CSC 374: Applied Cryptography

Course Notes

Year 3, Semester 2 -2025

Department of Computer Science, **Kibabii University**

Instructor: Elvira Khwatenge

Office Hours:

Classroom:

Course Description

This course provides an introduction to the principles and practices of modern cryptography. It covers fundamental concepts and techniques for securing information and systems against adversarial attacks. The course focuses on theoretical foundations and practical cryptography applications to equip students with the knowledge and skills to understand, implement, and analyze cryptographic systems in real-world scenarios. Students will learn about the importance of cryptography in various applications such as secure communication, data protection, and authentication

Assessments:

2 CATs and 1 assignment /1 individual, 1 group, 1 sIt in CAT

Learning Objectives

- Understand the fundamentals of cryptography and its historical development
- Master key cryptographic concepts, including encryption types and digital signatures
- Learn about various cryptographic systems and algorithms (e.g., AES, RSA)
- Explore cryptographic applications in secure communications and data protection
- Comprehend cryptanalysis methods and potential vulnerabilities in cryptographic systems
- Study key management techniques and public key infrastructure
- Examine current trends in cryptography, such as post-quantum and blockchain technologies

Recommended Reading & Required Textbook

Highlighted throughout the document

Introduction to Cryptography

History and evolution of cryptography

Origins date back thousands of years, with early examples in ancient Egypt and Rome. Ancient cryptography (before 1900s) relied on simple substitution and transposition ciphers like the Caesar cipher and Vigenère cipher, focusing mainly on message confidentiality. The World War era (1914-1945) saw major advances with rotor cipher machines like the Enigma, as cryptanalysis became more sophisticated and began employing mathematical and statistical techniques. The modern era (1950s-present) has been shaped by computers enabling more complex ciphers. Modern cryptography emerged in 1970s with computer-based systems and public-key cryptography followed by advanced algorithms like AES and quantum cryptography from the 1990s onward. The focus has shifted towards protecting digital data and communications. Important milestones include the 1976 publication of Diffie-Hellman key exchange, the 1978 invention of the RSA algorithm, and AES becoming the new standard cipher in 2001.

Milestones

- 1917: Vernam cipher (one-time pad)
- 1976: Diffie-Hellman key exchange
- 1978: RSA public-key cryptosystem
- 1991: First PGP release
- 2001: AES becomes official standard

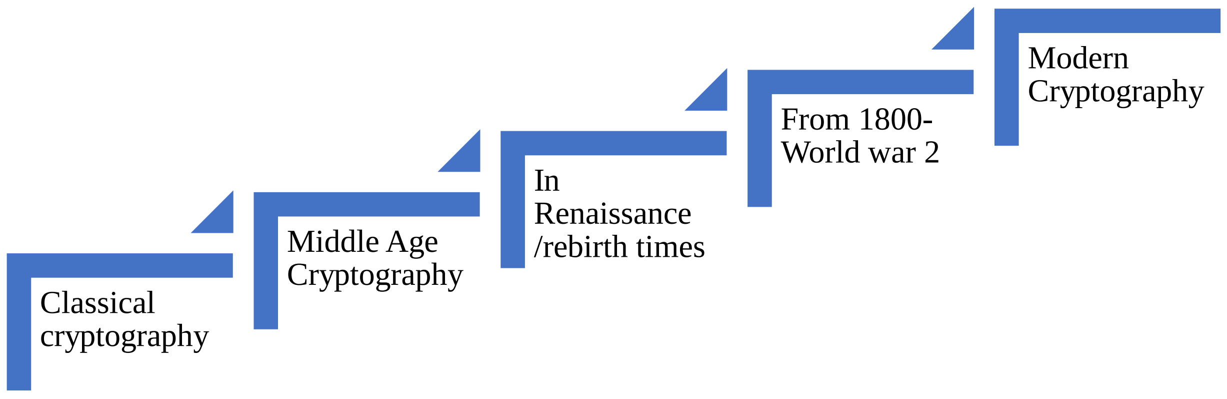
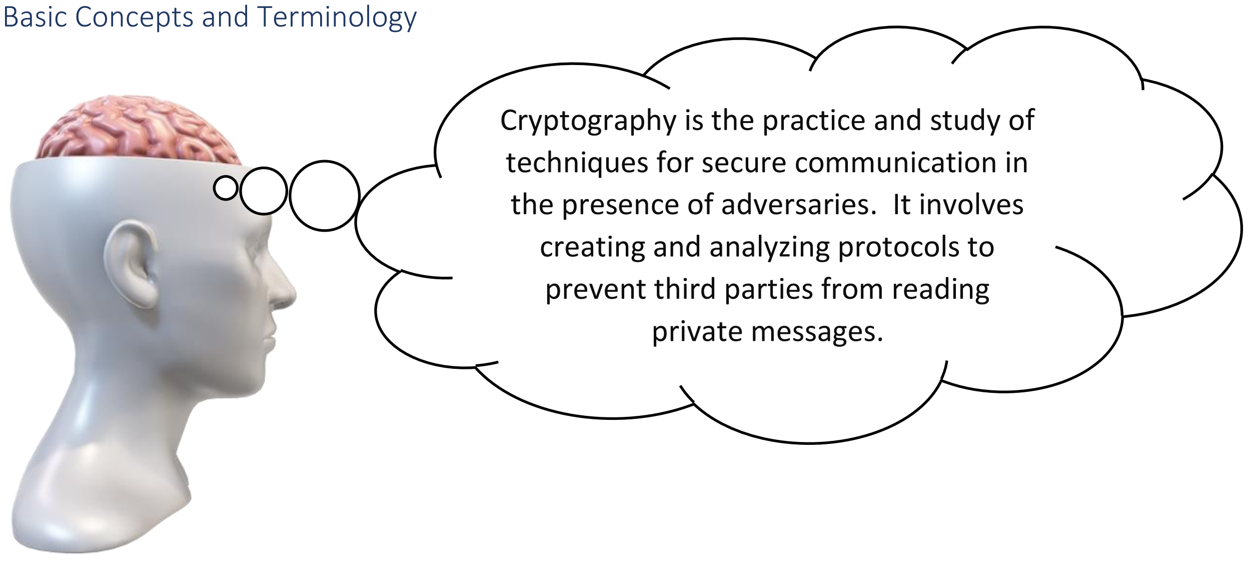


Figure 1 Evolution of cryptography

Basic Concepts and Terminology



Cryptography is the practice and study of techniques for secure communication in the presence of adversaries. It involves creating and analyzing protocols to prevent third parties from reading private messages.

Main concepts and terminology

Let's begin to formalize our scenario involving Alice, Bob, and Eve. Alice has a message m that she wants to send (privately) to Bob. We call m the plaintext. We assume she will somehow transform that plaintext into a value c (called the ciphertext) that she will actually send to Bob. The process of transformingm into c is called encryption, and we will use Enc to refer to the encryption algorithm. When Bob receivesc, he runs a corresponding decryption algorithm Dec to recover the original plaintext m. We assume that the ciphertext may be observed by the eavesdropper Eve, so the (informal) goal is for the ciphertext to be meaningful to Bob but meaningless to Eve.

$$\begin{tikzcd} {} \arrow[r, "m"] & \boxed{\text{Enc}} \arrow[r, "c"] & \boxed{\text{Dec}} \arrow[r, "m"] & {} \end{tikzcd}$$

- Plaintext is the original,readable message
- Ciphertext is the encrypted, unreadable message
- Encryption is the process of converting plaintext to ciphertext
- Decryption is the process of converting ciphertext back to plaintext
- Key is the secret parameter used in cipher operations

Others

1. Symmetric-key cryptography: Encryption systems that use the same key for encryption and decryption.

2. Public-key cryptography: Systems that use a public key for encryption and a private key for decryption.
3. Eavesdropper[1]
4. Digital signature: A mathematical scheme for verifying the authenticity of digital messages or documents.
5. Hash function: A function that maps data of arbitrary size to a fixed-size output, used for integrity checking.
6. Cryptanalysis: The study of methods for obtaining the meaning of encrypted information without access to the key. Or cryptanalysis is the science of recovering the plaintext of a message without access to the key. Successful cryptanalysis may recover the plaintext or the key. It also may find weaknesses in a cryptosystem that eventually lead to the previous results.
7. Man-in-the-middle attack: An attack where the attacker secretly relays and possibly alters the communications between two parties.
8. Brute-force attack: A cryptanalytic attack that tries every possible key until the correct one is found.
9. Key exchange: Protocols that allow two parties to securely share a secret key over an insecure channel.
10. Steganography: The practice of concealing a message within another message or physical object.

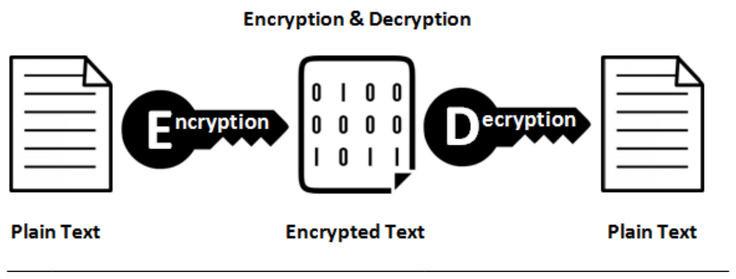


Figure 2 Encryption Decryption process

[1] "Eavesdropper" refers to someone who secretly listens in on a conversation between others. The term originated as a reference to someone who literally hung from the eaves of a building in order to hear conversations happening inside

Types of cryptographic systems

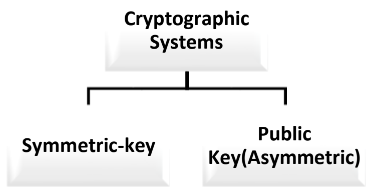


Figure 3 Types of cryptographic systems

Symmetric key systems **use the same key** for encryption and decryption, while *public key systems* **use different but mathematically related keys**. Symmetric is faster but requires secure key exchange, while public key solves the key distribution problem but is slower. Read more here https://cacr.uwaterloo.ca/hac/about/chap2.pdf .

Goals of Cryptography

Primary goals

1. **Confidentiality:** Keeping information secret from unauthorized parties
2. **Integrity:** Ensuring information has not been altered
3. **Authentication:** Verifying the identity of communicating parties
4. **Non-repudiation:** Preventing denial of previous actions or commitments

Secondary goals

- *Access control:* Restricting access to resources
- *Availability:* Ensuring information is accessible when needed
- *Privacy:* Protecting personal information

Communication participants

_ An *entity* or *party* is someone or something which sends, receives, or manipulates information. entity may be a person, a computer terminal, etc.
_ A *sender* is an entity in a two-party communication which is the legitimate transmitter of information.
_ A *receiver* is an entity in a two-party communication which is the intended recipient of information.

_ An *adversary* is an entity in a two-party communication which is neither the sender nor receiver, and which tries to defeat the information security service being provided between the sender and receiver. Various other names are synonymous with adversary such as enemy, attacker, opponent, tapper, eavesdropper, intruder, and interloper. An adversary will often attempt to play the role of either the legitimate sender or the legitimate receiver.

Channels

_ A *channel* is a means of conveying information from one entity to another.

_ A *physically secure channel* or *secure channel* is not physically accessible to the adversary.

_ An *unsecured channel* is one from which parties other than those for which the information is intended can reorder, delete, insert, or read.

_ A *secured channel* is one from which an adversary cannot reorder, delete, insert, or read.

Symmetric Key Cryptography

Symmetric key cryptography (SKC) is the oldest and most intuitive form of cryptography. With SKC, confidential information is secured through symmetric key encryption (SKE), that is, by using a *single secret key* for both encryption and decryption.

SKC involves:

- An encryption function that converts a given plain text instance to ciphertext while utilizing a secret key
- A decryption function that inverts the operation by converting the ciphertext back to plain text using the same secret key

In cryptography, a symmetric key is one that is used both to encrypt and decrypt information. This means that to decrypt information, one must have the same key that was used to encrypt it. The keys, in practice, represent a shared secret between two or more parties that can be used to maintain a private information link. This requirement that both parties have access to the secret key is one of the main drawbacks of symmetric key encryption, in comparison to public-key encryption.

Plain text can mean any kind of unencrypted data such as natural language text or binary code whose information content is in principle directly accessible, while ciphertext refers to encrypted data whose information content is intended to be inaccessible before decryption.

An algorithm that describes the encryption and decryption operations using a shared secret key is also called a symmetric cipher.

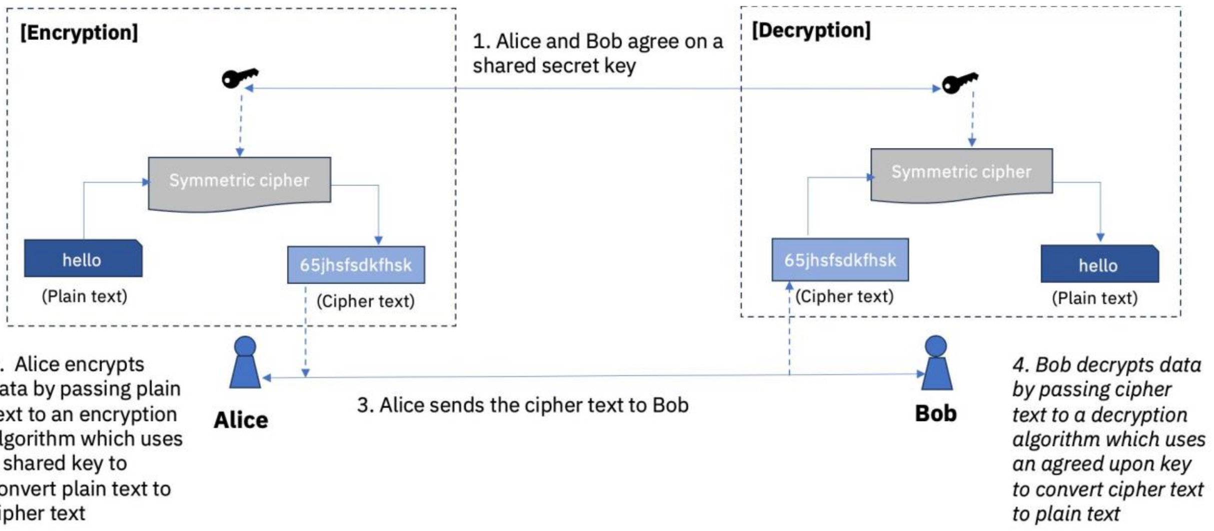

[Encryption]
1. Alice and Bob agree on a shared secret key
[Decryption]
Symmetric cipher
hello
(Plain text)
65jhsfsdkfhsk
(Cipher text)
Symmetric cipher
65jhsfsdkfhsk
(Cipher text)
hello
(Plain text)
2. Alice encrypts data by passing plain text to an encryption algorithm which uses a shared key to convert plain text to cipher text
Alice
3. Alice sends the cipher text to Bob
Bob
4. Bob decrypts data by passing cipher text to a decryption algorithm which uses an agreed upon key to convert cipher text to plain text

Figure 4: Symmetric key encryption of a given plain text to ciphertext and decryption back to plain text using the same key.

Properties of symmetric key cryptosystems

A symmetric key cryptosystem should ensure the following properties to secure messages statically stored data and/or communications over some transmission channel:

- **Confidentiality:** Refers to the property that the information content of encrypted messages is protected from unauthorized access.
- **Integrity:** Refers to the property that any tampering of encrypted messages during storage or transmission can be detected.
- **Authenticity:** Refers to the property that the receiver of a message can verify the identity of the sender and detect impersonation by an unauthorized party.

Furthermore, these properties should be realized in a setting where the algorithms or ciphers used for encryption and decryption may be public and

where access to the information content of encrypted messages is controlled exclusively through access to the secret key.
Implementing a secure symmetric key cryptosystem therefore involves two main tasks:

1. Employing a robust symmetric key encryption algorithm resistant to cryptographic attacks.
2. Ensuring confidentiality in the distribution and management of secret keys.

Illustration of symmetric key encryption using python

We show a simple example of encrypt and decrypt operations using the classical Caesar shift cipher and the modern Advanced Encryption System (AES), which has been the standard for symmetric key encryption since 2001. First, we set up some Python libraries that provide the needed symmetric key encryption ciphers, and then define the plain text we wish to encrypt.

Caesar shift cipher:

If you have a message you want to transmit securely, you can encrypt it (translate it into a secret code). One of the simplest ways to do this is with a shift cipher. Famously, Julius Caesar used this type of cipher when sending messages to his military commanders. A shift cipher involves replacing each letter in the message by a letter that is some fixed number of positions further along in the alphabet. We'll call this number the encryption key. It is just the length of the shift we are using.
The Caesar cipher shifts all the letters in a piece of text by a certain number of places. The key for this cipher is a letter which represents the number of place for the shift. So, for example, a key D means "shift 3 places" and a key M means "shift 12 places". Note that a key A means "do not shift" and a key Z can either mean "shift 25 places" or "shift one place backwards". For example, the word "CAESAR" with a shift P becomes "RPTHPG"
Caesar shift encryption involves defining

- An alphabet of possible characters to encode

- A *shift value* which can be between 0 (unencrypted) and the length of the alphabet. We consider this the *key*.

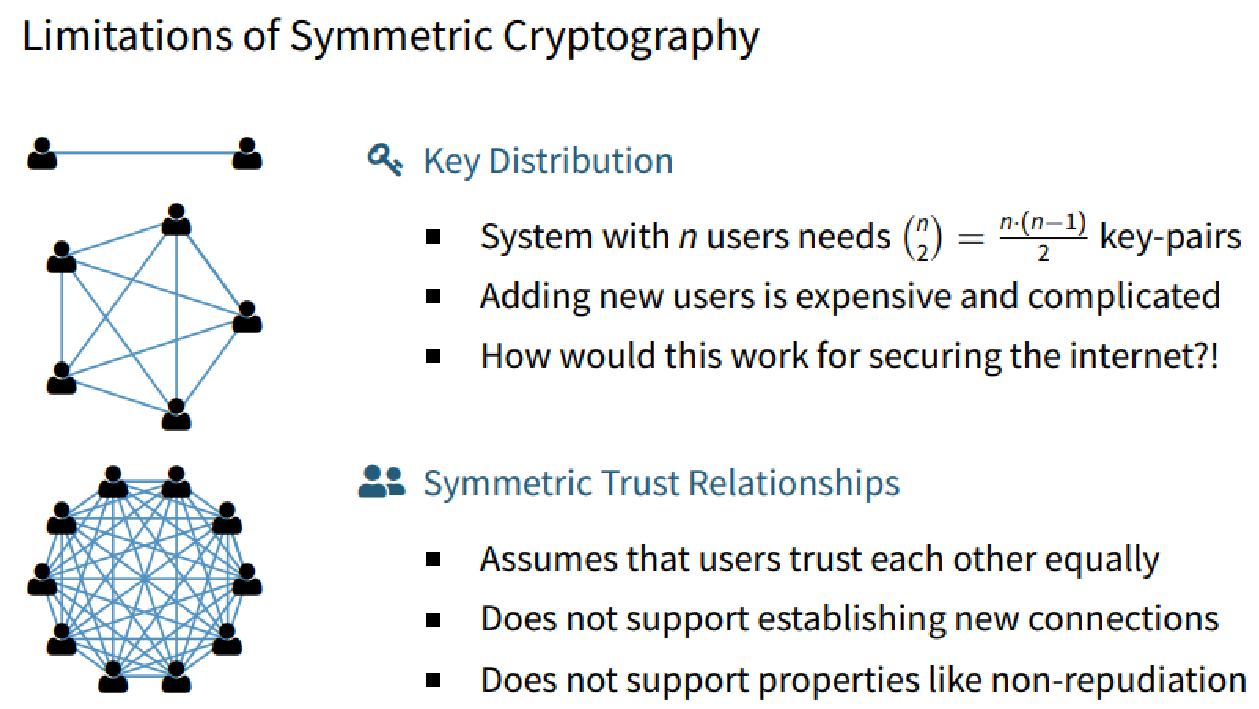
It is known as a *monoalphabetic substitution cipher* since each letter of the plain text is substituted with another in the ciphertext. In this example we will use lowercase letters of the alphabet.

Lets use our jupyter notebooks

Applications of symmetric key cryptography

Application	Description	Examples
Data Protection	Protects sensitive data at rest or in transit.	Full-disk encryption, SSL/TLS for secure web browsing.
Secure Communication	Ensures confidentiality and integrity of messages.	Wi-Fi networks (WPA2), Virtual Private Networks (VPNs).
Financial Transactions	Secures financial data during transactions.	Banking apps, card payments using AES encryption.
Messaging Apps	Provides end-to-end encryption for secure messaging.	iMessage, WhatsApp (combines with asymmetric methods).
File Storage and Compression Tools	Adds security to compressed files or stored data.	ZIP file encryption, encrypted cloud storage services like Dropbox when using symmetric keys internally for efficiency

symmetric cryptosystems have the following problems:
— ***Keys must be distributed in secret***. They are as valuable as all the messages they encrypt, since knowledge of the key gives knowledge of all the messages. For encryption systems that span the world, this can be a daunting task. Often couriers' hand-carry keys to their destinations.
— ***If a key is compromised (stolen, guessed, extorted, bribed, etc.),*** then Eve can decrypt all message traffic encrypted with that key. She can also pretend to be one of the parties and produce false messages to fool the other party.
— ***Assuming a separate key is used for each pair of users in a network***, the total number of keys increases rapidly as the number of users increases. A network of *n* users requires *n*(*n* - 1)/2 keys. For example, 10 users require 45 different keys to talk with one another and 100 users require 4950 keys. This problem can be minimized by keeping the number of users small, but that is not always possible.



Limitations of Symmetric Cryptography

Key Distribution

- System with *n* users needs $\binom{n}{2} = \frac{n\cdot(n-1)}{2}$ key-pairs
- Adding new users is expensive and complicated
- How would this work for securing the internet?!

Symmetric Trust Relationships

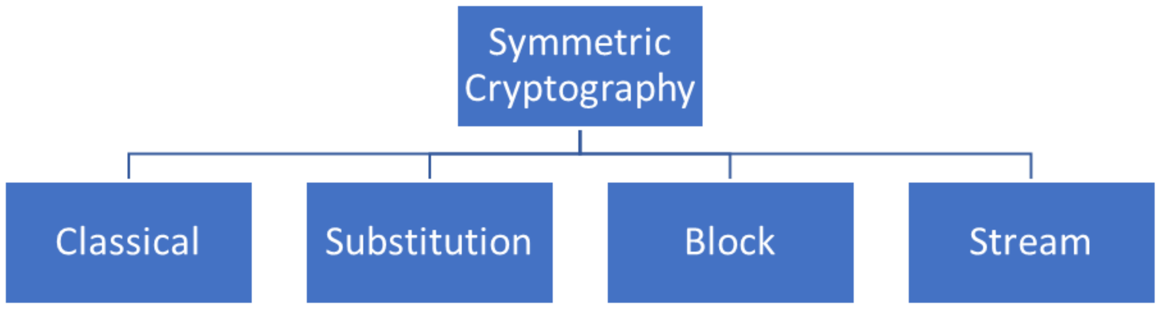
- Assumes that users trust each other equally
- Does not support establishing new connections
- Does not support properties like non-repudiation

The Characteristics of symmetric key cryptography include:

- ❖ Use the same key for encryption and decryption
- ❖ Generally faster and more efficient than asymmetric encryption

- ❖ Suitable for encrypting large amounts of data
- ❖ Require secure key exchange mechanisms
- ❖ Typically use keys of 128, 192, or 256 bits

There are four types of symmetric cryptography: classical. Substitution, block and stream ciphers.



Classical ciphers

- Early encryption methods predating modern computer-based cryptography.
- **Types**:
- **Substitution Ciphers**: Replace plaintext letters or symbols with others.
    - *Example*: Caesar cipher (shifts letters by a fixed number).
- **Transposition Ciphers**: Rearrange the order of letters without changing them.
    - *Examples*: Columnar transposition, Rail fence cipher.

- **Note**: Classical ciphers are now considered insecure due to vulnerabilities to frequency analysis and other cryptanalytic techniques but are valuable for illustrating basic encryption concepts.

Substitution ciphers

Substitution ciphers are encryption methods where plaintext units (like single letters or groups of letters) are replaced with ciphertext according to a regular system. The receiver deciphers the text by performing an inverse substitution.

Types of Substitution Ciphers

rearranges the position of the plain text's characters. In transposition Cipher Technique, The position of the character is changed but character's identity is not changed.

- **Simple Substitution**: Replaces each letter consistently, such as 'A' with 'X', 'B' with 'Y', etc.
- **Polygraphic**: Operates on larger groups of letters.
- **Monoalphabetic**: Uses fixed substitution throughout the message.
- **Polyalphabetic**: Uses multiple substitutions at different times. A well-known example is the Vigenère cipher.
- **Homophonic**: Replaces each letter with multiple possible symbols to flatten frequency distribution.

Vulnerabilities and Applications

Substitution ciphers are vulnerable to frequency analysis attacks because they preserve letter frequencies unless techniques like polyalphabetic substitution are used. Despite being considered weak by modern standards, they have historically been important and form part of more complex encryption methods like DES and AES block ciphers.

Block ciphers

Block ciphers encrypt **fixed-size** (typically 64 or 128 bits) blocks of data, Block ciphers are more common for general-purpose encryption. Further reading here https://mrajacse.wordpress.com/wp-content/uploads/2012/01/applied-cryptography-2nd-ed-b-schneier.pdf .

Examples include:

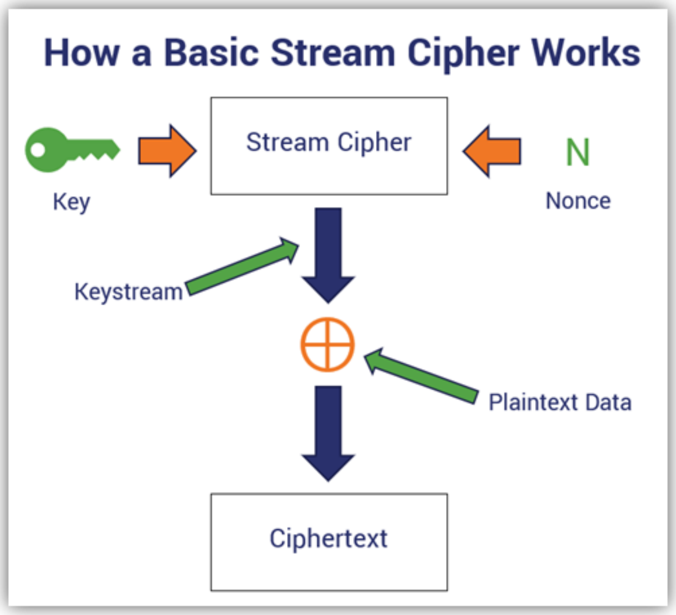
- ✓ **Advanced Encryption Standard (AES).** It’s the gold standard for symmetric encryption since its adoption by the U.S. government in 2001. This block cipher operates on 128-bit blocks and supports key sizes of 128, 192, or 256 bits, offering a balance of security and efficiency across various platforms. AES's widespread adoption is due to its security and versatility, making it the go-to choose for many applications today from Wi-Fi networks (WPA2/WPA3) to cloud storage services like Google Drive and Dropbox. It's also widely used in VPNs, secure messaging apps like WhatsApp, and government communications. (check the security settings of your phones or when you try to put wifi passwords or the famous whatsapp notification” this message is encrypted”
- ✓ **Data Encryption Standard (DES) and Triple DES (3DES)** developed in the 1970s, was once the primary encryption standard but has since been phased out due to its vulnerability to modern attacks. Its 64-bit block size and 56-bit key are no longer considered secure. To address these shortcomings, Triple DES (3DES) was introduced, applying the DES algorithm three times with different keys. While 3DES offers improved security over its predecessor, it is slower than modern alternatives. They can still be found in legacy systems in the financial sector. 3DES is used in

electronic payment systems and as a fallback in some TLS implementations, though it's being phased out.

- ✓ **Blowfish** created by Bruce Schneier in 1993, was designed as a fast and secure alternative to DES. With its 64-bit block size and variable key length ranging from 32 to 448 bits, Blowfish offers flexibility and security for various applications. Its efficiency and open-source nature have contributed to its continued use in many systems.
- ✓ **Twofish** the successor to Blowfish, was a finalist in the AES selection process. It operates on 128-bit blocks and supports key sizes up to 256 bits. While Twofish is considered highly secure, it has not gained as much traction as AES in widespread adoption.
- ✓ **Serpent** another AES finalist, was designed with a focus on security over speed. It uses a 128-bit block size and supports key sizes of 128, 192, or 256 bits. Although Serpent is regarded as very secure, its slower performance in software implementations compared to AES has limited its adoption.

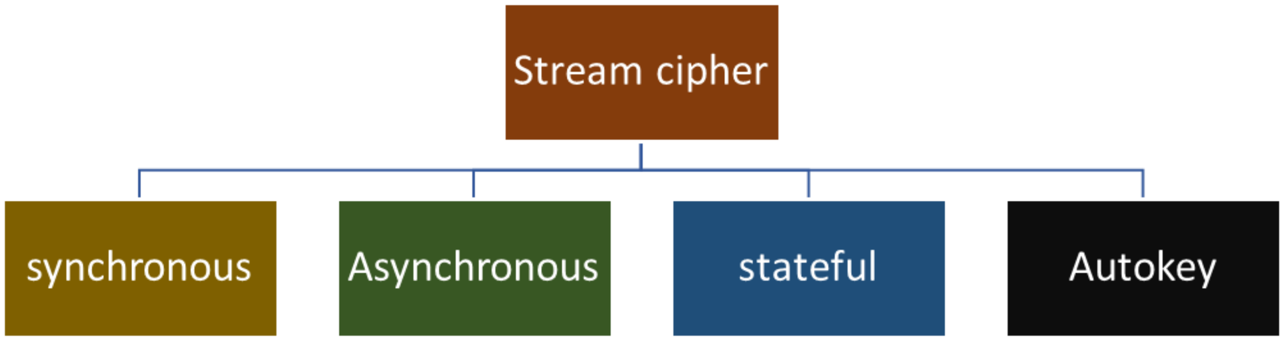
Stream ciphers

Stream ciphers are a type of encryption algorithm that encrypts data one character at a time using binary digits. Unlike block ciphers, which encrypt groups of characters simultaneously, stream ciphers process each symbol individually. This makes them particularly useful for continuous data streams and situations where buffering is limited or transmission errors are common.



Stream ciphers offer advantages such as speed in hardware implementation and minimal error propagation, making them suitable for applications like telecommunications where real-time processing is crucial. Examples of popular stream ciphers include ChaCha20 and Salsa20, although some older ones like RC4 are no longer considered secure due to vulnerabilities discovered over time.

Stream ciphers can be categorized into several types based on how they generate and synchronize their keystreams. The primary classification includes:



1. **Synchronous Stream Ciphers**:

   - These ciphers generate a keystream independently of the plaintext or ciphertext.
   - Both the sender and receiver must be synchronized to decrypt correctly.

- If synchronization is lost due to added or removed digits, it can be restored by trying different offsets.
- Examples include binary additive stream ciphers using XOR operations12.

2. **Self-Synchronizing (Asynchronous) Stream Ciphers**:
    - These update their internal state based on previous ciphertext digits.
    - They automatically resynchronize after receiving a fixed number of ciphertext digits, making them more robust against dropped or added data12.
3. **Stateful Stream Ciphers**:
    - This category includes ciphers like RC4 and Salsa20, where the internal state changes during keystream generation5.
4. **Autokey Ciphers (or Autoclave Ciphers)**:
    - A type of self-synchronizing cipher that incorporates plaintext into the key generation process1.

The following important design considerations for a stream cipher:

- The encryption sequence should have a large period. The longer the period of repeat the more difficult it will be to do cryptanalysis. A pseudorandom number generator uses a function that produces a deterministic stream of bits that eventually repeat. The longer the period of repeat, the more difficult it will be to do cryptanalysis.
- The keystream should approximate the properties of a true random number stream. For example, there should be an approximately equal number of 1s and 0s.
- the output of the pseudorandom number generator is conditioned on the value of the input key. To guard against brute-force attacks, the key needs to be sufficiently long.

Stream Cipher Usage

Stream ciphers are often used for their speed and simplicity of implementation in hardware, and in applications where plaintext comes in quantities of unknowable length like secure wireless connection. Another advantage of stream ciphers in military cryptography is that the cipher stream can be generated in a separate box that is subject to strict security measures

Table comparing substitution and transposition techniques:

Feature	**Substitution Technique**	**Transposition Technique**
Action	Replaces plaintext characters with other characters, numbers, or symbols.	Rearranges the position of plaintext characters.
Forms	Mono-alphabetic and poly-alphabetic ciphers.	Keyed and keyless transposition ciphers.
Change	Changes character identity but keeps its position unchanged.	Changes character position but keeps its identity unchanged.
Examples	Caesar Cipher, Playfair Cipher.	Rail Fence Cipher, Columnar Transposition Cipher.

Security principles

Symmetric cyphers adhere to two important security principles:

Kerckhoff's principle and Shannon's Maxim.

These principles emphasize that cryptographic security should depend on the secrecy of keys rather than algorithms.

Kerckhoffs' principle

The security of a crypto-system should depend only on the secrecy of the key and not on the secrecy of the algorithm If the algorithm is also kept secret then such ciphers are known as classified ciphers.

Classified ciphers are used in defense and military organizations, the usage and practice of which are not recommended in general.

Rationale behind Kerckhoffs' principle

- It is easy to hide a short key than a long algorithm.
- Algorithm can be easily reverse engineered.
- It is easy to replace a compromised key than a compromised algorithm, if required.

- If many parties communicate with different algorithms, then the complexity of maintenance matters.
- If the algorithm is public, it is available for analysis by experts worldwide, and if it survives we gain high confidence on the algorithm.
- A flaw in the algorithm can be fixed easily, if it is in public domain.
- Keeping algorithm open helps in standardization.

How to define security

- An obvious notion from the Kerckhoffs' principle follows that, the key should not be leaked from the cipher text. But it is not sufficient, because for known plain text attack model this notion might fail.
- Plain text should not be leaked from the cipher text. This may also be insufficient. For example, if we consider a situation where 80% of the plain text is leaked and the remaining 20% is secured then the amount of leaked plain text may be containing sufficient data for a successful attack.
- No plain text character should be leaked from the cipher text. This even may be insufficient, as for example in bank transactions even if the exact amount of account balance is not revealed from the cipher text, but leakage of any extra information about the account balance would be undesirable.
- No "meaningful information" about the plain text should be revealed from the cipher text. For example, in bank transactions even if the exact balance of an account related to a transaction is not revealed, but any kind of information, for example like, "account balance is more than one lakh" might come out to be a meaningful information leading to insecurity of the account.

In summary

- The security of a cryptosystem should rely on the secrecy of the key, not on the secrecy of the algorithm.
- The algorithm should be able to fall into enemy hands without compromising security.

- This allows algorithms to be publicly analyzed and vetted by the cryptographic community.
- It enables standardization and interoperability of cryptographic systems.
- Changing keys is much easier than changing algorithms if security is compromised.

Shannon's maxim

- "The enemy knows the system."
- This is a rephrasing and generalization of Kerckhoffs' principle.
- Assume the adversary knows all details of your cryptosystem except the key.
- Security should not rely on keeping the algorithm secret.
- Cryptographic systems should be designed to be secure even if everything except the key is public knowledge.

Applications

❖ Secure communication protocols (e.g., TLS/SSL)
❖ File and disk encryption
❖ Virtual Private Networks (VPNs)
❖ Secure storage of passwords (when combined with key derivation functions)

Advantages of symmetric-key cryptography

1. SKC is fast and efficient, making it suitable for large data volumes or real-time communications. It scales well with data size and has low computational overhead.
2. SKC handles multiple users and large amounts of data effectively due to its low resource requirements.
3. SKC protocols are easier to implement compared to asymmetric methods, which makes them more accessible for developers.
4. SKC can be used as a building block for other cryptographic tools like pseudorandom number generators, hash functions, and digital signatures.
5. Keys are relatively short compared to asymmetric cryptography, which simplifies key management in some respects.

Challenges and limitations of symmetric key cryptography

1. Key Distribution and Management: SKC requires both parties to share the same secret key, which must be kept confidential. Secure distribution is a significant challenge.
2. Lack of Non-Repudiation: Since the same key is used for encryption and decryption, it's impossible to prove who sent a message.
3. Key Management in Networks: Managing multiple keys in large networks is complex and often requires an unconditionally trusted third-party authority (TTP).
4. Frequent Key Changes: Best practices dictate changing keys frequently, ideally for each session.
5. Digital Signatures Limitations: Digital signatures based on SKC typically require large keys or a TTP for verification purposes.

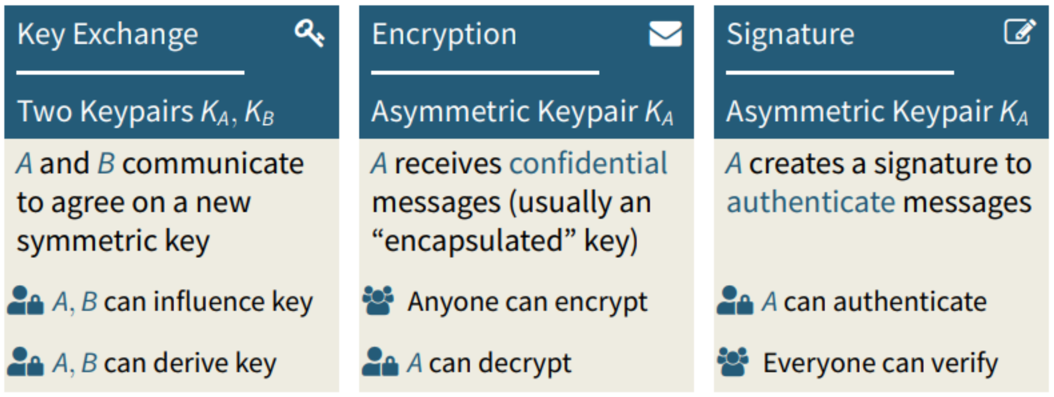
Asymmetric Cryptography

Remember symmetric key cryptography is very fast and efficient for protecting information, but it has a few limitations:

1. As the number of parties wishing to exchange secure information increases, the numbers of keys required grows combinatorially. It provides no mechanism to securely distribute these keys between senders and receivers.
2. There is no provision for *non-repudiation*. Any party is able to decrypt, or encrypt, messages with no way to guarantee a message was received or where it originated

The solution to both these problems is provided by *asymmetric key cryptography* (AKC), also known as *public key cryptography* (PKC), which therefore forms a cornerstone of modern digital security.

Asymmetric key cryptography (AKC) involves the use of a pair of keys – one public, one private. The public and private keys are cryptographically linked and typically generated at the same time as a *key pair* using a specialized mathematical algorithm. The public key, as suggested by its name, is then meant to be freely distributed, while the private key is kept secret by the party generating the key pair. Security of communications employing asymmetric key pairs is assured as long as the private key remains confidential.

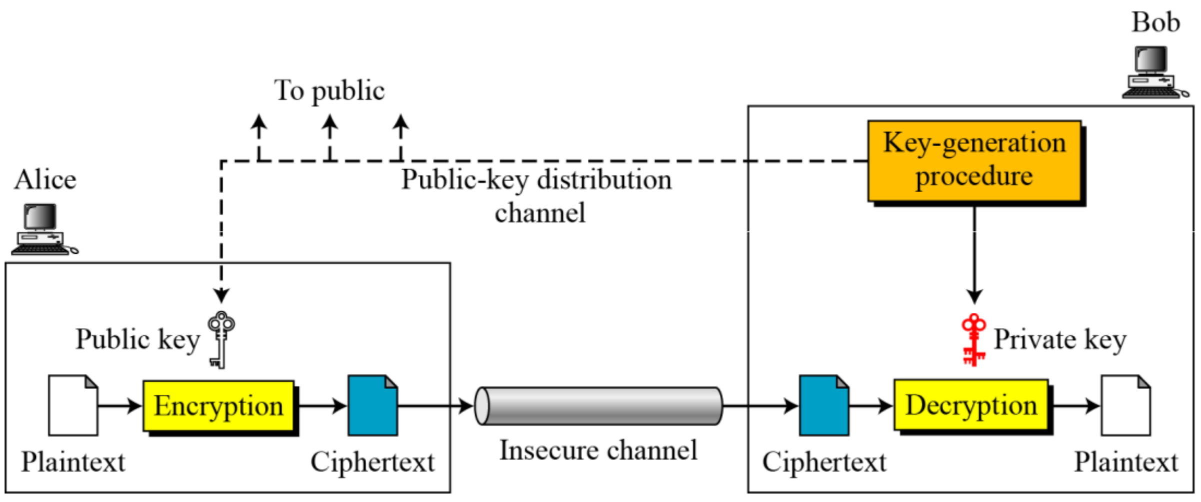


Key Exchange	Encryption	Signature
Two Keypairs K_A, K_B	Asymmetric Keypair K_A	Asymmetric Keypair K_A
A and *B* communicate to agree on a new symmetric key	*A* receives confidential messages (usually an “encapsulated” key)	*A* creates a signature to authenticate messages
A, *B* can influence key	Anyone can encrypt	*A* can authenticate
A, *B* can derive key	*A* can decrypt	Everyone can verify

Think of asymmetric algorithm as a safe. The key is the combination. Someone with the combination can open the safe, put a document inside, and close it again. Someone else with the combination can open the safe and take the document out. Anyone without the combination is forced to learn safecracking. Asymmetric or public-key cryptography uses pairs of related keys - a public key that can be freely shared and a private key that is kept secret. This solves the exchange problem inherent in symmetric cryptography.

Remember the Exchange Problem

- In symmetric cryptography, securely exchanging the shared secret key is challenging, especially over insecure channels
- Public-key cryptography solves this by allowing keys to be exchanged over insecure channels



$$\boxed{C = f(K_{public}, P) \qquad P = g(K_{private}, C)}$$

Figure 5 A general idea of asymmetric cryptography

Scenario

This is how Alice can send a message to Bob using public-key cryptography:

(1) Alice and Bob agree on a public-key cryptosystem.

(2) Bob sends Alice his public key.

(3) Alice encrypts her message using Bob's public key and sends it to Bob.

(4) Bob decrypts Alice's message using his private key.

Notice how public-key cryptography solves the key-management problem with symmetric cryptosystems. Before, Alice and Bob had to agree on a key in secret. Alice could choose one at random, but she still had to get it to Bob. She could hand it to him sometime beforehand, but that requires foresight. She could send it to him by secure courier, but that takes time. Public-key cryptography makes it easy. With no prior arrangements, Alice can send a secure message to Bob. Eve, listening in on the entire exchange, has Bob's public key and a message encrypted in that key, but cannot recover either Bob's private key or the message.

More commonly, a network of users agrees on a public-key cryptosystem. Every user has his or her own public key and private key, and the public keys are all published in a database somewhere. Now the protocol is even easier:
(1) Alice gets Bob's public key from the database.
(2) Alice encrypts her message using Bob's public key and sends it to Bob.
(3) Bob then decrypts Alice's message using his private key.
In the first protocol, Bob had to send Alice his public key before she could send him a message. The second protocol is more like traditional mail. Bob is not involved in the protocol until he wants to read his message.

Properties

These algorithms are based on computationally intensive problems such as finding the prime factors of large numbers.

- Longer the length of the key pair, the more time it takes to crack the private key
- Keys used in today's internet will take millions of years to crack using today's technologies

Public key cryptosystems are slow, really slow!

- three orders of magnitude (1000 times) slower than AES
- mainly used as key exchange tool

Scientists are supposed to be real "smart" and love to solve difficult problems

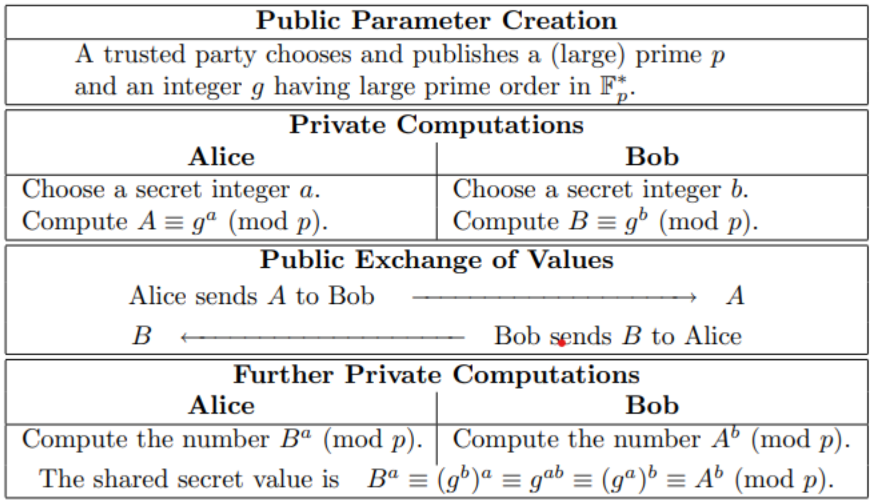
- but even they hope to never solve factoring
- if you can find a quick solution,
- fame, dollars and danger lurk!

Diffie-Hellman Key Exchange

The key exchange question was one of the first problems addressed by a cryptographic protocol. This was prior to the invention of public key cryptography. The Diffie-Hellman key agreement protocol (1976) was the first practical method for establishing a shared secret over an unsecured communication channel. The point is to agree on a key that two parties can use for a symmetric encryption, in such a way that an eavesdropper cannot obtain the key.

- Allows two parties to generate a shared secret key over an insecure channel
- Used in protocols like HTTPS/TLS to establish session keys
- Real-world applications: VPNs, encrypted messaging apps, secure email

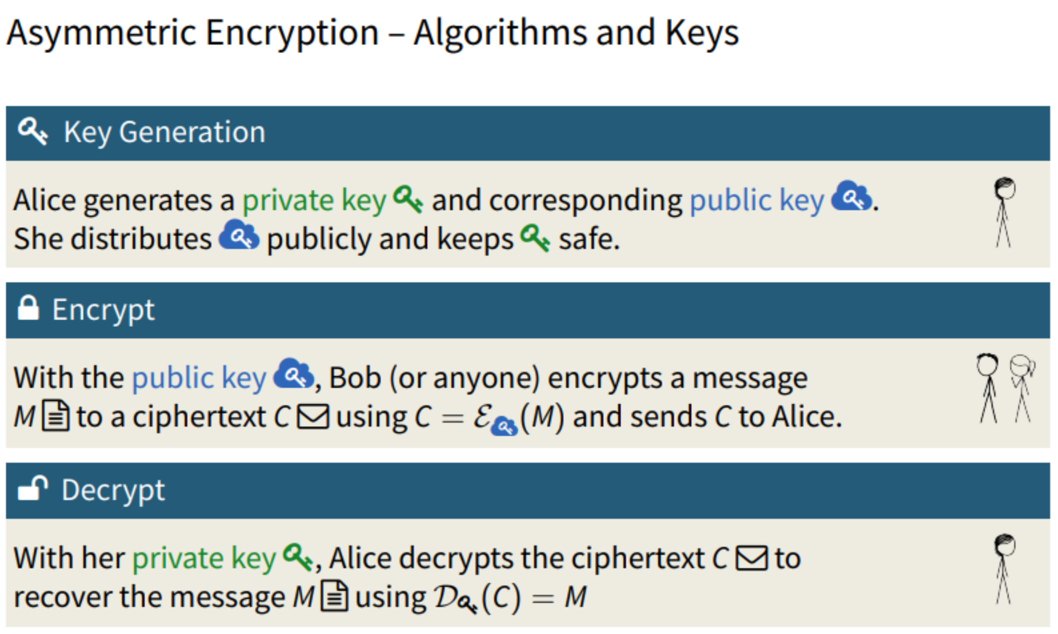
The Diffie–Hellman key exchange algorithm is summarized in Table 1 below



Public Parameter Creation	
A trusted party chooses and publishes a (large) prime p and an integer g having large prime order in $\mathbb{F}_p^*$.	
Private Computations	
Alice	**Bob**
Choose a secret integer a. Compute $A \equiv g^a \pmod{p}$.	Choose a secret integer b. Compute $B \equiv g^b \pmod{p}$.
Public Exchange of Values	
Alice sends A to Bob $\longrightarrow$	A
B	$\longleftarrow$ Bob sends B to Alice
Further Private Computations	
Alice	**Bob**
Compute the number $B^a \pmod{p}$.	Compute the number $A^b \pmod{p}$.
The shared secret value is $B^a \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv A^b \pmod{p}$.	

Asymmetric Encryption

Asymmetric cryptography makes extensive use of "one-way functions": easy to compute, hard to invert. A "trapdoor one-way function" is a one-way function which can be inverted with an additional piece of information, the trapdoor information.



Asymmetric Encryption – Algorithms and Keys

Key Generation

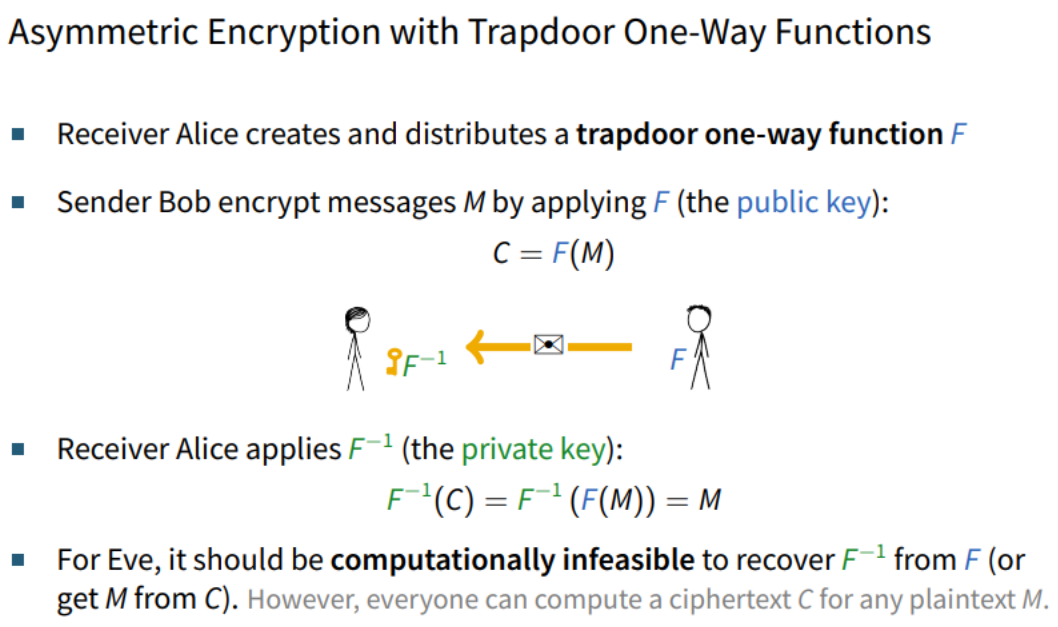
Alice generates a private key and corresponding public key. She distributes publicly and keeps safe.

Encrypt

With the public key, Bob (or anyone) encrypts a message M to a ciphertext C using $C = \mathcal{E}(M)$ and sends C to Alice.

Decrypt

With her private key, Alice decrypts the ciphertext C to recover the message M using $\mathcal{D}(C) = M$



Asymmetric Encryption with Trapdoor One-Way Functions

- Receiver Alice creates and distributes a **trapdoor one-way function** F
- Sender Bob encrypt messages M by applying F (the public key):

$$C = F(M)$$

- Receiver Alice applies F^{-1} (the private key):

$$F^{-1}(C) = F^{-1}(F(M)) = M$$

- For Eve, it should be **computationally infeasible** to recover F^{-1} from F (or get M from C). However, everyone can compute a ciphertext C for any plaintext M.

RSA Encryption and Digital Signatures

RSA can be used for key exchange, digital signatures and the encryption of small blocks of data.

- RSA is primarily used to encrypt the session key used for secret key encryption or the message hash value (digital signature).
- RSA's mathematical hardness comes from the ease in calculating large numbers and the difficulty in finding the prime factors of those large numbers.
- To create an RSA public/private key pair, here are the basic steps:

1- Choose two prime numbers, p and q such that $p \neq q$.
2- Calculate the modulus, $n = p \times q$.
3- Calcuate $\phi(n) = (p-1) \times (q-1)$.
4- Select integer e such that gcd $(\phi(n), e) = 1$ and $1 < e < \phi(n)$. (* gcd is greater common divisor)
5- Calculate an integer d from the quotient $de \equiv 1 \pmod{\phi(n)} \Rightarrow de = 1 + k\,\phi(n) \Rightarrow$
$d = (1 + k\,\phi(n)) / e$

- To encrypt a message, M, with the public key (e, n), create the ciphertext, C, using the equation:
  $C = M^e \bmod n$
- The receiver then decrypts the ciphertext with the private key (d, n) using the equation:
  $M = C^d \bmod n$

The RSA Example

1. Select two prime numbers, p = 17 and q = 11.
2. Calculate $n = p \times q = 17 \times 11 = 187$.
3. Calculate $\phi(n) = (p - 1)(q - 1) = 16 \times 10 = 160$.
4. Select e such that e is relatively prime to $\phi(n) = 160$ and less than $\phi(n)$; we choose e = 7.
5. Determine d such that $de \equiv 1 \pmod{160}$ and $d < 160$. $de = 1 + k\,\phi(n)$
The correct value is d = 23, because $23 \times 7 = 161 = 1 + (1 \times 160)$.
The resulting keys are public key PU = {7, 187} and private key PR = {23, 187}.

Given a plaintext input of M = 88. For encryption, we need to calculate $C = 88^7 \bmod 187$. we can do this as follows.
$88^7 \bmod 187 = [(88^4 \bmod 187) * (88^2 \bmod 187) * (88^1 \bmod 187)] \bmod 187$
$88^1 \bmod 187 = 88$
$88^2 \bmod 187 = 7744 \bmod 187 = 77$
$88^4 \bmod 187 = 59{,}969{,}536 \bmod 187 = 132$
$88^7 \bmod 187 = (88 * 77 * 132) \bmod 187 = 894{,}432 \bmod 187 = 11$

For Decryption

For decryption, we calculate $M = 11^{23} \bmod 187$:
$11^{23} \bmod 187 = [(11^1 \bmod 187) * (11^2 \bmod 187) * (11^4 \bmod 187) * (11^8 \bmod 187) * (11^8 \bmod 187)] \bmod 187$
$11^1 \bmod 187 = 11$
$11^2 \bmod 187 = 121$
$11^4 \bmod 187 = 14{,}641 \bmod 187 = 55$
$11^8 \bmod 187 = 214{,}358{,}881 \bmod 187 = 33$
$11^{23} \bmod 187 = (11 * 121 * 55 * 33 * 33) \bmod 187 = 79{,}720{,}245 \bmod 187 = 88$

In the preceding example shows, we can make use of a property of modular arithmetic:
$[(a \bmod n) * (b \bmod n)] \bmod n = (a * b) \bmod n$

The security of RSA:

Five possible approaches to attacking the RSA algorithm are

- 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.
- Real-world applications: HTTPS websites, signed software updates, encrypted email

ElGamal Encryption and Digital Signatures

- Based on the difficulty of computing discrete logarithms
- Used in PGP and GNU Privacy Guard
- Real-world applications: Encrypted messaging (Signal, WhatsApp), secure email systems

Elliptic Curve Cryptography

- Offers equivalent security to RSA with smaller key sizes
- Used in Bitcoin, Signal, TOR, and other cryptocurrencies
- Real-world applications: Mobile devices, smart cards, IoT devices

Further Reading

- Introduction to Public-Key Cryptography
- Practical Cryptography for Developers
- A Graduate Course in Applied Cryptography
- Real World Cryptography

Advantages of public-key cryptography

1. Only the private key must be kept secret (authenticity of public keys must, however, be guaranteed).

2. The administration of keys on a network requires the presence of only a functionally trusted TTP as opposed to an unconditionally trusted TTP. Depending on the mode of usage, the TTP might only be required in an “off-line” manner, as opposed to in real time.
3. Depending on the mode of usage, a private key/public key pairmay remain unchanged for considerable periods of time, e.g., many sessions (even several years).
4. Many public-key schemes yield relatively efficient digital signature mechanisms. The key used to describe the public verification function is typically much smaller than for the symmetric-key counterpart.
5. In a large network, the number of keys necessary may be considerably smaller than in the symmetric-key scenario.

Disadvantages of public-key encryption

1. Throughput rates for the most popular public-key encryption methods are several orders of magnitude slower than the best known symmetric-key schemes.
2. Key sizes are typically much larger than those required for symmetric-key encryption (see Remark 1.53), and the size of public-key signatures is larger than that of tags providing data origin authentication from symmetric-key techniques.
3. No public-key scheme has been proven to be secure (the same can be said for block ciphers). The most effective public-key encryption schemes found to date have their security based on the presumed difficulty of a small set of number-theoretic problems.
4. Public-key cryptography does not have as extensive a history as symmetric-key encryption, being discovered only in the mid 1970s.

Symmetric Encryption v/s Asymmetric Encryption

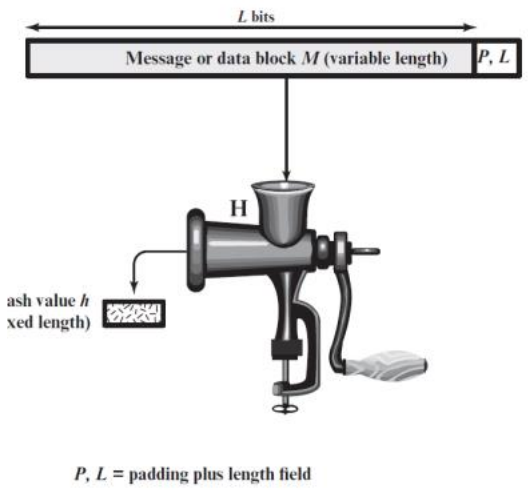
According to	**Symmetric Encryption**	**Asymmetric Encryption**

Keys used	The communication is encrypted and decrypted using just one shared key (secret key).	It encrypts data using one key, then decrypts it using a different key.
Cipher text size	Compared to the original plain text file, the cipher text is smaller.	The cipher text is more significant than the original plain text file.
The amount of data	used to send large amounts of data.	used to send small amounts of data.
Utilisation of Resources	Symmetric key encryption uses only a few resources to function.	High resource usage is necessary for asymmetric encryption.
Key Lengths	Key sizes are 128 or 256 bits.	RSA key size of 2048 bits or more.
Efficiency	Given that this method should be used for texts with extensive data, it is effective.	Since only brief messages are utilized with this approach, it is ineffective.
Approaches	It is an old approach.	It is a novel encryption method.
Security	There is a risk that a single key used for encryption and decoding can be compromised.	Separately created keys for encryption and decryption
Rapidity	Symmetric encryption is a quick method.	The speed of asymmetric encryption is slower.
Algorithms	RC4, AES, DES, 3DES, and QUAD.	RSA, Diffie-Hellman, ECC algorithms.

Cryptographic Hash Functions

Cryptographic hash functions are fundamental tools in modern information security, designed to convert input data of any size into a fixed-size output called a hash or digest. These functions play a role in various security applications, from data integrity verification to digital signatures.

- A hash function H accepts a variable-length block of data **M** as input and produces a fixed-size hash value **h = H(M).**
- A "good" hash function has the property that the results of applying the function to a large set of inputs will produce outputs that are evenly distributed and random.
- In general terms, the principal object of a hash function is data integrity. A change to any bit or bits in **M** results, with high probability, in a change to the hash value.
- The kind of hash function needed for security applications is called a **cryptographic hash function**.
- A cryptographic hash function is an algorithm for which it is computationally infeasible (because no attack is significantly more efficient than brute force) to find either o a data object that maps to a pre-specified hash result (the one-way property) or two data objects that map to the same hash result (the collision-free property).
- Because of these characteristics, hash functions are often used to determine whether or not data has changed.


L bits
Message or data block M (variable length)
P, L
H
ash value h
xed length)
P, L = padding plus length field

Cryptographic Hash function h=H(M)

The above figure depicts the general operation of a cryptographic hash function.

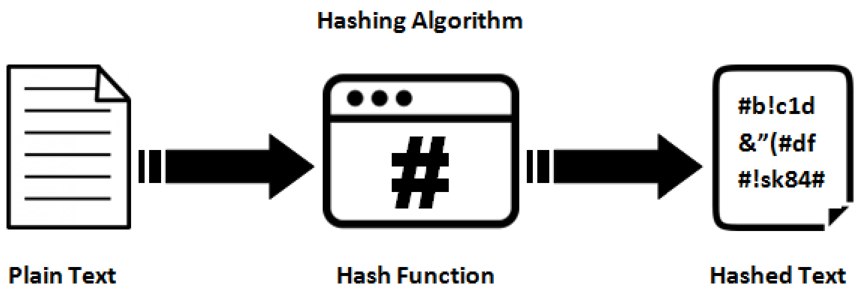
- Typically, the input is padded out to an integer multiple of some fixed length (e.g., 1024 bits), and the padding includes the value of the length of the original message in bits.
- The length field is a security measure to increase the difficulty for an attacker to produce an alternative message with the same hash value.

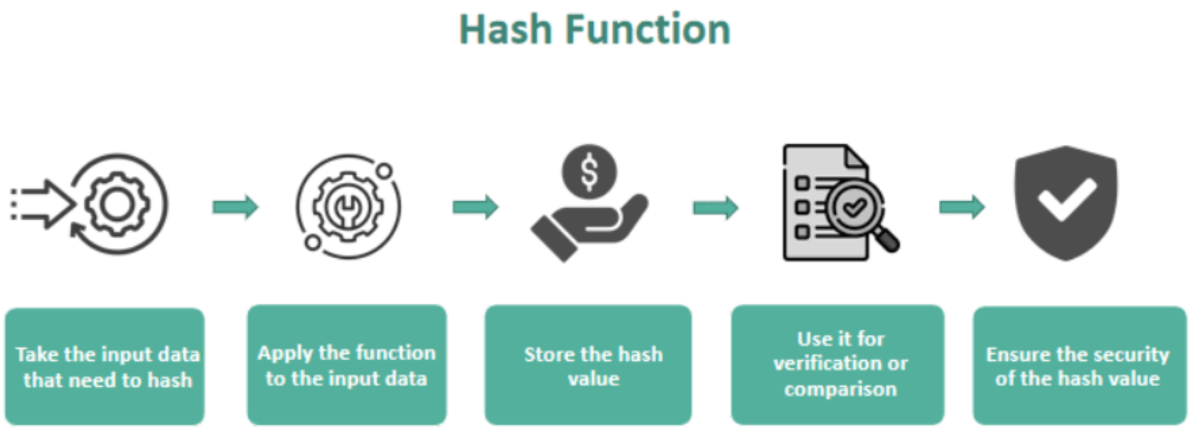
Properties of Hash Functions

Cryptographic hash functions possess several properties that make them suitable for security applications including:

1. **Deterministic**: The same input always produces the same hash output.
2. **Quick Computation**: Hash values can be calculated for any given input.
3. **Pre-image Resistance**: Given a hash value, it should be computationally infeasible to find any input that produces that hash.
4. **Second Pre-image Resistance**: Given an input, it should be difficult to find another input that produces the same hash.
5. **Collision Resistance**: It should be computationally infeasible to find two different inputs that produce the same hash output.

6. **Avalanche Effect**: A small change in the input should result in a different hash output.




Hash Function
Take the input data that need to hash
Apply the function to the input data
Store the hash value
Use it for verification or comparison
Ensure the security of the hash value

Applications of Cryptographic Hash Functions

The most versatile cryptographic algorithm is the cryptographic hash function. It is used in a wide variety of security applications and Internet protocols. The following are various applications where it is employed.

1. Digital Signatures

Purpose: Digital signatures use cryptographic hashes to verify the authenticity and integrity of messages.

Process: A message is hashed, and this hash is encrypted with the sender's private key. The recipient can decrypt it with the sender's public key and compare it to a new hash of the received message to ensure authenticity.

2. File Integrity Verification

Purpose: To ensure that downloaded files have not been tampered with or corrupted during transmission.

Process: Websites publish hash values for downloadable files; users can verify these hashes after downloading to confirm file integrity.

3. Password Security

Purpose: Passwords are stored as hashes rather than plaintext, enhancing security by making it difficult for attackers to obtain original passwords even if they access stored data.

4. Blockchain Technology

Cryptocurrencies like Bitcoin use SHA-256 for maintaining transaction records' integrity and security[3].

5. Message Authentication Codes (MACs)

Used in conjunction with secret keys to authenticate messages, ensuring they come from an authorized source without modification[7].

6. SSL/TLS Protocols

These secure communication protocols rely on cryptographic hash functions for authentication and encryption processes[1].

7. Intrusion Detection and Virus Detection

Hashes can be used to identify known malicious software patterns by comparing them against databases of known malware hashes[6].

8. Key Derivation Functions (KDFs)

Used to derive multiple keys from a single password or passphrase, enhancing security by separating encryption keys from authentication keys[5].

These applications highlight the versatility and importance of cryptographic hash functions in maintaining digital security across various domains.

Common Cryptographic Hash Functions:

Secure Hash Algorithm (SHA) Family

The SHA family, developed by the National Institute of Standards and Technology (NIST), includes several generations of hash functions:

1. **SHA-1**: Produces a 160-bit hash value. Now considered cryptographically weak and not recommended for security applications.
2. **SHA-2**: Includes SHA-224, SHA-256, SHA-384, and SHA-512. These are still widely used and considered secure.
3. **SHA-3**: The newest member of the SHA family, based on the Keccak algorithm. It offers improved security and performance compared to SHA-2

Message Digest (MD) Algorithm Family

The MD family, developed by Ronald Rivest, includes these two algorithms:

1. **MD5**: Produces a 128-bit hash value. While still widely used for non-security purposes, it's considered cryptographically broken and unsuitable for security applications.

2. **MD6**: A newer algorithm designed to be more secure than MD5, but not widely adopted.

Security Status

While innovative in their time, the MD4 and MD5 algorithms are now considered cryptographically broken. Collision attacks have been demonstrated against both. They should not be used in applications requiring strong cryptographic security.

Legacy and Influence

Despite their weaknesses, the MD algorithms were very influential in hash function design. Many modern hash functions like SHA-1 and SHA-2 use similar overall structures and operations, albeit with improvements to address the flaws in MD4/MD5.For secure hashing today, algorithms like SHA-256 or SHA-3 are recommended instead of the MD family. However, MD5 is still sometimes used for non-cryptographic purposes like checksums.

Further Reading

1. "Introduction to Modern Cryptography" by Jonathan Katz and Yehuda Lindell
2. NIST's Secure Hash Standard (FIPS 180-4): https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
3. "Cryptography Engineering" by Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno
4. "Applied Cryptography" by Bruce Schneier
5. "Handbook of Applied Cryptography" by Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone (Chapter 9)

Message Authentication Codes and Digital Signatures

Message Authentication Codes (MACs) are cryptographic techniques with a one-way hash function with the addition of a secret key used to verify the integrity and authenticity of messages. A message authentication code (MAC), is also known as a data authentication code (DAC). You can create a MAC out of a hash function or a block encryption algorithm; there are also dedicated MACs They work as follows:

1. The sender and receiver share a secret key.
2. The sender creates a MAC by applying an algorithm that combines the message and the secret key.
3. The sender transmits both the message and the MAC.
4. The receiver recreates the MAC using the same algorithm, message, and shared key.
5. If the receiver's MAC matches the sender's MAC, the message is authentic and unaltered.

Common MAC algorithms include:

- HMAC (Hash-based Message Authentication Code)
- CMAC (Cipher-based Message Authentication Code)
- GMAC (Galois Message Authentication Code)

MACs provide data integrity and authentication but not non-repudiation, as both parties know the shared key.

Digital Signature Algorithms

Previously handwritten signatures serve as proof of authorship or agreement with a document's contents. Digital Signature Algorithms (DSA) are essential components of modern cryptography, ensuring the authenticity, integrity, and non-repudiation of digital messages and documents. Their compelling nature stems from five attributes:

1. Authenticity: Confirms deliberate signing
2. Unforgeability: Proves only the signer signed
3. Non-reusability: Cannot be moved to another document
4. Unalterability: Document cannot be changed after signing
5. Non-repudiation: Signer cannot deny signing

However, these attributes are not absolute. Signatures can be forged, moved, and documents can be altered. We accept these risks due to the difficulty of cheating and the likelihood of detection. Digital signatures face unique challenges:

1. Easy file copying: Valid signatures can be easily copied and pasted
2. Simple file modification: Documents can be altered after signing without evidence

These issues make replicating the security of handwritten signatures in digital form more complex.

Digital signatures *are the electronic equivalent of handwritten signatures, providing authentication, integrity, and non-repudiation*. They work using public-key cryptography:

1. The signer has a private key (kept secret) and a public key (shared openly).
2. To sign a document, the signer creates a hash of the document and encrypts it with their private key.
3. Anyone can verify the signature using the signer's public key to decrypt the hash and compare it to a newly computed hash of the document.

Common digital signature algorithms include:

- RSA (Rivest-Shamir-Adleman)
- DSA (Digital Signature Algorithm)
- ECDSA (Elliptic Curve Digital Signature Algorithm)

Digital signatures provide stronger security than MACs because they use asymmetric cryptography, ensuring non-repudiation.

Public Key Infrastructure (PKI)

PKI is a system that manages digital certificates and public-key encryption. It consists of:

1. Certificate Authorities (CAs): Trusted entities that issue and verify digital certificates.
2. Registration Authorities (RAs): Verify the identity of entities requesting certificates from a CA.
3. Digital certificates: Electronic documents that bind a public key to an entity's identity.
4. Certificate repositories: Databases of issued certificates.

5. Certificate Revocation Lists (CRLs): Lists of certificates that are no longer valid.

PKI enables secure communication and authentication in various applications, including secure web browsing (HTTPS), email encryption, and digital signatures.

Certificate Authorities and Digital Certificates

Certificate Authorities (CAs) are trusted third parties that issue digital certificates. They help to establish trust on the internet by verifying the identities of certificate applicants and digitally signing the issued certificates. Digital certificates contain:

- The subject's identity (e.g., domain name for websites)
- The subject's public key
- The issuing CA's identity
- The certificate's validity period
- The CA's digital signature

When you visit a secure website (HTTPS), your browser verifies the site's certificate by checking the CA's signature and ensuring the certificate is valid and not revoked. There are different types of digital certificates, including:

- SSL/TLS certificates for secure websites
- Code signing certificates for software developers
- Email certificates for secure email communication
- Client certificates for user authentication

Further Reading

1. "Understanding Cryptography" by Christof Paar and Jan Pelzl - Chapters on MAC and digital signatures
2. "Applied Cryptography" by Bruce Schneier - In-depth coverage of cryptographic protocols
3. NIST Special Publication 800-57: Recommendation for Key Management - Detailed guidelines on cryptographic key management
4. RFC 5280: Internet X.509 Public Key Infrastructure Certificate and CRL Profile - Technical details of digital certificates

5. "PKI Uncovered: Certificate-Based Security Solutions for Next-Generation Networks" by Andre Karamanian, et al. - guide to PKI implementation

- MAC algorithms
- Digital signature algorithms
- Public Key Infrastructure
- Certificate Authorities and digital certificates

Key Management and Distribution

Key Distribution Technique is a term that refers to the means of delivering a key to two parties who wish to exchange data without allowing others to see the key. Key management and distribution are aspects of cryptographic systems often being the weakest link in secure communication systems. Even with strong encryption algorithms, the security of the system depends on keeping the keys secret and distributing them securely hence poor key management can compromise the entire system. Here's why:

1. Vulnerability: Keys obtained from unreliable sources can compromise security without breaking algorithms or exploiting protocol flaws.
2. Human factor: People are often easier to exploit than cryptosystems. Methods include:
    - Bribery
    - Theft
    - Coercion
    - Seduction
3. Cost-effectiveness: For attackers, exploiting human weaknesses is often cheaper and easier than cryptanalysis.
4. Protection requirements: Keys must be protected as thoroughly as the data they encrypt.
5. Implementation flaws: Even strong algorithms can be rendered useless by poor implementation. For example, the DiskLock program for Macintosh

used DES encryption but stored the key with the encrypted file, making it easily accessible.

1. Key Generation Techniques

- Keys should be generated using cryptographically secure random number generators
- For symmetric keys, use at least 128 bits of randomness
- For asymmetric keys, use at least 2048 bits for RSA and 256 bits for elliptic curve cryptography
- Hardware random number generators can provide higher quality randomness than software-based ones
- Key generation should be done in a secure environment to prevent compromise

Key Distribution Protocols

Alice and Bob need to securely share a symmetric encryption key. Here are the main methods and protocols:

Symmetric Key Distribution

Key Management and Distribution: The Basics

- **Key Distribution Problem**-How to securely give secret keys to people who need to communicate privately.
- **Symmetric Key Distribution**
  - The goal is to share a secret key between two parties (A and B) without anyone else seeing it.
  - Methods
    1. A physically gives the key to B.
    2. A trusted third party physically gives the key to both A and B.
    3. If A & B have an old key, use it to encrypt and send the new key.
    4. A & B each have encrypted connection to third party (C), C sends key on encrypted links to A & B.

- **Key Hierarchy**
    - Master Key: Long-term key shared between a user and a Key Distribution Center (KDC).
    - Session Key: Temporary key used for actual communication, obtained from KDC.
    - KDC (Key Distribution Center): Responsible for giving keys to pairs of users.
- **A Key Distribution Scenario (using KDC)**
    - A asks KDC for a session key to talk to B (includes a unique ID, called a nonce).
    - KDC sends A a message (encrypted with A's master key) that includes:
        1. The new session key.
        2. The original request.
        3. Info for B (the session key and A's ID), encrypted with B's master key.
    - A forwards the info intended for B.
    - B sends A a nonce encrypted with the new session key.
    - A responds to B using the session key.
- **Hierarchical Key Control**
    - Use multiple levels of KDCs (local and global).
    - Local KDCs handle key distribution within their area.
    - Global KDCs help local KDCs communicate.
- **Session Key Lifetime**
    - Trade-off: Frequent key changes = more security, but more overhead.
    - Options: Change keys with each new connection, periodically, or with each transaction.
- **Transparent Key Control:**

    - Session Security Module (SSM): Software that handles encryption and key requests without the user knowing.
- **Decentralized Key Control:**
    - No KDC. Each system must securely communicate with every other system.
- **Controlling Key Usage:**
    - Control Vector: Specifies what a session key can and cannot be used for.
- **Symmetric Key Distribution Using Asymmetric Encryption**
    - Using public-key cryptography to encrypt secret keys for distribution.

Public Key Distribution

1. Public-key cryptography (if available)
2. Public key certificates: Bind public keys to identities, signed by Certificate Authority (CA)
3. Public Key Infrastructure (PKI): System of CAs, registration authorities, and directories to manage certificates

Lets put everything together and understand whats going on

The Big Picture: *Imagine a company where employees need to send secret messages to each other. Key management is like setting up a secure system to make sure only the right people can read those messages.*

The Players & Their Roles:

- ***Key Distribution Center (KDC):*** *Think of this as the company's security office. Its main job is to hand out secret keys safely.*
- ***Master Key:*** *This is a long-term secret key that each employee shares only with the security office (KDC). It's like a special ID card that proves who they are to the KDC.*
- ***Session Key:*** *This is a short-term secret key used for the actual secret message between two employees. The KDC gives out these temporary keys.*

How It Works (The Simple Scenario):

1. ***Alice Needs a Key:*** *Alice (an employee) wants to send a secret message to Bob. She asks the KDC for a session key to talk to Bob.*
2. ***KDC's Response:*** *The KDC creates a new session key and sends it to Alice. But, it's not that simple!*
    - *The message to Alice is encrypted using Alice's Master Key. Only Alice and the KDC know this, so it's secure.*
    - *The message includes the session key and a part specifically for Bob (containing the session key and Alice's ID), encrypted with Bob's Master Key.*
3. ***Alice Passes the Message to Bob:*** *Alice can't read the part intended for Bob, so she just forwards it to him.*
4. ***Bob Gets the Key:*** *Bob receives the message, decrypts his part using his master key, and gets the session key.*
5. ***Verification:*** *Bob sends Alice a secret code (nonce) to prove he really got the key and is who he says he is. Alice responds to confirm.*

Why This Setup?

- ***Security:*** *Master keys are rarely used, reducing the risk of them being compromised. Session keys are temporary, so even if one is stolen, it's only useful for a short time.*
- ***Centralized Control:*** *The KDC manages who gets which keys, making it easier to track and control access.*

Other Important Ideas:

- ***Hierarchical Key Control:*** *If the company is big, there might be local security offices (local KDCs) and a main headquarters (global KDC). Local offices handle their area, and the global office helps them talk to each other.*
- ***Session Key Lifetime:*** *How long should a session key be valid? Too long, and it's a security risk. Too short, and it's a hassle to keep getting new ones.*

- ***Transparent Key Control:*** *The system works in the background. Employees don't have to worry about the technical details of getting keys. The software handles it for them.*
- ***Decentralized Key Control:*** *Instead of a central KDC, each employee has to securely exchange keys with everyone they might want to talk to. This is more complex, but it removes the single point of failure (the KDC).*
- ***Control Vector:*** *A set of rules about what a session key can and cannot be used for (e.g., "This key is only for encrypting emails, not for accessing bank accounts").*
- ***Symmetric Key Distribution Using Asymmetric Encryption:*** *Using methods that do not use master keys, using public and private keys to distribute.*

How it all relates

These are different approaches to address the key management and distribution problem. The concepts fit together by providing different solutions and ways to manage and distribute keys securely. Each method has its own advantages and disadvantages, and the choice of which method to use depends on the specific needs and requirements of the system.

Key Agreement Protocols

- Allow two parties to agree on a shared secret key over an insecure channel
- Diffie-Hellman key exchange:
    - Parties exchange public values and combine with private values to derive shared secret
    - Vulnerable to man-in-the-middle attacks without authentication
- Authenticated key agreement:
    - Station-to-Station protocol: Combines Diffie-Hellman with digital signatures
    - MQV protocol: More efficient authenticated key agreement

Key Escrow and Recovery

- Key escrow: Copies of keys held in escrow by trusted third party

- Allows authorized access to keys for law enforcement or business continuity
- Controversial due to privacy and security concerns
- Key recovery: Methods to recover lost keys, e.g., secret sharing schemes

Further Reading

- Handbook of Applied Cryptography by Menezes et al., Chapter 13
- Cryptography and Network Security by William Stallings, Chapter 14
- Applied Cryptography by Bruce Schneier, Chapter 8
- NIST Special Publication 800-57: Recommendation for Key Management