifying the identity of the source of the public keys
  - This prevents man-in-the-middle attacks
  - This can be achieved using digital signatures and certificates – we will discuss that topic in a future lecture

Key establishment

- The techniques for key establishment can be classified into two main groups:
  - Key transport methods
  - Key agreement methods



Key establishment examples

- Examples of a *key transport* method are the KDC method we described earlier, as well as RSA

- On the other hand, an example of a *key agreement* method is DHKE

More modular arithmetic…

- Before we discuss DHKE, we need to cover some additional modular arithmetic concepts needed for understanding DHKE
- Particularly the concept of *cyclic groups*

- We will need these concepts to understand discrete logarithm public-key algorithms, of which DHKE is one
  - These algorithms are based on the discrete logarithm problem, which is a very difficult problem to solve if the *domain parameters* are sufficiently large
  - We'll explain domain parameters later, when we describe DHKE

Group: definition (recap)

- A group $G = (S, \circ)$ is a set of elements, S, together with an operation $\circ$ which combines two elements of S. A group **must** have the following four properties:

- The group is **closed**, i.e. $a \circ b = c \in S$ for all $a, b \in S$
- The group operation is **associative**, i.e. $a \circ (b \circ c) = (a \circ b) \circ c$ for all $a, b, c \in S$
- There is an **identity** element $i \in S$ with respect to the operation $\circ$, such that $i \circ a = a \circ i = a$ for all $a \in S$
- For each $a \in S$, there exists an **inverse** element $a^{-1} \in S$, such that $a \circ a^{-1} = a^{-1} \circ a = i$

Commutative group (recap)

- A group $G = (S, \circ)$ is considered to be **commutative** if, in addition to the aforementioned four required properties, the group possesses the following property:

  $a \circ b = b \circ a$ for all $a, b \in S$

Order of a finite group (recap)

- We have seen sets with an infinite number of elements, such as $\mathbb{Z}$ and $\mathbb{R}$
- In cryptography, we are generally more interested in sets with a finite number of elements (i.e. finite sets) such as the set $\mathbb{Z}_m$, which has m elements
- The order $|G|$ of a finite group G is the number of elements in G

- E.g. the order of the group $G = (\mathbb{Z}_m, +)$ is $|G| = |\mathbb{Z}_m| = m$

Order of an element in a finite group (recap)

- The order $ord(a)$ of an element $a \in S$ in a group $G = (S, \circ)$ is the smallest positive integer k, such that

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

  where $i \in S$ is the identity element with respect to the operation $\circ$

- E.g.
  - The order of the element 1 in $G = (\mathbb{Z}_6, +) = (\{0, 1, 2, 3, 4, 5\}, +)$ is 6, since $1^6 = 1 + 1 + 1 + 1 + 1 + 1 \equiv 0 \bmod 6$, with element 0 being the additive identity element
  - The order of the identity element 0 in $G = (\mathbb{Z}_6, +)$ is 1, since $0^1 \equiv 0 \bmod 6$
  - The order of the element 2 in $G = (\mathbb{Z}_6, +)$ is 3, since $2^3 = 2 + 2 + 2 \equiv 0 \bmod 6$
  - The order of the element 1 in $G = (\mathbb{Z}_m, +)$ is m

Euler's phi function

e.g. 9 & 8, gcd is 1 btwn 2 numbers.

- The number of integers in the set $\mathbb{Z}_m$ that are *relatively prime to m* is denoted by $\Phi(m)$
  - An integer j is relatively prime to m if $gcd(j, m) = 1$

- E.g.
  - When $m = 6$, we have $\mathbb{Z}_6 = \{0, 1, 2, 3, 4, 5\}$. Then, we have $gcd(0, 6) = 6$; $gcd(1, 6) = 1$; $gcd(2, 6) = 2$; $gcd(3, 6) = 3$; $gcd(4, 6) = 2$ and $gcd(5, 6) = 1$. So, there are two integers that are relatively prime to 6 and thus, $\Phi(6) = 2$
  - When $m = 5$, we have $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$. Then, we have $gcd(0, 5) = 5$; $gcd(1, 5) = 1$; $gcd(2, 5) = 1$; $gcd(3, 5) = 1$ and $gcd(4, 5) = 1$. So, there are four integers that are relatively prime to 5 and thus, $\Phi(5) = 4$

Euler's phi function: comments

- For large *m*, calculating Euler's phi function $\Phi(m)$ by going through all elements of the set $\mathbb{Z}_m$ and computing the greatest common divisor is extremely slow
- However, <u>if we know the *factorization* of *m*</u>, then there is a much faster method to compute $\Phi(m)$

- This property is critical to the RSA algorithm – we'll revisit this issue when we discuss RSA in the next lecture
- Meanwhile, for this lecture we'll limit ourselves to the portions of Euler's phi function that are relevant to DHKE

Fast method to compute Euler's phi function

- Let m have the following *factorized* form:

  $m = p_1^{e_1} \cdot p_2^{e_2} \cdot \ \ldots \ \cdot p_n^{e_n}$, where the p_j are distinct prime numbers and the e_j are positive integers
- Then we have $\Phi(m) = \prod_{j=1}^{n}(p_j^{e_j} - p_j^{e_j - 1})$

- E.g. when $m = 240 = 2^4 \cdot 3 \cdot 5$, we have

  $\Phi(240) = (2^4 - 2^3) \cdot (3^1 - 3^0) \cdot (5^1 - 5^0) = 8 \cdot 2 \cdot 4 = 64$
- Note that computing $\Phi(240)$ by computing the gcd 240 times would have been much slower than using the formula above; but the formula requires that we know the factorization of m

The finite group $\mathbb{Z}^*_m$

- The set $\mathbb{Z}^*_m$ consists of all integers j = 1, …, m-1 for which $gcd(j, m) = 1$
  - In other words, $\mathbb{Z}^*_m$ consists of all integers from 1 to m-1 that are *relatively prime* to m
- $\mathbb{Z}^*_m$ forms a **commutative group** under <u>multiplication modulo *m*</u>, and the multiplicative identity element is 1
- The order of $\mathbb{Z}^*_m$ is $\Phi(m)$

- Note: $\mathbb{Z}^*_m$ is basically $(\mathbb{Z}^*_m, \cdot)$ and we use these two terms interchangeably (i.e. both terms refer to the <u>group</u> $\mathbb{Z}^*_m$)

The finite group $\mathbb{Z}^*_m$: example

- When m = 9, the group $\mathbb{Z}^*_9$ consists of the integers {1, 2, 4, 5, 7, 8}
- The multiplication table for $\mathbb{Z}^*_9$ is as follows:

× mod 9	**1**	**2**	**4**	**5**	**7**	**8**
1	1	2	4	5	7	8
2	2	4	8	1	5	7
4	4	8	7	2	1	5
5	5	1	2	7	8	4
7	7	5	1	8	4	2
8	8	7	5	4	2	1

- The multiplication modulo 9 operation satisfies the **closure**, **associativity**, **identity**, **inverse** and **commutativity** requirements
- The order of $\mathbb{Z}^*_9$ is $\Phi(9) = 3^2 - 3^1 = 6$, which is consistent with the number of elements in the set: {1, 2, 4, 5, 7, 8}

Cyclic groups

- A group G which contains an element α that has maximum order, i.e. $ord(\alpha) = |G|$, is called a *cyclic group*.

- Elements with maximum order are known as *generators* or *primitive elements*
  - Such an element is called a generator, because every element in the cyclic group can be generated by that element
  - i.e. if an element $\alpha \in G$ is a generator of the cyclic group G, then every element $a \in G$ can be written as $\alpha^k = a$ for some positive integer k

Cyclic groups: example

cyclic group: $\mathbb{Z}^*_{11}$

$\Rightarrow \alpha = 2$ is generator

$\alpha=2$	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
2	4	8	5	10	9	7	3	6	1

mod 11 $\Rightarrow$

$\therefore$ cyclic group as 2^n can generate all elements in $\mathbb{Z}^*_{11}$

- $\mathbb{Z}^*_{11}$ is an example of a cyclic group
- This group consists of the integers {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, and the order of the group is 10
- One generator of this group is 2, i.e. we have $\alpha = 2$

- Let's check that $\alpha = 2$ is indeed a generator, by checking the elements that are generated by powers of $\alpha = 2$:

  $\alpha = 2$; $\alpha^2 = 4$; $\alpha^3 = 8$; $\alpha^4 \equiv 5 \ mod\ 11$; $\alpha^5 \equiv 10 \ mod\ 11$;

  $\alpha^6 \equiv 9 \ mod\ 11$; $\alpha^7 \equiv 7 \ mod\ 11$; $\alpha^8 \equiv 3 \ mod\ 11$; $\alpha^9 \equiv 6 \ mod\ 11$; and

  $\alpha^{10} \equiv 1 \ mod\ 11$
- From the last result for α^{10}, we can see that $ord(\alpha) = 10$ and so, $\alpha = 2$ is a generator of $\mathbb{Z}^*_{11}$. Also, observe that $\alpha = 2$ is able to generate all elements of $\mathbb{Z}^*_{11}$.

The finite group $\mathbb{Z}^*_p$

- There is an important theorem concerning $\mathbb{Z}^*_m$ when $m = p$, where p is a prime integer:

  For every prime integer *p*, ($\mathbb{Z}^*_p$, ·) is a commutative finite cyclic group.

Notes:

- This theorem basically states that the multiplicative group of <u>every</u> prime field is cyclic, and that such a group has at least one generator
- The order of $\mathbb{Z}^*_p$ is $\Phi(p) = p - 1$

The discrete logarithm problem (DLP) in $\mathbb{Z}^*_p$

- Description of the problem:

  Given a commutative finite cyclic group of $\mathbb{Z}^*_p$ of order $p – 1$, a generator $\alpha \in \mathbb{Z}^*_p$, and another element $\beta \in \mathbb{Z}^*_p$, the DLP is the problem of determining an integer x, with $1 \le x \le p – 1$, such that

  $\alpha^x \equiv \beta \; mod \; p$

- In other words, in the DLP, we are attempting to find $x \equiv \log_\alpha \beta \; mod \; p$
- The integer x is called the discrete logarithm of β to the base α
- The integer x must exist because α is a generator, but finding such an integer x is very, very difficult if p is sufficiently large

One-way functions (w.r.t. asymmetric crypto)

- 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.
- Important note:
  - The 'one-way function' defined here for asymmetric cryptosystems is **not the same** as a 'one-way function' defined in the context of *hash functions*
  - In the case of a hash function, the 'one-way function' refers to a non-invertible function $f(x)$ where the inverse $f^{-1}(y)$ does **not** mathematically exist
  - In the case of asymmetric cryptosystems, the inverse of a one-way function does exist, but is extremely difficult to compute
- Most asymmetric crypto schemes of practical use, including RSA and DHKE, are based on a one-way function

One-way functions: examples

- The discrete logarithm problem (or equivalently, modular exponentiation) is an example of a one-way function
- Given α, x and p, computing β for which $\beta \equiv \alpha^x \mod p$ is relatively easy
- Conversely, given α, β and p, computing x for which $\alpha^x \equiv \beta \mod p$ is really difficult, for a large p

- Another example of a one-way function is the integer factorization problem; this is used as a mathematical basis for RSA

Diffie-Hellman key exchange (DHKE)

- DHKE is an asymmetric (public-key) cryptoalgorithm
  - The first asymmetric scheme
  - Proposed by Whitfield Diffie and Martin Hellman in 1976
  - It is used in many cryptographic protocols, like SSH, TLS and IPSec

Diffie-Hellman key exchange (DHKE)

- The fundamental idea behind DHKE:
  - Modular exponentiation in $\mathbb{Z}^*_p$, where p is a large prime integer, is a one-way function and modular exponentiation is commutative, i.e.

    $k = (\alpha^x)^y \equiv (\alpha^y)^x \mod p$, where k is the shared secret key between the two parties Alice and Bob
- This is the discrete logarithm problem in the group $\mathbb{Z}^*_p$
  - In other words, the discrete logarithm problem in $\mathbb{Z}^*_p$ forms the mathematical basis for DHKE

- DHKE actually consists of two protocols (can think of these as phases): setup and key exchange

Diffie-Hellman key exchange (DHKE)

a. Diffie-Hellman setup phase (picking of domain parameters):
   1. Choose a large prime p
   2. Choose a generator $\alpha \in \{2, 3, \ldots, p-2\}$
   3. Publish the domain parameters p and α (for Alice and Bob to use in the next phase – key exchange)

Diffie-Hellman key exchange (DHKE)

Alice		**Bob**
choose $a = k_{pr,A} \in \{2,\ldots,p-2\}$		choose $b = k_{pr,B} \in \{2,\ldots,p-2\}$
compute $A = k_{pub,A} \equiv \alpha^a \bmod p$		compute $B = k_{pub,B} \equiv \alpha^b \bmod p$
	$\xrightarrow{k_{pub,A}=A}$	
	$\xleftarrow{k_{pub,B}=B}$	
$k_{AB} = k_{pub,B}^{k_{pr,A}} \equiv B^a \bmod p$		$k_{AB} = k_{pub,A}^{k_{pr,B}} \equiv A^b \bmod p$

b. Diffie-Hellman key exchange phase (generate a joint secret key k_{AB}):

1. Alice picks a private key, $k_{pr,\ A} = a \in \{2, 3, \ldots, p\text{-}2\}$ and computes her public key $k_{pub,\ A} = A \equiv \alpha^a \ mod\ p$, then sends the public key over to Bob
2. Likewise, Bob picks a private key, $k_{pr,\ B} = b \in \{2, 3, \ldots, p\text{-}2\}$ and computes his public key $k_{pub,\ B} = B \equiv \alpha^b \ mod\ p$, then sends the public key over to Alice
3. Upon receiving each other's public key, Alice computes the joint secret key $k_{AB} = (k_{pub,\ B})^a \equiv \alpha^{ba} \ mod\ p$ and Bob computes $k_{AB} = (k_{pub,\ A})^b \equiv \alpha^{ab} \ mod\ p$

DHKE: example

Alice		**Bob**
choose $a = k_{pr,A} = 5$		choose $b = k_{pr,B} = 12$
$A = k_{pub,A} = 2^5 \equiv 3 \bmod 29$		$B = k_{pub,B} = 2^{12} \equiv 7 \bmod 29$
	$\xrightarrow{A=3}$	
	$\xleftarrow{B=7}$	
$k_{AB} = B^a \equiv 7^5 = 16 \bmod 29$		$k_{AB} = A^b = 3^{12} \equiv 16 \bmod 29$

DHKE: comments

- The joint secret key k_{AB} computed separately by Alice and Bob is the same, since modular exponentiation is commutative in $\mathbb{Z}^*_p$, that is, $\alpha^{ab} \bmod p \equiv \alpha^{ba} \bmod p$
- Only the public keys of Alice and Bob are transmitted over the insecure channel
- With a sufficiently large p, Eve/Oscar will have extreme difficulty in computing k_{AB}, despite knowing α, p, and the public keys A, B
- This joint secret key can then be used by Alice and Bob as the key for a symmetric cipher, such as 3DES and AES
    - Truncate the bits of the joint secret key as necessary for the required key length of the symmetric cipher

DHKE: comments

- Note that by itself, DHKE is not really a cipher; it doesn't have an encryption and decryption function
- But it can be extended to offer this functionality – with extensions, it becomes the *Elgamal* encryption scheme (we'll discuss this scheme in the next lecture)

- The discrete logarithm problem in $\mathbb{Z}^*_p$ is the one-way function used in DHKE; this problem can be generalized over any cyclic group
  - The one-way function used by elliptic curve cryptosystems is the generalized discrete logarithm problem over a cyclic group formed by an elliptic curve

Security of DHKE

- How might the attacker Eve/Oscar compromise the security of DHKE?
- The attacker only knows α, p, A and B
- The attacker can try the following:
    1. Solve the DLP by finding a value a such that $a \equiv \log_{\alpha} A \bmod p$
    2. Then compute the joint secret key $k_{AB} \equiv B^{a} \bmod p$

- As mentioned earlier, this is generally a difficult process for a large enough p

Security of DHKE

- Suppose Eve/Oscar wishes to solve the DLP to find the value a. There are a few methods available to the attacker who want to solve this problem:
- Brute Force Search
  - Keep trying values of a, in sequence, together with the generator α until the attacker obtains $\alpha^a = A \bmod p$
  - To defeat a brute force attack using today's computer technology, the order of $\mathbb{Z}^*_p$ needs to be around 2^{80}
  - Recall that $|\mathbb{Z}^*_p| = p-1$
  - This implies that at the bare minimum, we require p to be at least 80 bits long
  - But there are more powerful methods, listed in the next slide, which will increase the minimum length of p needed for security

Security of DHKE

- Shanks’ Baby-step Giant-step Method
- Pollard’s Rho Method
- Pohlig-Hellman Algorithm
- The Index-Calculus Method
  - This attack is powerful enough that in order to ensure that Eve/Oscar has to perform at least 2^{80} runs of this method, we require p to be at least 1024 bits long

Security of asymmetric cryptosystems

- In general, an asymmetric cryptosystem will require significantly more bits to achieve a similar security level as a symmetric cipher, as shown in the table below:

Algorithm Family	Cryptosystems	Security Level (bit)			
		80	128	192	256
Integer factorization	RSA	1024 bit	3072 bit	7680 bit	15360 bit
Discrete logarithm	DH, DSA, Elgamal	1024 bit	3072 bit	7680 bit	15360 bit
Elliptic curves	ECDH, ECDSA	160 bit	256 bit	384 bit	512 bit
Symmetric-key	AES, 3DES	80 bit	128 bit	192 bit	256 bit

Square and multiply algorithm

- One more thing before we conclude our discussion of DHKE…
- Consider the modular exponentiation operation for the **legitimate** parties Alice and Bob
  - They know *a* and *b*, and need to calculate $\alpha^a \bmod p$, $\alpha^b \bmod p$ and $\alpha^{ab} \bmod p$
- As we saw in the previous slide, the bit length of the operands need to be rather long, for security (1024 bits at a minimum)
  - This has a consequence on the computation time of the modular exponentiation operation
  - CPUs do **not** have an exponentiation instruction
  - The values *a* and *b* used in DHKE are on the order of 1024 bits; performing the multiplication operation of α by itself *a* and/or *b* times would require around 2^{1024} multiplications – that's infeasible
  - Fortunately, there is a much faster way – the square and multiply algorithm

Square and multiply algorithm

- Broadly speaking, the algorithm works on $x^e \ mod \ m$ as follows:

1. Initialize the result to x
2. Convert the exponent e to an unsigned binary number b_e
3. Scan the bit of b_e from the left to right (excluding the MSB), i.e. from the MSB (exclusive) to the LSB (inclusive) – each bit scanned counts as one iteration
   a. If the bit scanned is a ‘0’: just square the current result (i.e. multiply the current result by itself), modulo m
   b. Otherwise, the bit scanned is a ‘1’: square the current result modulo m, then multiply the new result by x, modulo m
4. Return the final result

Square and multiply algorithm

- A simple Python function that implements this algorithm :

```
# 'x' is the base, 'e' is the exponent and 'm' is the modulus (all are +ve)
def squareAndMult(x, e, m):
        result = x
        for i in bin(e)[3:]: # skip the "0b" part of the string and the MSB
                result = (result * result) % m    # square
                if (i == '1'):
                        result = (result * x) % m    # multiply

        return result
```

- The modulus operation is applied after every squaring operation and after every multiply operation to keep the intermediate results small

Square and multiply algorithm

- As an exercise, try applying the square and multiply algorithm to the following exponentiations:
  - 3^5 *mod* 11 (result is 1 *mod* 11)
  - 5^{18} *mod* 13 (result is 12 *mod* 13)
- Let's work through the algorithm for 3^5 *mod* 11:
  - Initialize the result to 3
  - We have $5_{10} = 101_2$, and thus two iterations to execute
  - First iteration:
    - Bit is a '0', so just square the current result: $3 \cdot 3$ *mod* 11 ≡ 9 *mod* 11
  - Second iteration:
    - Bit is a '1', so square the current result, then multiply by 3:
    - $9 \cdot 9$ *mod* 11 ≡ 81 *mod* 11 ≡ 4 *mod* 11 → $4 \cdot 3$ *mod* 11 ≡ 12 *mod* 11 ≡ 1 *mod* 11
  - We obtain 3^5 *mod* 11 ≡ 1 *mod* 11 as our final result