Principles of Cryptography

Josh Wilcox (jw14g24@soton.ac.uk)

March 8, 2025

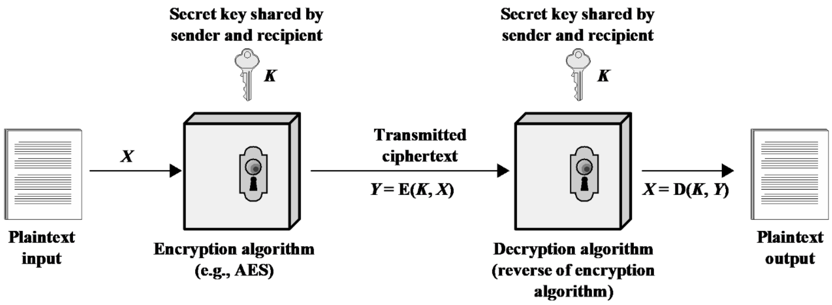
Contents

1 Brief History

- Cryptography is the practice and study of techniques for secure communication in the presence of **adversarial** behaviour
- History:
  - 400BC - Greeks used a Scytale to encode messages
  - Middle Ages - Caesar cipher was used to shift each letter of the alphabet by a fixed number of positions
  - 16th Century - Vigenere cipher - Uses interwoven caesar ciphers
  - 19th Century - Princple that security of cryptography should **depend only on the secrecy of the key** and not the algorithm
  - 20th Century - Enigma machine by the sneaky Germans introduced public-key cryptography

2 Symmetric Encryption

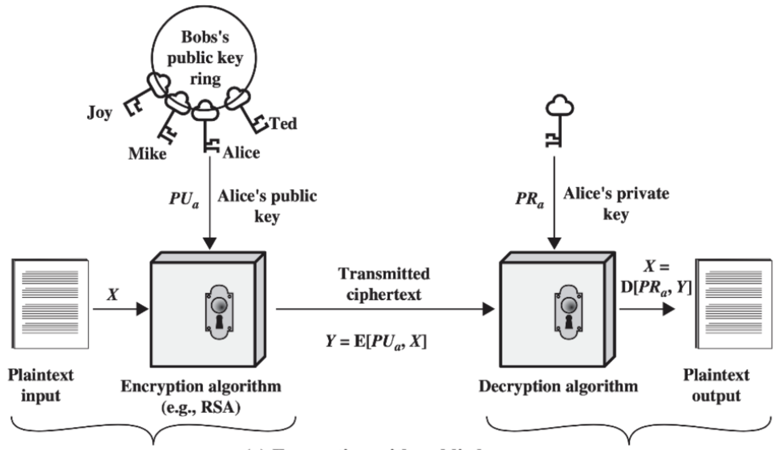
- Same key is used to encrypt and decrypt apeice of information X
- Used in the *Advanced Encryption Standard* (AES)



3 Assymetric Encryption

- Each user has a pair of keys, a **Private Key** and a **Public Key**
- Sender encrypts a piece of information X with the **public key** of the recipient
- Recipient decrypts with its private key

3.1 RSA Encryption



(a) Encryption with public key

3.2 How RSA Works

RSA (Rivest–Shamir–Adleman) is one of the first public-key cryptosystems widely used for secure data transmission. Let's break down how it works step by step:

3.2.1 Key Generation

1. **Choose two distinct prime numbers**: p and q
2. **Compute** $N = p \times q$: This will be part of both the public and private keys
3. **Compute Euler's totient function**: $T = (p - 1) \times (q - 1)$
4. **Choose an integer** e: Such that $1 < e < T$ and e is *coprime* to T (they share no common factors except 1)
5. **Determine** d: Such that $(e \times d) \bmod T = 1$
6. The **public key** is (N, e)
7. The **private key** is (N, d)

3.2.2 Encryption

To encrypt a message M:

$$\text{EncryptedMessage} = M^e \bmod N \tag{1}$$

3.2.3 Decryption

To decrypt the encrypted message C:

$$\text{DecryptedMessage} = C^d \bmod N = (M^e)^d \bmod N = M^{ed} \bmod N = M \tag{2}$$

3.2.4 Why RSA Works

The security of RSA relies on the fact that:

- It's easy to multiply large prime numbers ($p \times q = N$)
- It's extremely difficult to factor N back into p and q when these are large primes
- The relationship $(e \times d) \bmod T = 1$ ensures that $M^{ed} \bmod N = M$

3.2.5 Simple Example

Let's work through a simplified example:

1. Choose $p = 5$ and $q = 11$
2. $N = p \times q = 5 \times 11 = 55$
3. $T = (p - 1) \times (q - 1) = 4 \times 10 = 40$
4. Choose $e = 7$ (coprime with 40)
5. Find d such that $(7 \times d) \bmod 40 = 1$
6. $d = 23$ works because $7 \times 23 = 161$ and $161 \bmod 40 = 1$
7. Public key: $(55, 7)$
8. Private key: $(55, 23)$

To encrypt the message $M = 9$:

$$C = 9^7 \bmod 55 = 4 \tag{3}$$

To decrypt:

$$M = 4^{23} \bmod 55 = 9 \tag{4}$$

This demonstrates how the original message can be recovered using the private key.

4 Digital Signature Encryption

- The **Sender** uses their own **Private Key** to encrypt
- Recipient decrypts with the sender's **public key**

5 Hash Function

- A **One Way Function**
- Maps data of arbitrary size to a bit string of a fixed size
- Not **Encryption** - As it cannot be decrypted

5.1 Applications

- Verifying the integrity of messages and files
  - Hash functions generate a fixed-size hash value from the original data.
  - Any change in the data results in a different hash value, indicating tampering.
- Signature generation and verification
  - Digital signatures use hash functions to create a unique representation of the data.
  - The hash value is encrypted with the sender's private key to form the signature.
  - The recipient decrypts the signature with the sender's public key and compares it to the hash of the received data.
- Password verification
  - Passwords are hashed and stored in a database.
  - During login, the entered password is hashed and compared to the stored hash.
  - This ensures that the actual password is never stored or transmitted.

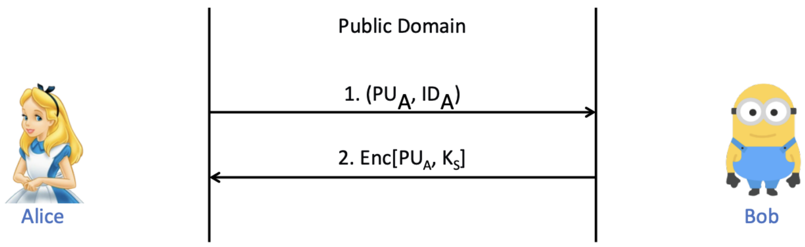
- Proof-of-work
  - Used in blockchain and cryptocurrency mining.
  - Miners must find a hash value that meets certain criteria, which requires significant computational effort.
  - This ensures the integrity and security of the blockchain.

6 Key Distribution

- Key distribution refers to the process of sharing cryptographic keys between parties who wish to communicate securely
- A fundamental challenge in cryptography: "How do we securely share the keys needed for secure communication?"
- The number of keys required differs significantly between symmetric and asymmetric encryption:
  - **Asymmetric encryption**: For n entities, $2n$ keys are needed (each entity has a public-private key pair)
    * Example: For A, B, and C to communicate with D, $4 \times 2 = 8$ keys are needed
  - **Symmetric encryption**: For n entities to communicate with each other, $\frac{n(n-1)}{2}$ keys are needed
    * Example: For A, B, C, and D to communicate with each other, $\frac{4\times3}{2} = 6$ keys are needed
- Symmetric encryption requires a secure method for initially distributing the shared secret keys
- Solutions to the key distribution problem include:
  - Using asymmetric encryption to share symmetric keys
  - Diffie-Hellman key exchange protocol

6.1 Example

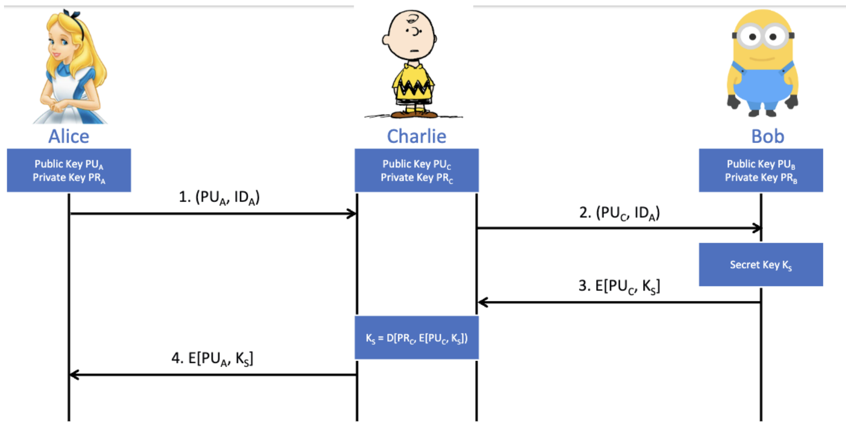
6.1.1 Key Distribution with Public Key Encryption



1. **Participants**: Alice and Bob are the two parties wishing to communicate securely.
2. **Public Key (PU)**: Each participant has a public key (PU) and a private key (PR).
   - Alice has PU_A (public) and PR_A (private).
   - Bob has PU_B (public) and PR_B (private).
3. **Process**:
   - Step 1: Alice sends her public key PU_A along with her identity ID_A to Bob.
   - Step 2: Bob then encrypts his secret key K_S using Alice's public key PU_A and sends it to her. This ensures that only Alice can decrypt it using her private key PR_A.

This method allows Bob to securely share his secret key with Alice, ensuring that only she can read it.

6.1.2 Man-in-the-Middle Attack (MITM)



1. **Participants**: Alice, Bob, and an attacker named Charlie.
2. **Process**:
   - Step 1: Alice sends her public key PU_A and identity ID_A to Bob. However, Charlie intercepts this message.
   - Step 2: Charlie then relays a modified version of the message to Bob. He could send his own public key instead of Alice's.
   - Step 3: Bob receives the message, believing it to be from Alice, and sends his secret key K_S encrypted with the public key he received (which is actually Charlie's).
   - Step 4: Charlie decrypts the message using his private key PR_C to obtain K_S and can now communicate with both Alice and Bob, pretending to be each other.

6.2 Diffie-Hellman Key Exchange Protocol

- Enables two users to securely exchange a key that can be used for symmetric encryption of messages
- Algorithm is limited to the **exchange of secret values**
- Its effectiveness depends on the difficulty of computing discrete logarithms

6.2.1 The Diffie-Hellman Process

The Diffie-Hellman key exchange protocol follows these steps:

1. **Setup public parameters:**
   - Choose a prime number q (in our example, $q = 13$)
   - Choose a primitive root α of q (in our example, $\alpha = 2$)
     - A primitive root α of q ensures that the powers of α generate all the elements of the multiplicative group of integers modulo q.
     - Using a primitive root guarantees that the discrete logarithm problem is hard to solve.
     - It ensures that the generated keys or values are uniformly distributed over the group, preventing patterns that could be exploited by attackers.
   - These parameters are public and known to all participants
2. **Private key generation:**
   - Alice selects a random private key $PR_A < q$ (in our example, $PR_A = 4$)
   - Bob selects a random private key $PR_B < q$ (in our example, $PR_B = 5$)

- These private keys are kept secret

3. **Public key calculation:**
   - Alice calculates her public key: $PU_A = \alpha^{PR_A} \bmod q = 2^4 \bmod 13 = 16 \bmod 13 = 3$
   - Bob calculates his public key: $PU_B = \alpha^{PR_B} \bmod q = 2^5 \bmod 13 = 32 \bmod 13 = 6$
   - These public keys are exchanged over an insecure channel
4. **Shared secret calculation:**
   - Alice computes: $K = (PU_B)^{PR_A} \bmod q = 6^4 \bmod 13 = 1296 \bmod 13 = 9$
   - Bob computes: $K = (PU_A)^{PR_B} \bmod q = 3^5 \bmod 13 = 243 \bmod 13 = 9$
   - Both Alice and Bob now share the same secret key $K = 9$

The security of this protocol relies on the computational difficulty of the discrete logarithm problem. Even if an attacker knows q, α, PU_A, and PU_B, they cannot easily determine PR_A or PR_B, which are needed to calculate the shared key.

6.2.2 Mathematical Proof

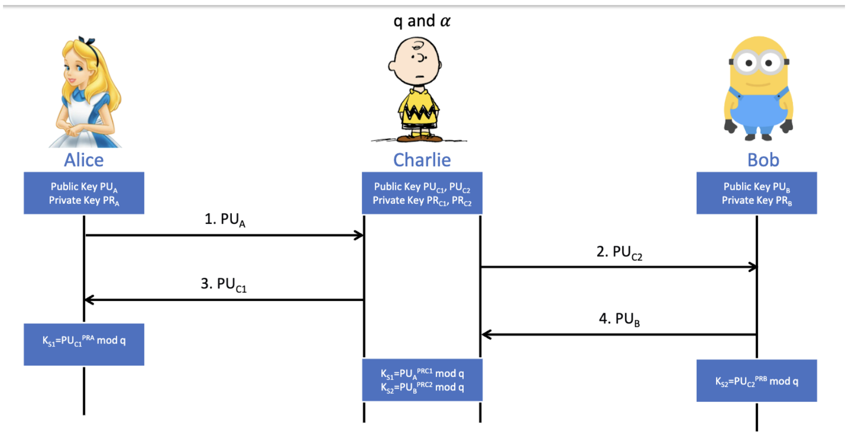
The reason both parties arrive at the same key is:

$$K_{Alice} = (PU_B)^{PR_A} \bmod q = (\alpha^{PR_B})^{PR_A} \bmod q = \alpha^{PR_B \cdot PR_A} \bmod q \tag{5}$$

$$K_{Bob} = (PU_A)^{PR_B} \bmod q = (\alpha^{PR_A})^{PR_B} \bmod q = \alpha^{PR_A \cdot PR_B} \bmod q \tag{6}$$

Since multiplication is commutative, $PR_B \cdot PR_A = PR_A \cdot PR_B$, thus both parties compute the same shared secret.

6.2.3 MITM Attack for Diffie-Hellman



The Diffie-Hellman key exchange protocol, while mathematically sound, is vulnerable to man-in-the-middle (MITM) attacks when implemented without authentication. Here's how such an attack works:

1. **Attack Setup**:
   - Alice and Bob want to establish a shared secret key using Diffie-Hellman
   - Charlie positions himself between Alice and Bob to intercept their communications
   - Public parameters q and α are known to all participants
2. **Attack Process**:
   - Alice generates her private key PR_A and calculates her public key $PU_A = \alpha^{PR_A} \bmod q$
   - Alice attempts to send PU_A to Bob, but Charlie intercepts it

- Charlie generates his own private key PR_{C1} and calculates $PU_{C1} = \alpha^{PR_{C1}} \bmod q$
- Charlie forwards PU_{C1} to Bob (pretending it's from Alice)
- Bob generates his private key PR_B and calculates his public key $PU_B = \alpha^{PR_B} \bmod q$
- Bob sends PU_B to Alice, but Charlie intercepts it
- Charlie generates another private key PR_{C2} and calculates $PU_{C2} = \alpha^{PR_{C2}} \bmod q$
- Charlie forwards PU_{C2} to Alice (pretending it's from Bob)

3. **Key Establishment**:
   - Alice computes her secret key: $K_1 = (PU_{C2})^{PR_A} \bmod q = \alpha^{PR_{C2} \cdot PR_A} \bmod q$
   - Bob computes his secret key: $K_2 = (PU_{C1})^{PR_B} \bmod q = \alpha^{PR_{C1} \cdot PR_B} \bmod q$
   - Charlie computes two keys:
     - $K_1 = (PU_A)^{PR_{C2}} \bmod q = \alpha^{PR_A \cdot PR_{C2}} \bmod q$ (shared with Alice)
     - $K_2 = (PU_B)^{PR_{C1}} \bmod q = \alpha^{PR_B \cdot PR_{C1}} \bmod q$ (shared with Bob)

6.2.4 Attack Outcome

- Alice believes she has established a secure key K_1 with Bob, but has actually established it with Charlie
- Bob believes he has established a secure key K_2 with Alice, but has actually established it with Charlie
- Charlie now has two separate keys:
  - K_1 for decrypting and re-encrypting messages from Alice
  - K_2 for decrypting and re-encrypting messages to Bob
- Charlie can now:
  - Read all messages between Alice and Bob
  - Modify messages without detection
  - Inject new messages that appear to come from either party

6.2.5 Prevention Measures

The primary weakness exploited in this attack is the lack of authentication. Solutions include:

- **Digital signatures**: Having participants sign their Diffie-Hellman public values
- **Public key certificates**: Using a trusted third party to verify participant identities
- **Out-of-band verification**: Confirming key fingerprints through a separate secure channel
- **Station-to-Station protocol**: An extension of Diffie-Hellman that includes mutual authentication

This attack demonstrates why authentication is a critical component of secure communication—the ability to establish a shared secret is not sufficient without confirming the identity of the party with whom that secret is shared.