Cryptography

Diffie—Hellman Key Exchange, MiMA
ECC

M S Vilkhu

22 Nov 2024(C1/C3/C5)

Diffie—Hellman Key Exchange, MiMA ECC

Other Public-key Cryptosystems

- **Diffie—Hellman Key Exchange**
- **Elliptic Curve Cryptography**

- Elgamal Cryptographic System
- Pseudorandom Number Generation Based on an Asymmetric Cipher

Other Public-key Cryptosystems

- **Diffie—Hellman Key Exchange**
  - The Algorithm Key
  - Exchange Protocols
  - Man-in-the-Middle Attack
- **Elliptic Curve Cryptography**
  - Analog of Diffie—Hellman
  - Key Exchange Elliptic Curve
  - Encryption/Decryption
  - Security of Elliptic Curve Cryptography

- **Elgamal Cryptographic System**
- **Pseudorandom Number Generation Based on an Asymmetric Cipher**
  - PRNG Based on RSA PRNG
  - Based on Elliptic Curve Cryptography

Diffie—Hellman Key Exchange

- The **Diffie-Hellman Key Exchange** (DHKE) is a **cryptographic protocol** that **allows two parties to securely establish a shared secret key** over an **insecure communication channel**.
- This **shared key** can then be used to **encrypt subsequent communications** using **symmetric encryption**.

Diffie—Hellman Key Exchange

Overview of Diffie-Hellman Key Exchange

- The Diffie-Hellman key exchange is based on **mathematical properties** of modular **arithmetic and discrete logarithms**, which are **computationally difficult to reverse**. Here's a step-by-step explanation of how it works:

1. Setup: Public Parameters:

- Choose a **large prime number p** and a **generator g** (a primitive root modulo p).
- Both **p and g are publicly known** and can be **shared over an insecure channel**.

2. Private Keys:

- **Alice** and **Bob**, who want to communicate, each **choose a private key**:
  - **Alice** chooses a **private key a**, which is a random integer.
  - **Bob** chooses a **private key b**, also a random integer.
- These **private keys are kept secret** and are **not shared**.

Diffie—Hellman Key Exchange

Overview of Diffie-Hellman Key Exchange

3. Compute Public Keys:

- **Each party calculates a public key** based on their **private key** and the **generator** g:
  - Alice computes **her public key** $A=g^a \bmod p$ sends **A to Bob**.
  - Bob computes his **public key** $B=g^b \bmod p$ sends **B to Alice.**

4. Compute the Shared Secret Key:

- Upon **receiving each other's public keys**, Alice and Bob each **compute the shared secret key:**
  - Alice computes $s=B^a \bmod p$.
  - Bob computes $s = A^b \bmod p$.
- Since $s=g^{ab} \bmod p$ both **computations yield the same result**, so both Alice and Bob have now established a **shared secret key s**, which can be used as a **symmetric encryption key.**

Diffie—Hellman Key Exchange

Why Diffie-Hellman Works

- The security of the Diffie-Hellman key exchange relies on the **difficulty of computing discrete logarithms**.
- Given g, p, and $g^a \bmod p$, it's computationally hard to determine **a** (the discrete logarithm problem) for sufficiently large values of p.
- An attacker who intercepts the public keys A and B would need to solve this problem to determine $s=g^{ab} \bmod p$, making it impractical to derive the shared key.

Diffie—Hellman Key Exchange

- **Example of Diffie-Hellman Key Exchange**
- Let's work through a simplified example:

1. Choose Public Parameters:

- Let p=23 (a prime number) and g= 5 (a generator modulo p).

2.Alice's Private Key and Public Key:

- Alice selects a **private** key a=6.
- Alice computes her **public** key: $A=g^a \bmod p= 5^6 \bmod 23 =8$
- Alice sends A=8 to Bob.

Prime & generator p=23 g=5
Alice a=6 A = 8

Diffie—Hellman Key Exchange

- **Example of Diffie-Hellman Key Exchange**

3. Bob's Private Key and Public Key:

- Bob selects a private key b=15.
- Bob computes his public key: $B=g^b \bmod p=5^{15} \bmod 23=19$
- Bob sends B=19 to Alice.

4. Compute the Shared Secret Key:

- Alice computes $s=B^a \bmod p=19^6 \bmod 23=2$.
- Bob computes $s=A^b \bmod p=8^{15} \bmod 23=2$.
- Both Alice and Bob have derived the shared secret key s=2.

Prime & generator p=23 g=5
Alice a=6 A = 8
Bob b=15 B = 19

s=2

Diffie—Hellman Key Exchange

- **Advantages and Limitations**

Advantages:

- Provides a method to **securely share** a symmetric key over an **insecure channel.**
- Does not require **a prior shared** secret.

Limitations:

- Diffie-Hellman is vulnerable to **Man-in-the-Middle (MitM) attacks if authentication is not used**, as an attacker can **intercept and replace public keys**.
- **Without additional authentication** (such as digital signatures or certificates), both parties **cannot be certain** they are communicating with the intended person.

Man-in-the-Middle (MitM) attack on the Diffie-Hellman Key Exchange

- **Man-in-the-Middle (MitM) attack** on the **Diffie-Hellman Key Exchange** occurs when an **attacker intercepts and potentially alters the communication** between two parties (Alice and Bob) during the **key exchange process**. The attacker **tricks both parties** into establishing **separate shared keys** with the attacker **instead of with each other**, allowing the **attacker to eavesdrop or manipulate** their communication.
- **How a Diffie-Hellman MitM Attack Works**

1. Setup:

1. Alice and Bob intend to establish a shared secret key using the Diffie-Hellman protocol over an insecure channel.
2. An attacker (**Eve**) is positioned to intercept messages between Alice and Bob.

Man-in-the-Middle (MitM) attack on the Diffie-Hellman Key Exchange

- **Attack Steps**:
- **Step 1: Intercept Public Keys**:
  - Alice sends her public key $A=g^a \bmod p$ to Bob, but Eve intercepts it.
  - Similarly, Bob sends his public key $B=g^b \bmod p$ but Eve intercepts it as well.
- **Step 2: Replace Public Keys**:
  - Eve generates her **own private key e**.
  - Eve sends $E_A=g^e \bmod p$ to Bob, pretending it came from Alice.
  - Eve sends $E_B=g^e \bmod p$ to Alice, pretending it came from Bob.

Man-in-the-Middle (MitM) attack on the Diffie-Hellman Key Exchange

- **Attack Steps**:
- **Step 3: Compute Separate Shared Secrets**:
  - Alice computes $S_A = E_b^{\ a} \bmod p$ as the shared secret, believing E_B is Bob's public key.
  - Bob computes $S_B = E_a^{\ b} \bmod p$ as the shared secret, believing E_A is Alice's public key.
  - Eve computes both $S_{AE} = A^e \bmod p$ (shared secret with Alice) and $S_{BE} = B^e \bmod p$ (shared secret with Bob).
- **Step 4: Eavesdrop or Manipulate Communication**:
  - Eve decrypts messages sent by Alice using S_{AE}, reads or modifies them, and re-encrypts them using S_{BE} before sending them to Bob (and vice versa).

Man-in-the-Middle (MitM) attack on the Diffie-Hellman Key Exchange

- **Mitigation**

To prevent such attacks, Diffie-Hellman key exchange must incorporate **authentication methods** like:

- **Digital certificates** or **signatures**.
- **Pre-shared keys (PSK)**.
- **TLS/SSL protocols**, which include authentication mechanisms.

Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography (ECC)

- **Elliptic Curve Cryptography (ECC)** is a public-key cryptography method **based** on the **algebraic structure of elliptic curves over finite fields**.
- ECC offers **high levels of security** with **smaller key sizes** than other cryptographic methods, making it **efficient** and **secure** for modern **encryption, digital signatures, and key exchange.**

Elliptic Curve Cryptography (ECC)

- **Key Concepts of Elliptic Curve Cryptography**

1. **Elliptic Curves**:
    1. An elliptic curve is represented by an equation of the form:

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

       where a and b are constants that satisfy certain conditions to ensure the curve **is non-singular (no cusps or self-intersections).**
    1. For cryptographic purposes, these **curves are usually defined over a finite field**, meaning x and y take **values from a set of numbers within a finite range**.
3. **Points on the Curve**:
    1. Points on the curve represent possible values for (x,y) pairs that satisfy the elliptic curve equation.
    2. A **special point** known as the **point at infinity** acts as the identity element for the group structure.

       In elliptic curve cryptography (ECC), the **zero point** (often denoted as O or the **point at infinity**) is a special point that serves as the **identity element** for the elliptic curve's group operation. It plays a crucial role in defining the arithmetic of points on the elliptic curve.

Elliptic Curve Cryptography (ECC)

- **Key Concepts of Elliptic Curve Cryptography**

3. Elliptic Curve Group Operations:

- **Point Addition**: Adding two points on the curve produces a third point on the curve.
- **Point Doubling**: Doubling a point (adding it to itself) results in another point on the curve.
- These operations allow for the creation of a "multiplicative" structure, which forms the basis of ECC.

4. Scalar Multiplication:

- In ECC, **scalar multiplication** is the process of adding a point to itself repeatedly. For example, multiplying a point P by a scalar k is equivalent to adding P to itself k times:

  $kP=P+P+\cdots+P$ (k times)

- **Scalar multiplication** is the core of ECC because it's **easy to compute** but **difficult to reverse**, which **provides security** for ECC-based cryptography.

Elliptic Curve Cryptography (ECC)

- **Key Concepts of Elliptic Curve Cryptography**

3. Elliptic Curve Group Operations:

- **Point Addition**: Adding two points on the curve produces a third point on the curve.
- **Point Doubling**: Doubling a point (adding it to itself) results in another point on the curve.
- These operations allow for the creation of a "multiplicative" structure, which forms the basis of ECC.

4. Scalar Multiplication:

- In ECC, **scalar multiplication** is the process of adding a point to itself repeatedly. For example, multiplying a point P by a scalar k is equivalent to adding P to itself k times:

  $kP=P+P+\cdots+P$ (k times)

- **Scalar multiplication** is the core of ECC because it's **easy to compute** but **difficult to reverse**, which **provides security** for ECC-based cryptography.

Elliptic Curve Cryptography (ECC)

- **How ECC Is Used in Cryptography**
- ECC enables several cryptographic functions similar to traditional public-key algorithms like RSA but **with smaller key sizes for comparable security**:

1. Key Exchange:

1. ECC supports secure key exchange through protocols like **Elliptic Curve Diffie-Hellman (ECDH)**, which allows two parties to establish a shared secret over an insecure channel.
2. Example: Each party generates a public key from their private key and an agreed-upon elliptic curve point. By sharing the public keys and performing scalar multiplication, both parties can independently derive the same shared secret.

2. Digital Signatures:

3. ECC is used for creating **digital signatures** through algorithms like **Elliptic Curve Digital Signature Algorithm (ECDSA)**.
4. ECDSA allows a user to sign a message with their private key, and others can verify the signature using the user's public key, ensuring data integrity and authenticity.

Elliptic Curve Cryptography (ECC)

- **How ECC Is Used in Cryptography**

3. Encryption:

1. **Elliptic Curve Integrated Encryption Scheme (ECIES)** combines ECC with symmetric encryption to create a secure hybrid encryption scheme.
2. ECIES enables encryption of a message using a recipient's public key, allowing only the intended recipient, who possesses the corresponding private key, to decrypt it.

Elliptic Curve Cryptography (ECC)

- **Benefits of ECC**

1. **Efficiency**: ECC provides **strong security with smaller keys** (e.g., a **256-bit ECC key is roughly equivalent in security to a 3072-bit RSA key**).
2. **Speed**: Smaller keys **reduce computational load**, making **ECC faster for encryption, decryption, and key generation.**
3. **Bandwidth Saving**: **Smaller keys** mean **smaller data exchanges**, which is beneficial for resource-constrained devices and networks.
4. **Security**: ECC is based on the **Elliptic Curve Discrete Logarithm Problem (ECDLP)**, **a hard-to-solve problem that gives ECC its strength**.

Elliptic Curve Cryptography (ECC)

- **Example of Elliptic Curve Diffie-Hellman (ECDH) Key Exchange**

1. **Agree on Curve**: Alice and Bob agree on an elliptic curve and a base point **G** on that curve.
2. **Generate Private Keys**:
    1. Alice selects a private key a and computes her public key A=aG.
    2. Bob selects a private key bbb and computes his public key B=bG.
3. **Exchange Public Keys**:
    1. Alice and Bob exchange their public keys A and B.
4. **Compute Shared Secret**:
    1. Alice computes S=aB=abG.
    2. Bob computes S=bA=abG.
    3. Both end up with the same shared secret S, which can be used as a symmetric key for encryption.

Elliptic Curve Cryptography (ECC)

- **Applications of ECC**
- ECC is widely used in protocols that require **secure communication**:
- **TLS/SSL** (for **secure web** browsing)
- **SSH** (for **secure remote** access)
- **Bitcoin and other cryptocurrencies** (for **signing** transactions)
- **Smart cards and mobile devices** (for lightweight, efficient encryption)

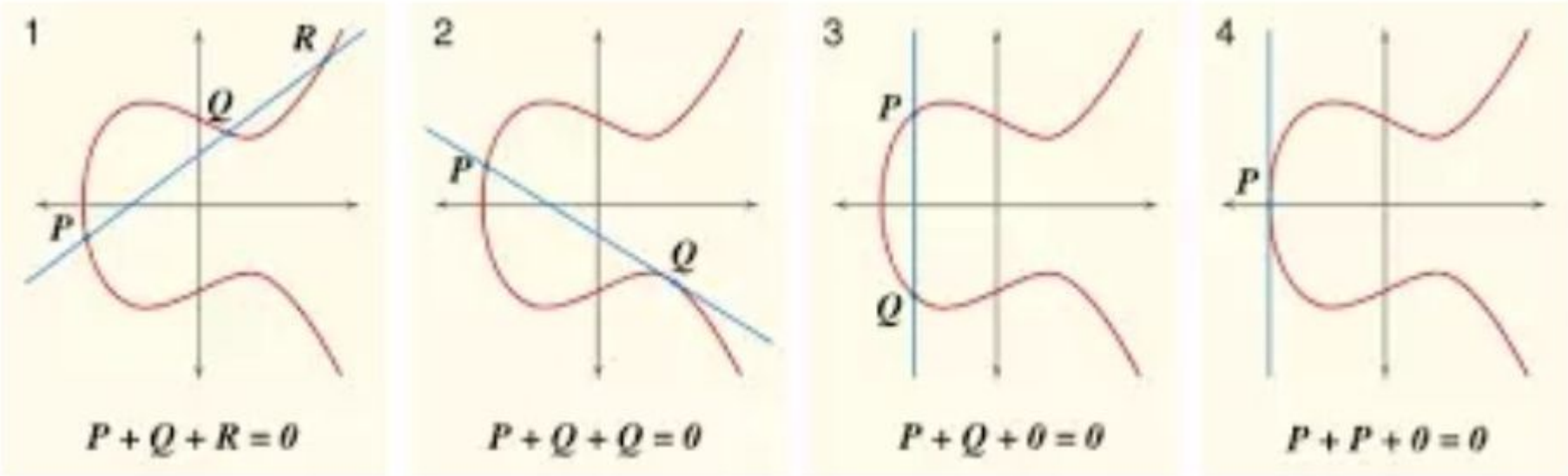
ECC

- Elliptic curve cryptography is used to implement public key cryptography. It was discovered by Victor Miller of IBM and Neil Koblitz of the University of Washington in the year 1985. ECC popularly used an acronym for Elliptic Curve Cryptography. It is based on the latest mathematics and delivers a relatively more secure foundation than the first generation public key cryptography systems for example RSA.

- Concepts of ECC is more difficult to explain than RSA or Diffie-Hellman.
- Concepts involves - abelian group.
- Concept of elliptic curves defined over the real numbers.
- Elliptic curves defined over finite fields.

ECC

- Examples of curves and some equations



ECC Uses

- Websites make extensive use of ECC to secure customers' hypertext transfer protocol connections.
- It is used for encryption by combining the key agreement with a symmetric encryption scheme.

ECC

- Elliptic curve cryptography makes **use of elliptic curves** in which the **variables and coefficients are all restricted to elements of a finite field**.
- **Two families of elliptic curves are used** in cryptographic applications:
  - **prime curves over Zp** and
  - **binary curves** over GF(2^m)
- For **prime curve** over Zp, we use a **cubic equa**tion in which the **variables and coefficients** all take on values in the **set of integers** from 0 through p — 1 and in which **calculations are performed modulo p**.
- For a **binary curve** defined over GF(2^m), the **variables** and **coefficients** all take on values in GF(2^m), and in calculations are performed GF(2^m).
- [FERN99] points out that **prime curves** are best for **software applications**, because the extended bit-fiddling operations needed by binary curves are not required; and
- that **binary curves** are best for **hardware applications**, where it takes remarkably **few logic gates to create a powerful, fast cryptosystem**.

Note: More you can explore Stallings book.

ECC

- The **addition operation** in ECC is the **counterpart** of **modular multiplication** in RSA, and **multiple addition** is the **counterpart** of **modular exponentiation**.
- To **form a cryptographic system using elliptic curves**, we need to **find a "hard problem"** corresponding to **factoring the product of two primes** or **taking the discrete logarithm**.
- Consider the equation **Q = kP** where Q, P ∈ Ep(a, b) and k < p.
  - It is relatively **easy** to **calculate Q** given k and P,
  - but it is **hard** to determine **k** given Q and P.
- This is called the discrete logarithm problem for elliptic curves.

ECC

- an example
- Consider the group E_{23} (9, 17)
- defined by the equation $Y^2 \bmod 23 = (x^3 + 9x + 17) \bmod 23$.
- What is the discrete logarithm k of Q (4, 5) to the base P = (16, 5)?
- The **brute-force** method is to compute multiples of P until Q is found. Thus,

  P = (16, 5); 2P = (20, 20); 3P = (14, 14); 4P = (19, 20); 5P = (13, 10) ; 6P = (7, 3) ;

  7P = (8, 7) ; 8P = (12, 17); 9P = (4, 5)
- Because 9P = (4, 5) = Q, the discrete logarithm Q = (4, 5) to the base P = (16, 5) is k = 9.
- In a real application, **k would be so large** as to make the brute force approach **infeasible**.

Security of Elliptic Curve Cryptography

- The **security of ECC depends** on how difficult it is to determine k given **kP and P**. This is referred to as the **elliptic curve logarithm problem**.
- The **fastest known technique** for taking the **elliptic curve logarithm** is known as the **Pollard rho method**.
- **Table 10.3, from NIST SP 800-57** (Recommendation for Key Management Part 1: General, September 2015), compares various algorithms by showing **comparable key sizes in terms of computational effort for cryptanalysis**.

- Next slide

Security of Elliptic Curve Cryptography

As can be seen, a considerably smaller key size can be used for ECC compared to RSA. Based on this analysis, SP 800-57 recommends that at least through 2030**, acceptable key lengths are RSA - from 3072 to 14,360 bits for RSA and ECC - 256 to 512 bits for ECC.**

Similarly, the **European Union Agency for Network and Information Security (ENISA**) recommends in their 2014 report (Algorithms, Key Size and Parameters report 2014, November 2014) minimum key lengths for future system of **3072 bits and 256 bits for RSA and ECC, respectively.**

Table 10.3 Comparable Key Sizes in Terms of Computational Effort for Cryptanalysis (NIST SP-800-57)

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.

Analysis indicates that for **equal key lengths**, the computational effort required for **ECC and RSA is comparable** [JURI97]. Thus, there is a **computational advantage to using ECC** with a **shorter key length** than a comparably secure RSA.

Thank You

Man-in-the-Middle (MitM) attack on the Diffie-Hellman Key Exchange