Week 5
Asymmetric Cryptography

THE UNIVERSITY OF SYDNEY

Dr. Thilini Dahanayaka

School of Computer Science, The University of Sydney

Announcements

- Student representative
- Assignment 1 - Any questions?
- Assignment 1 Submission Instructions - Please read the assignment description carefully.
- A break between tutorials and assignments

Agenda

- Asymmetric/Public-key cryptography
- Key exchange
- Elliptic curve arithmetic and elliptic curve cryptography (ECC)
  - Non-Examinable (for information only)

Recommended Reading

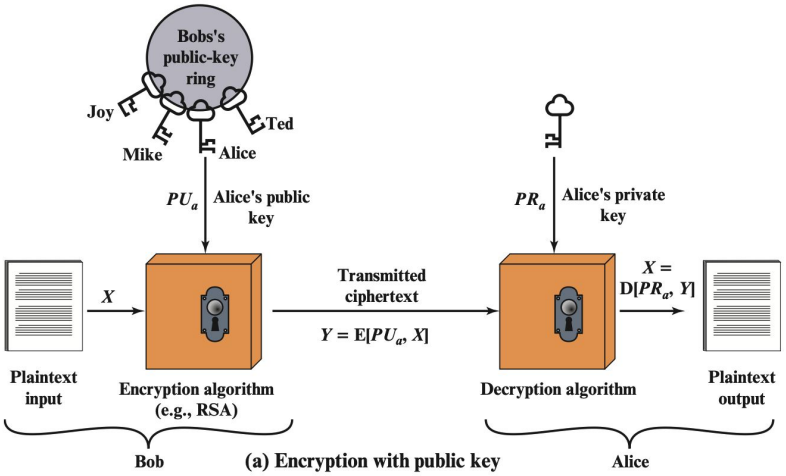
- Cryptography & Network Security - 7th Edition by William Stallings
  - **Chapter 9** – Public-key Cryptography and RSA
  - **Chapter 10** – Other Public-key Cryptosystems

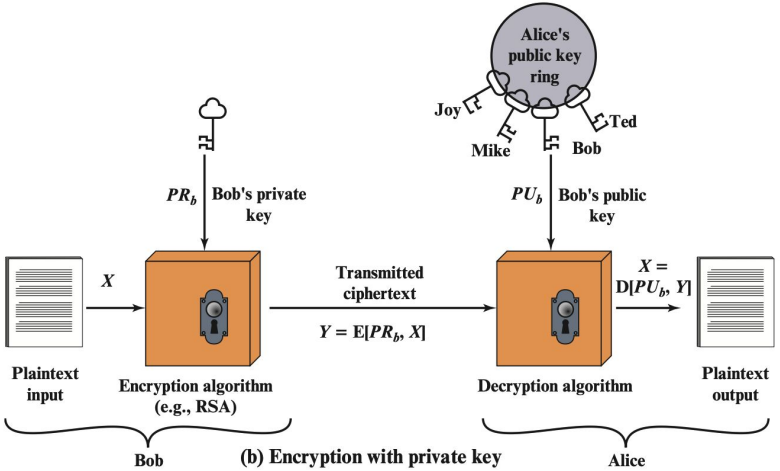
1. Asymmetric/Public Key Cryptography

Public-key Cryptography

- Symmetric cryptography is based on shared keys and scrambling of messages to achieve confusion and diffusion.
- Public-key cryptography is a change of paradigm in two respects:
  - Each participant has a public key (publicly distributed) and a private key (secret)
  - Anyone may use the receiver's public key to encrypt a message
  - Only the receiver (owner of the private key) can decrypt it
  - Hence also known as asymmetric cryptography
- This form of cryptography is based on mathematical problems with certain properties
- A number of mathematical problems are believed to have the desired properties.
  - But no proof yet!

Public-key Cryptography





Confidentiality - Encrypt by the public key and decrypt by the private key. Anyone can encrypt. But only the receiver can decrypt, because only they have the private key.

Origin Authentication - Encrypt by private key and decrypt by public key. Anyone can decrypt. But only the receiver could have encrypted it because only they have the private key.

Public-key Cryptography - Applications

- Encryption/Decryption, Digital Signatures, and Key Exchange
- We will discuss RSA, Diffie-Hellman, Elliptic Curve and Digital Signature Schemes (DSS)

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie–Hellman	No	No	Yes
DSS	No	Yes	No

Trapdoor Properties

- We are looking for a function with the following properties:
  - **Computationally fast** to compute the function value $f(x) = y$
  - **Computationally infeasible** to compute the inverse function $f^{-1}(y) = x$
  - **Unless** we are in possession of a piece of information that allows to speed it up dramatically.
  - Hence the name "*trapdoor function*".
  - **Corollary:** It must be **computationally infeasible** to compute the private key from the public key (without the trapdoor information)

Classic Trapdoor Candidate Problems

- **Discrete Logarithms in Modular Arithmetic:** It is computationally infeasible to compute the discrete logarithm modulo p, for certain p. (Diffie and Hellman, 1976)

Discrete Logarithm Problem (DLP): Given ***y, p, g*** satisfying the equation

$$y = g^x \bmod p$$

where ***p*** is a prime number and ***g*** is primitive root of ***p***, how can we find ***x***?

- **RSA Problem:** It is computationally infeasible to compute the e^{th} root of an integer modulo n, for certain n.

When $c = m^e \bmod n$ and ***c, e,*** and ***n*** are known can you find ***m***?

Both the Discrete Logarithm and the RSA problem have the property 'computationally infeasible'. However they become 'easy to compute' with additional information (i.e., private key).

Recap: Coprime Numbers & Euler's Phi function

- We say two numbers are coprime if they share only one divisor, and that divisor is 1
  - Example: 8 and 27 are coprime:
    - $8 = 2 \cdot 2 \cdot 2$, and $27 = 3 \cdot 3 \cdot 3$

- We define $\mathbb{Z}^*_n$ as all non-negative integers < n that are coprime to the integer n
  - Example: n = 12. Then, $\mathbb{Z}^*_{12} = \{1, 5, 7, 11\}$

- $\mathbb{Z}^*_n$ is useful: it is an Abelian group under multiplication. Euler's Phi Function (also called at the totient function) defines the number of elements in $\mathbb{Z}^*_n$.
  - For example, $\phi(12) = 4$.

- When n is a product of two primes p and q where $p \neq q$, there is an easier way to calculate $\phi(n)$
  - In that case $\phi(n) = (p-1)(q-1)$

The RSA Problem

- RSA is developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT
- The RSA problem allows us to build a trapdoor function. It is defined for a particular class of functions.

RSA Key Derivation and Encryption/Decryption

1) Let p and q, $p \neq q$ be prime
   - In practice: **very** large numbers
2) Define $n = p \cdot q$, (we call n as the modulus)
3) Use Euler's totient function to calculate $\phi(n)$
   $= (p - 1)(q - 1)$
4) Choose $1 < e < \phi(n)$, $\gcd(e, \phi(n)) = 1$
   - i.e., e is coprime with $\phi(n)$
   - One way: choose $e > \max(p, q)$ to be prime
   - Testing primality of an integer is fast
5) Calculate d such that $d \equiv e^{-1} \bmod \phi(n)$
   - i.e., d is the modular inverse of e
6) Now we have the keys
   - (e,n) is the public key
   - (d,n) is the private key
   - At this point we delete p, q, and $\phi(n)$ permanently

- Let m be the plaintext and c be the cipher text

7) Encryption: $c = m^e \bmod n$
8) Decryption: $m = c^d \bmod n$

RSA - Toy Example

1) Let p=7 and q=11
2) $n = p \cdot q \Rightarrow n = 77$
3) $\phi(n) = (p - 1)(q - 1) = (7-1)(11-1) = 6 \times 10 = 60$
4) Let's select e as 13
   - Note that 13 is less than 60 and is coprime with 60
5) Calculate d such that 13d mod 60 = 1
   - i.e., d is the modular inverse of e
   - We get d = 37
6) Now we have the keys
   - (13,77) is the public key
   - (37,77) is the private key

- Let 3 be the plaintext and c be the cipher text

7) Encryption: $c = 3^{13} \bmod 77 = 38$
8) Decryption: $m = 38^{37} \bmod 77 = 3$

The RSA Problem

- RSA encryption is defined as a function from $\mathbb{Z}/n\mathbb{Z}$ to $\mathbb{Z}/n\mathbb{Z}$:
  - RSA : $\mathbb{Z}/n\mathbb{Z} \rightarrow \mathbb{Z}/n\mathbb{Z}$ [Recall $\mathbb{Z}/n\mathbb{Z}$ is a commutative ring]
  - Note: n is not prime, but product of two primes.

- We had a very good reason to choose RSA to work over $\mathbb{Z}/n\mathbb{Z}$
  - Exponentiation in $\mathbb{Z}/n\mathbb{Z}$ is fast to compute [Recall exponentiation by squaring]
  - In other words, computing $c = m^e \mod n$ is very efficient

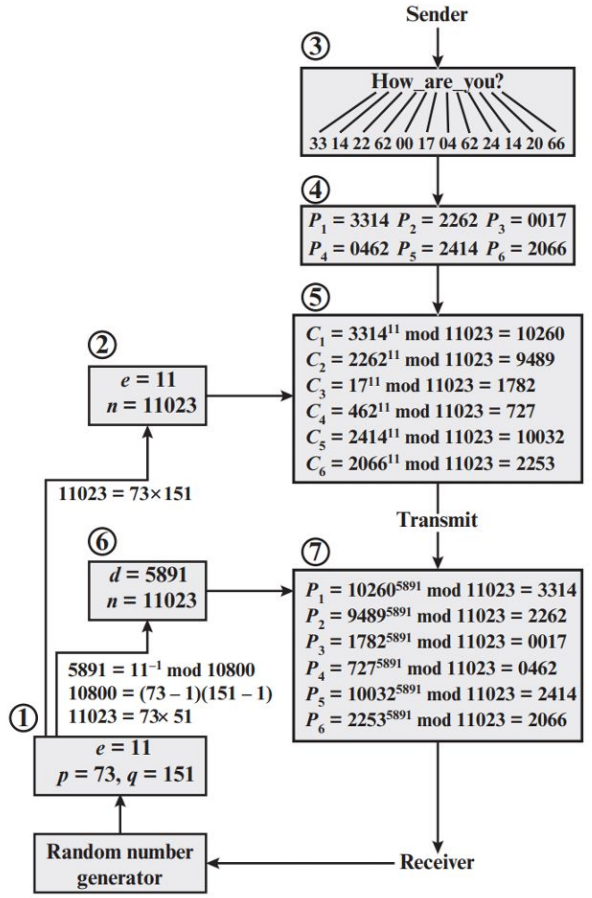
- But getting the $\mathbf{e^{th}}$ root of $\sqrt{\mathbf{c}}$ to get the original **m** is believed to be infeasible

- Note that RSA cannot uniquely encrypt values greater than n; such values are reduced modulo n, so decryption returns a value less than n, not the original input.

The RSA Problem

- By now it is clear that due to its mathematical nature, RSA can encrypt only numbers. More specifically only the number in $\mathbb{Z}/n\mathbb{Z}$.
- That means sender and receiver must use a mapping of numbers to characters to make this useful: a ⇒ 1, b ⇒ 2.

The RSA Problem - Full example



Source: Cryptography and Network Security (Seventh Edition - William Stallings)

Security of RSA

- The security of RSA rests on
  - Obtaining the e^{th} root of a number in $\mathbb{Z}/n\mathbb{Z}$ is computationally infeasible unless you know d
  - Factoring n = pq, for very large p and q, is computationally infeasible
  - This is for now though - quantum risks later. Australia plans to get rid of RSA by 2030
- p and q, $p \neq q$, must be very large.
- Encoding n needs thousands of bits (RSA key lengths: 2048 bit and more)
- p and q, $p \neq q$, must not be too close together (allows forms of fast factoring)
  - In practice, there are many more limiting factors
  - Not every key is a good key
- Never implement RSA yourself and use in production

Attacks on RSA

- Five possible approaches:
  - **Brute force:** This involves trying all possible private keys.
  - **Mathematical attacks:** There are several approaches, all equivalent in effort to factoring the product of two primes.
  - **Timing attacks:** These depend on the running time of the decryption algorithm.
  - **Hardware fault-based attack:** This involves inducing hardware faults in the processor that is generating digital signatures.
  - **Chosen ciphertext attacks:** This type of attack exploits properties of the RSA algorithm (e.g. Malleability)

Never use RSA without Padding

- RSA is not secure to use without padding:

Malleability: Malleability in RSA means that an attacker can modify a ciphertext in a predictable way so that the decrypted plaintext is also changed in a related, predictable way, without knowing the private key.

Example

- Attacker intercepts ciphertext c
- Attacker computes modified ciphertext and send to the receiver:
  - $c' \equiv c \cdot 2^e \bmod n$
- Receiver c' and obtains 2m
  - $(c')^d \bmod n \equiv (c \cdot 2^e \bmod n)^d \bmod n \equiv 2m$
  - A slightly hand-wavy proof is available in lecture notes

- Result is valid-looking, not random
- Receiver cannot detect that the message was altered
- **Key idea:** Vanilla RSA preserves structure ⟶ tampering yields a predictable, manipulated plaintext (not garbage)
- This is a generic result, the scalar multiplication can be any number, not just 2.
- Recall, this is a Chosen Ciphertext Attack (CCA)

Never use RSA without Padding

No ‘semantic security’

- An attacker can forward-compute (likely) messages and see if they match a given ciphertext.
- Deterministic encryption and public key is out -> Chosen Plain Text Attacks (CPA)
- Lack of randomness

Solution

- Sophisticated padding, which adds randomness on encryption and removes it on decryption.
- Padding is crucial, but non-trivial! Today, we use OAEP (Optimal Asymmetric Encryption Padding).

Hybrid Encryption

RSA has two key problems

- **Problem 1:** Public-key cryptography is slow - long keys!
  - RSA is 100-1000 times slower than AES

- **Problem 2:** Needs a mapping from characters to numbers to be useful

Solution: Hybrid Encryption

- Therefore, in general we don’t use RSA continuously. Rather we use hybrid cryptography using RSA to quickly establish a symmetric key.

Hybrid Encryption

Almost all public-key encryption uses hybrid encryption in practice.

- Generate random symmetric key k
- Encrypt actual message as $c_m = Enc_k(m)$
- Encrypt k as $c_k = Enc_{PK}(k)$ using receiver's public key
- Send (c_k , c_m)

2. Key Exchange

Key Exchange

- There are many public-key cryptosystems besides RSA.
- One of the best-known ones is **not used for encryption**, but for key exchange.
- Question to solve: ‘Can Alice and Bob want to establish a shared symmetric key without a **purely listening attacker** being able to obtain the key?’
- How do they do that?
- Finite fields come to our aid.

Diffie-Hellman

- First published algorithm that implemented public-key (trap-door) principle
- Became the basis for:
  - Diffie-Hellman Key Exchange
  - ElGamal encryption/decryption
  - Digital Signature Standard (DSA/DSS)
- Diffie-Hellman is based on finite fields

Discrete Logarithm Problem

- Just as with factorization and the e^{th} root before over our special ring, there is another trapdoor problem.
- It is computationally infeasible to compute a logarithm over a finite field, if the prime number used to construct the field is large enough.

Discrete Logarithm Problem (DLP): Given ***y, p, g*** satisfying the equation

$$y = g^x \bmod p$$

where ***p*** is a prime number and ***g*** is primitive root of ***p***, how can we find ***x***?

- But if you have one extra piece of information ('private key'), then it suddenly becomes 'easy'. How do we make use of it?

Finite Fields Revisited: Primitive Roots

- Let p be prime. We know $\mathbb{Z}/p\mathbb{Z}$ is a finite field.
- Elements of $\mathbb{Z}/p\mathbb{Z}$ are {0, 1, ... , p - 1}
- You can define a new group over the numbers
  - $\mathbb{Z}_p^*$ = {1, ... , p - 1} (with multiplication operation)
- You can also show: there is a special element $g \in \mathbb{Z}_p^*$
- Every number $a \in \mathbb{Z}_p^*$ is a power of g
- So, actually, $\mathbb{Z}_p^* = \{g^0 = 1, g^1 = \dots, g^{p-2} = \dots\}$
- g is called a primitive root (or generator) of $\mathbb{Z}_p^*$

Diffie-Hellman Key Exchange

Let p be prime, and g, a generator for $\mathbb{Z}_p^*$

Alice

- Choose random value a < p
- Compute $X = g^a \mod p$
- Send X to Bob
- Compute $k = Y^a \mod p$

Bob

- Choose random value b < p
- Compute $Y = g^b \mod p$
- Send Y to Alice
- Compute $k = X^b \mod p$

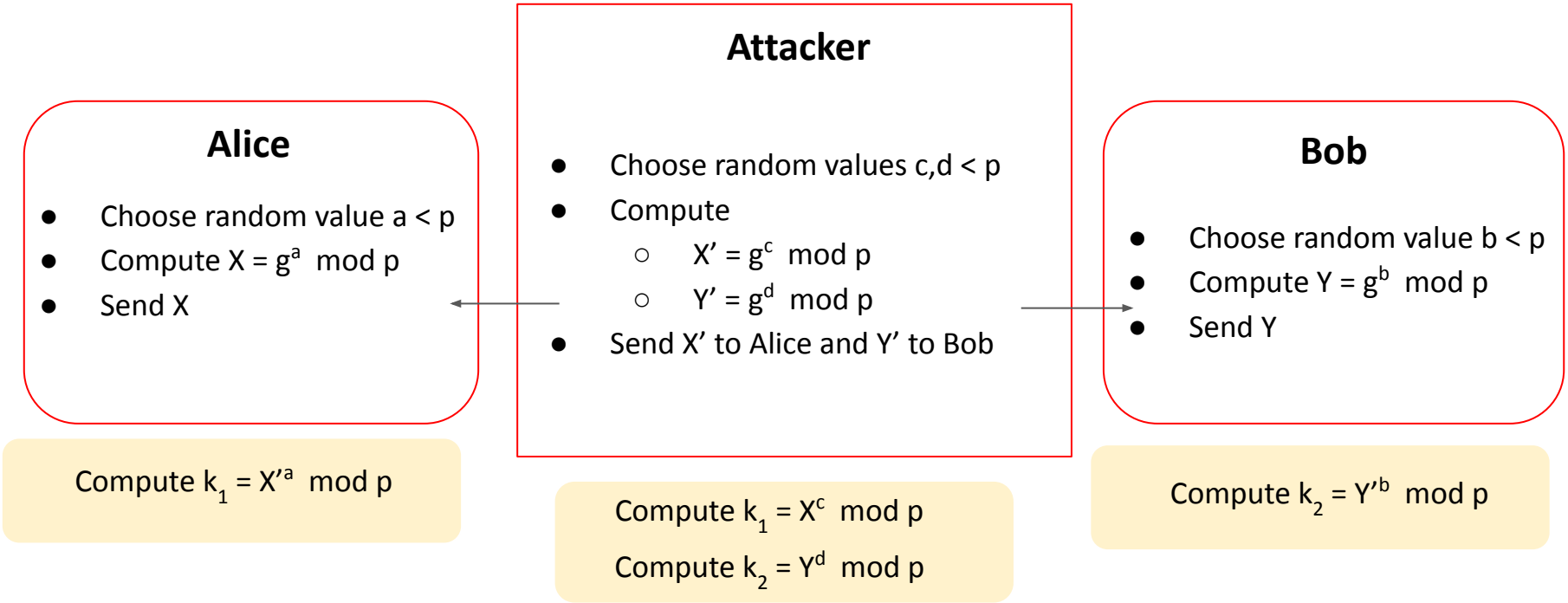
→ $Y^a \mod p = (g^b)^a = g^{ab} = (g^a)^b = X^b \mod p$

→ Both sides obtain the same k value

DH Key exchange allows key establishment over an insecure channel, under a passive attacker

Person-in-the-Middle Attack

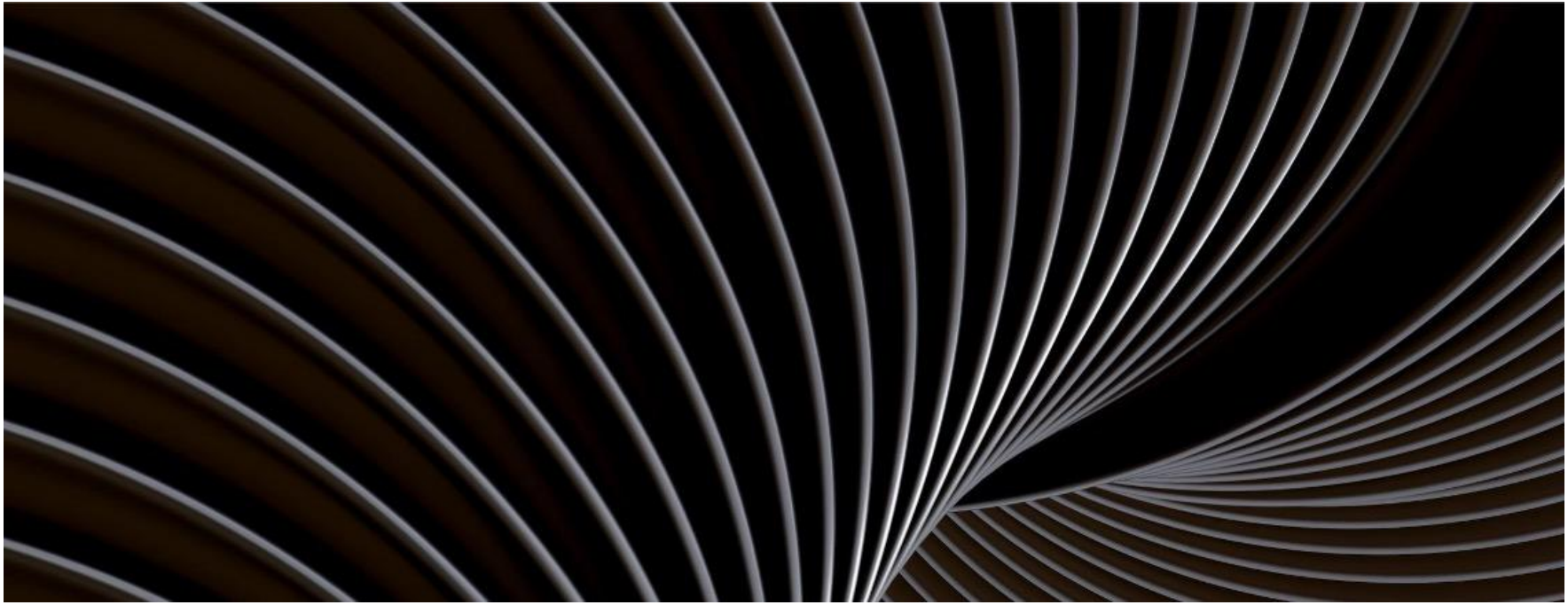
- When the attacker is active, the standard DH key exchange is not secure



Person-in-the-Middle-Attack

- The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants.
- **Solution:** Digital signatures and public-key certificates (more on this later)
- We will discuss those topics in the next lecture

3. Elliptic Curve Cryptography (Non-Examinable)



Elliptic Curve Cryptography (ECC)

The following is a very brief introduction.

- Our schemes so far are defined for rings and fields constructed with the help of the mod operation - i.e., we use modular arithmetic
- Elliptic curves E(a,b) are defined by equations of the form $y^2=x^3+ax+b$
- When defined over a finite field, they produce a finite set of points
- A group can be defined over these points using point addition
- This allows us to define a Discrete Logarithm Problem (ECDLP)
  - ECDLP is harder per bit than classical DLP
- Therefore, elliptic curve cryptography achieves the same security with much shorter keys
  - Typical ECC keys: 256 bits (very common)384 bits (high security)

Elliptic Curve Cryptography

Diffie-Hellman (classical setting)

- Group: $\mathbb{Z}_p^*$
- Operation: multiplication mod p
- Computation: $a^k \bmod p$ (repeated multiplication)
- Key idea: Security comes from the Discrete Logarithm Problem (DLP)

Key Transition

- So far, we have used groups derived from modular arithmetic
- We now construct a different Abelian group with the same goal

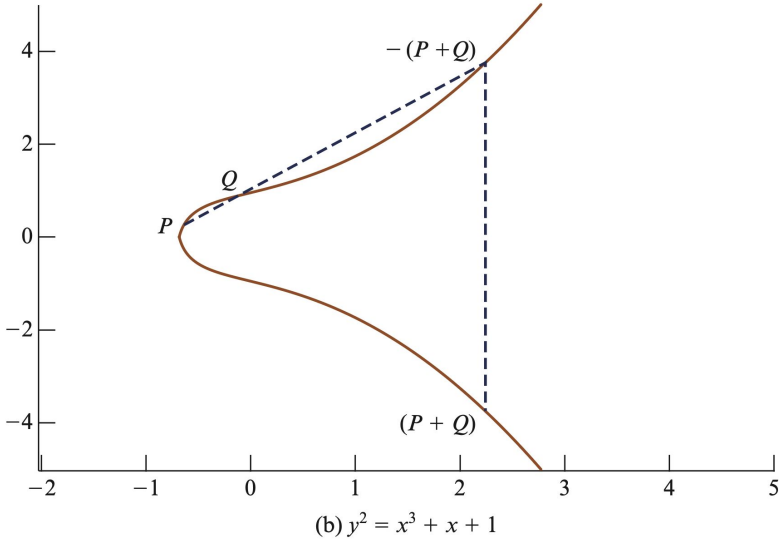
Elliptic Curve Cryptography (ECC): same idea, different group

- Elements: points on a curve over a finite field $\mathbb{Z}_p$
- Operation: point addition i.e., $kP = P + P + \cdots + P$
- Given P and Q=kP, it is computationally infeasible to recover k.
- **Note:** All arithmetic is performed modulo p (implicitly)

Elliptic Curves - Geometric Intuition

Geometric Intuition

- Consider E(1,1) which is $y^2=x^3+x+1$
- We define O: the point at infinity (identity element)
- This is an imaginary point to complete the math



(b) $y^2 = x^3 + x + 1$

Point addition

- Take two points P and Q on the curve
- Draw the line through P and Q
- The line intersects the curve at a third point
- Reflect this point across the x-axis: This gives P+Q

Key Properties

- The negative of a point: -P = (x,-y). i.e., reflection across x-axis
- Three points on a line sum to zero: P+Q+R = 0

Special Case related to ECC

- What if P=Q? Use the tangent line at P
- The tangent intersects the curve at another point. Reflect it across the x-axis. This gives 2P

Elliptic Curves over a Finite Field

Elliptic Curves over $\mathbb{Z}_p$

- We now define elliptic curves over a finite field $\mathbb{Z}_p$
- Now the curve equation becomes $E_p(a,b)$
  - $y^2=x^3+ax+b \pmod p$

Example curve over $\mathbb{Z}_{23}$

- Let's consider $E_{23}(1,1)$: $y^2=x^3+x+1 \pmod{23}$
- Check if point (x,y)=(9,7) lies on the curve:
  - $7^2 \equiv 9^3+10+1 \pmod{23}$
  - $49 \equiv 739 \pmod{23}$
  - $3 \equiv 3$ ✅

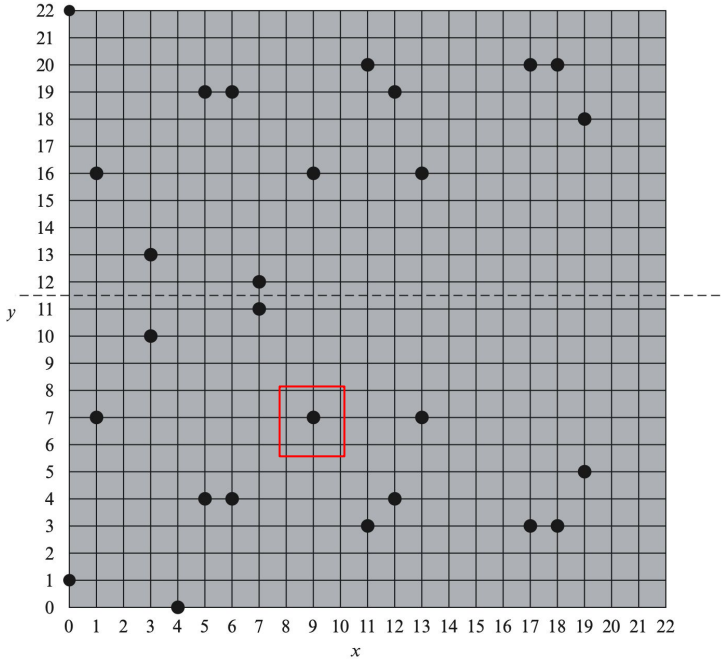
Points on the curve $E_{23}(1,1)$

- The curve consists of
  - All pairs of (x,y) satisfying the above equation
  - Plus O, the point at the infinity
- This forms a finite set of points

(0, 1)	(6, 4)	(12, 19)
(0, 22)	(6, 19)	(13, 7)
(1, 7)	(7, 11)	(13, 16)
(1, 16)	(7, 12)	(17, 3)
(3, 10)	(9, 7)	(17, 20)
(3, 13)	(9, 16)	(18, 3)
(4, 0)	(11, 3)	(18, 20)
(5, 4)	(11, 20)	(19, 5)
(5, 19)	(12, 4)	(19, 18)

Points (other than O) on the curve $E_{23}(1,1)$

Elliptic Curves over a Finite Field



Points (other than O) on the curve $E_{23}(1,1)$

Elliptic Curves over a Binary Field $GF(2^m)$

Transition from Prime Fields

- Previously, we used prime fields $\mathbb{Z}_p$ over an Elliptic Curve
- Now, we define curves over binary fields $GF(2^m)$ [Recall - We did something similar in AES]
- Elements are polynomials with coefficients in GF(2) (bit strings)
- Arithmetic is done modulo an irreducible polynomial

Curve Equation

- $y^2 + xy = x^3 + ax^2 + b$
- Addition/multiplication is in $GF(2^m)$

Field Generator

- Let g be a primitive element (generator) of $GF(2^4)$
- Every non-zero field element can be expressed as a power of g:
  - $GF(2^4)* = \{g^0, g^1, g^2, \ldots, g^{14}\}$

Elliptic Curves over a Binary Field GF(2^4)

Example Curve and point

- Let's consider the curve $y^2 + xy = x^3 + ax^2 + b$ where a= g^4 and b=1
- We can denote this curve as $E_{2^4}(g^4, 1)$
- Let's consider the test point (x,y) = (g^5, g^3) and verify whether it is on the curve

Verification

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

$(g^3)^2 + g^5 g^3 = (g^5)^3 + g^4(g^5)^2 + 1$

$g^6 + g^8 = g^{15} + g^{14} + 1$

1100+0101 = 0001+1001+001

1001 = 1001 ✅

⇒ Point (x,y) = (g^5, g^3) is on the curve

Properties of GF(2^4)

- We select f(x) = x^4+x+1 as the irreducible polynomial and g as 0010 (i.e., g=x).
- Now we can develop powers of g as below

$g^0 = 0001$	$g^4 = 0011$	$g^8 = 0101$	$g^{12} = 1111$
$g^1 = 0010$	$g^5 = 0110$	$g^9 = 1010$	$g^{13} = 1101$
$g^2 = 0100$	$g^6 = 1100$	$g^{10} = 0111$	$g^{14} = 1001$
$g^3 = 1000$	$g^7 = 1011$	$g^{11} = 1110$	$g^{15} = 0001$

- Let's calculator g4 which is x^4. x^4 is greater than the largest degree polynomial that can be supported by 4 bits (which is x^3). So we need to calculate. $x^4 = x^4 \bmod (x^4+x+1)$

$x^4 = x^4 \bmod (x^4+x+1)$

$= x^4 - (x^4+x+1)$

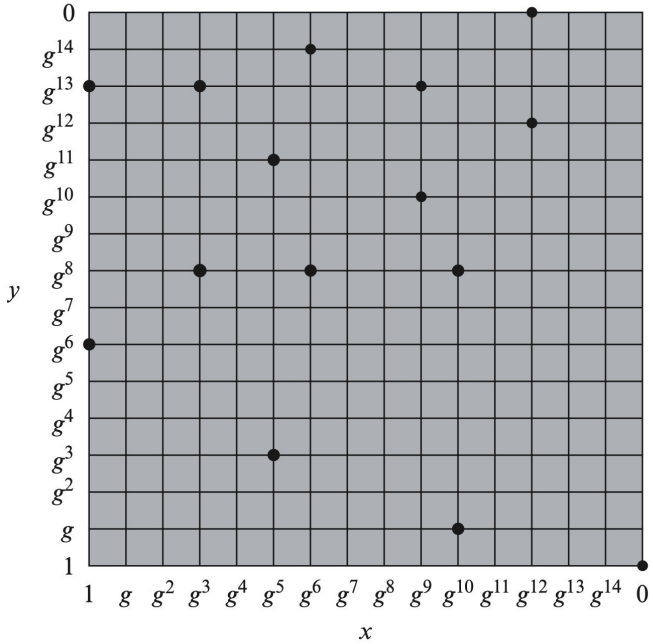
$= x^4 + (x^4+x+1) = x+1$

$= 0011$

1) In GF addition and subtraction is the same done through XOR.

2) Adding the same element twice results in zero.

Elliptic Curves over a Binary Field



The Elliptic Curve $E_{2^4}(\boldsymbol{g^4}, 1)$

Elliptic Curve Cryptography (ECC)

- Consider the equation Q = kP where Q, P $\in E_p(a, b)$ and k< p
  - It is hard to determine k given P, Q $\rightarrow$ discrete logarithm problem for elliptic curves

- Example: consider the group $E_{23}(9,17)$
  - Defined by: $y^2 \mod 23 = (x^3 + 9x + 17) \mod 23$
  - Find discrete logarithm k of Q = (4, 5) to base P = (16, 5)
  - The only way to do this is brute force
    - P = (16,5) $\Rightarrow$ 2P = (20,20) $\Rightarrow$ 3P = (14,14) $\Rightarrow$ 4P = (19,20) $\Rightarrow$5P = (13,10) $\Rightarrow$ 6P = (7,3) $\Rightarrow$ 7P = (8,7) $\Rightarrow$ 8P = (12, 17) $\Rightarrow$ 9P = (4,5) $\Rightarrow$ k is 9

- In practice, k would be very large, thus the brute-force approach infeasible

Elliptic Curve Diffie-Hellman Key Exchange (ECDH)

Global Public Elements

$E_q(a, b)$	elliptic curve with parameters a, b, and q, where q is a prime or an integer of the form 2^m
G	point on elliptic curve whose order is large value n

User A Key Generation

Select private n_A	$n_A < n$
Calculate public P_A	$P_A = n_A \times G$

User B Key Generation

Select private n_B	$n_B < n$
Calculate public P_B	$P_B = n_B \times G$

Calculation of Secret Key by User A

$K = n_A \times P_B$

Calculation of Secret Key by User B

$K = n_B \times P_A$

- Both users A and B (and any attacker) know a,b,q,G,n
- These are system-wide public parameters
- G is a base point (generator) on the curve
- n is the order of G. That is nG=O. Usually n is chosen to be a large prime

- Public keys are computed as: $P_A = n_A \times G$, $P_B = n_B \times G$
- This scalar multiplication is efficient
- Private keys n_A and n_B are kept secret
- Even with P_A and G, attacker cannot compute n_A
- Security relies on the hardness of the Elliptic Curve Discrete Logarithm Problem

- Both parties compute the same key

$n_A * P_B = n_A * (n_B * G) = n_B * (n_A * G) = n_B * P_A$

EC El-Gamal Encryption

- Public parameters: Elliptic curve $E_q(a,b)$, generator point G with large order n, message encoded as curve point P_m

Alice (sender)

1. Select private key n_A where $n_A < n$
2. Compute public key $P_A = n_A \times G$
3. Choose random integer k (new for each message)
4. Send ciphertext: $C_m = \{kG, P_m + kP_B\}$

kP_B acts as a one-time mask on P_m . This is the embedded ECDH shared secret. P_B is B's public key

Bob (receiver)

1. Select private key n_B where $n_B < n$
2. Compute public key $P_B = n_B \times G$ (shared publicly)
3. Receive $C_m = \{ kG, P_m + kP_B \}$
4. Recover: $P_m = (P_m + kP_B) - n_B \times (kG)$

Works because $n_B \times (kG) = k \times (n_BG) = kP_B$ i.e., The mask cancels out

Why is k random and chosen fresh each time? Reusing k for two different messages leaks information. An attacker can XOR the two ciphertexts and eliminate the mask, exposing the relationship between messages. A fresh k per message prevents this.

EC El-Gamal Encryption - Example

- Consider the global (system-level) parameters: $E_{257}(0, -4)$: $y^2 = x^3 - 4$, $G(2, 2)$
- Encryption
  - Bob's private key $n_B = 101$, Bob's public key $P_B = n_B G = 101(2, 2) = (197, 167)$
  - Alice wants to send $P_m = (112, 26)$ and choose k = 41. Compute $kG = 41(2,2) = (136, 128)$
  - Alice encrypts: $kP_B = 41(197, 167) = (68, 84)$ and $P_m + kP_B = (112, 26) + (68, 84) = (246, 174)$.
  - Alice sends: $C_m = \{C_1, C_2\} = \{(136, 128), (246, 174)\}$ to Bob
- Decryption
  - Bob computes: $C_2 - n_B C_1 = (246, 174) - 101(136, 128) = (246, 174) - (68, 84) = (112, 26)$

Security of ECC

- The security of ECC depends on how difficult it is to determine k given kP and
  - Elliptic curve logarithm problem
- The table compares various algorithms by showing comparable key sizes in terms of computational effort for cryptanalysis
- Pay attention to key length of RSA and ECC

Symmetric Key Algorithms	Diffie–Hellman, Digital Signature Algorithm	RSA (size of n in bits)	ECC (modulus size in bits)
80	$L = 1024$ $N = 160$	1024	160–223
112	$L = 2048$ $N = 224$	2048	224–255
128	$L = 3072$ $N = 256$	3072	256–383
192	$L = 7680$ $N = 384$	7680	384–511
256	$L = 15{,}360$ $N = 512$	15,360	512+

Note: L = size of public key, N = size of private key.

Post Quantum Cryptography (PQC)

- **Breaking Asymmetric Cryptography**
  - Shor's algorithm, can efficiently factor large numbers and compute discrete logarithms
  - RSA, Diffie-Hellman Key Exchange, ECC will be vulnerable

- **Symmetric Cryptography is More Resilient**
  - Grover's algorithm, provides a quadratic speedup for brute-forcing the keys of symmetric ciphers
  - Symmetric key's effective security level is halved
  - A 256-bit key would offer security comparable to a 128-bit key against a quantum adversary
  - Doubling Key Sizes as a solution.

- **Post Quantum Cryptography (PQC)**
  - Lattice-based, Code-based, Hash-based cryptography

Recap

- **We discussed:**
  - Public-key cryptosystem and its applications
  - RSA algorithm and attacks on RSA
  - Hybrid encryption
  - Diffie-Hellman key exchange and person-in-the-middle attack
  - An overview of the elliptic curves and Elliptic curve cryptography
  - ECC Diffie-Hellman key exchange
  - The security of ECC
  - Post Quantum Cryptography (PQC)