Lecture 8.1

Network Security

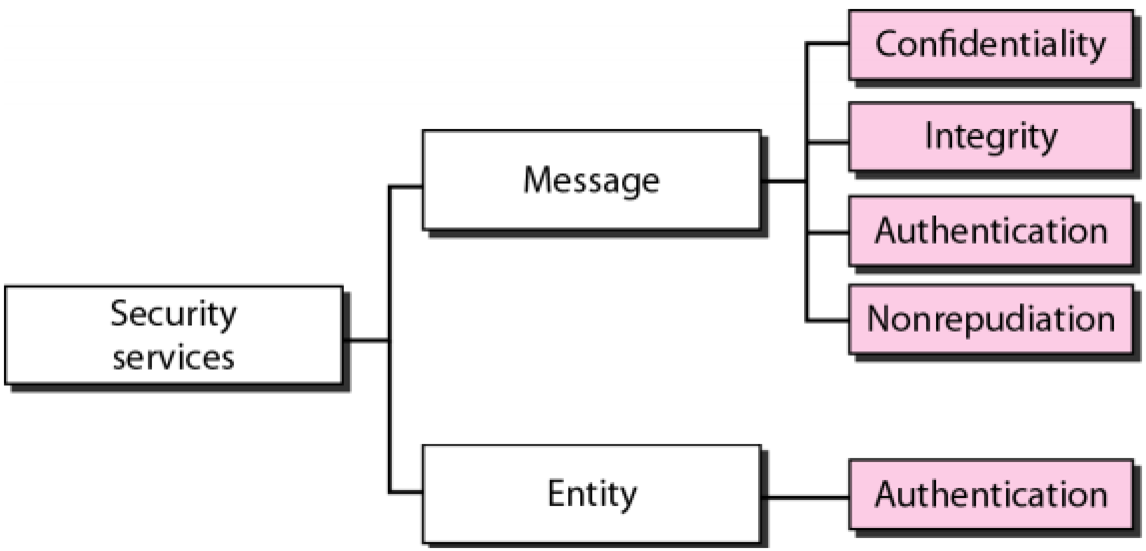
Dr. Vandana Kushwaha

Department of Computer Science

Institute of Science, BHU, Varanasi

SECURITY SERVICES

- **Network security** can provide one of the **five services** as shown in **Figure** below.
- **Four** of these **services** are **related** to the **message exchanged** using the **network:** *message confidentiality, integrity, authentication*, and *nonrepudiation.*
- The **fifth service** provides **entity authentication** or **identification.**



SECURITY SERVICES

1. Message Confidentiality

- **Message confidentiality** or **privacy** means that the **sender** and the **receiver** expect **confidentiality.**
- The **transmitted message** must **make sense** to **only** the **intended receiver.**
- To **all others**, the **message** must be **garbage.**
- When a **customer** communicates with her **bank**, she **expects** that the **communication** is **totally confidential.**

SECURITY SERVICES

2. Message Integrity

- **Message integrity** means that the **data must arrive** at the **receiver exactly as they were sent.**
- There **must be no changes** during the **transmission**, **neither accidentally** nor **mali-ciously.**
- As more and more **monetary exchanges** occur over the **Internet, integrity** is **crucial.**
- For **example,** it would be **disastrous** if a **request** for **transferring $100 changed** to a **request** for **$10,000 or $100,000.**
- The **integrity** of the **message** must be **preserved** in a **secure communication.**

SECURITY SERVICES

3. Message Authentication

- **Message authentication** is a **service** beyond **message integrity.**
- In **message authentication** the **receiver needs to be sure** of the **sender's identity** and that an **imposter** has **not sent** the **message.**

4. Message Nonrepudiation

- **Message nonrepudiation** means that a **sender** must **not be able** to **deny sending a message** that he or she, in fact, **did send.**
- The **burden of proof** falls on the **receiver.**
- For **example**, when a **customer sends** a **message** to **transfer money** from **one account** to **another,** the **bank must have proof** that the **customer actually requested** this **transaction.**

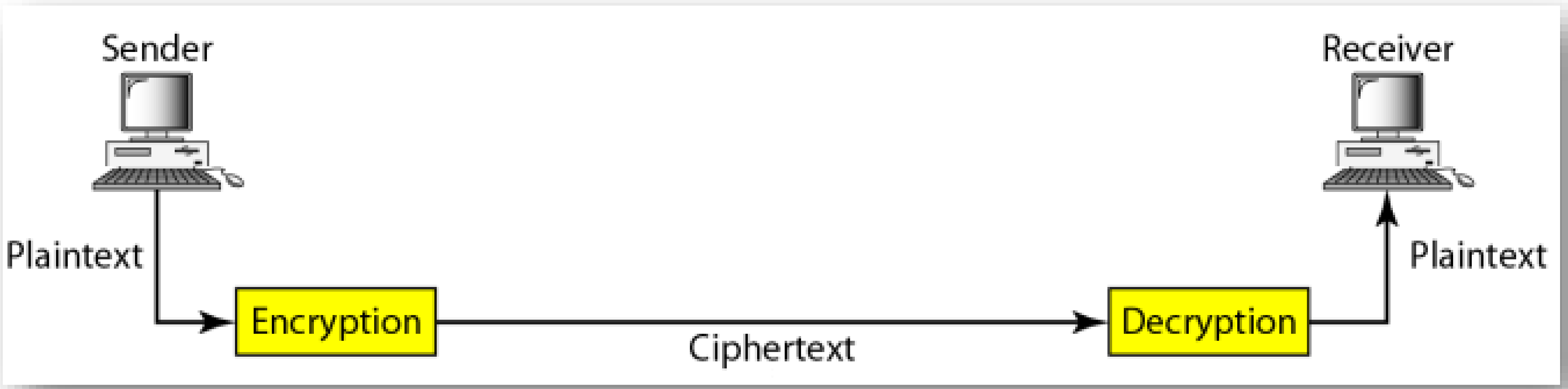
SECURITY SERVICES

5. Entity Authentication

- In **entity authentication** (or user identification) the **entity** or **user** is **verified** prior to **access** to the **system resources** (files, for example).
- For **example**, a **student** who **needs to access** her **university resources** needs to be **authenticated** during the **logging process.**
- This is to **protect** the **interests of the university** and the **student.**

Cryptography

- **Cryptography, a word** with **Greek origins,** means "**secret writing.**"
- However, we use the **term** to **refer** to the **science of transforming messages** to make them **secure and immune to attacks.**
- **Figure** below shows the **components** involved in **cryptography:**



Cryptography

Plaintext and Ciphertext

- The **original message**, before being transformed, is called **plaintext.**
- After the **message** is **transformed,** it is called **ciphertext.**
- An **encryption algorithm transforms** the **plaintext** into **ciphertext**;
- A **decryption algorithm transforms** the **ciphertext** back into **plaintext**.
- The **sender** uses an **encryption algorithm**.
- The **receiver** uses a **decryption algorithm**.

Cipher

- We refer to **encryption** and **decryption algorithms** as **ciphers**.
- The term ***cipher*** *is also* used to refer to different categories of **algorithms** in **cryptography.**

Cryptography

Key

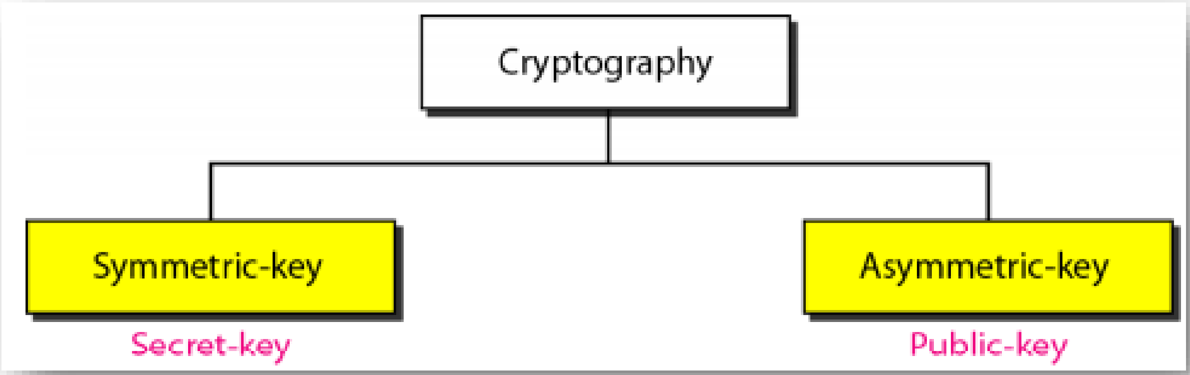
- A **key** is a **number (or a set of numbers)** that the **cipher,** as an **algorithm,** operates on.
- To **encrypt** a **message**, we need an **encryption algorithm**, an **encryption key**, and the **plaintext.**
- These create the **ciphertext.**
- To **decrypt a message**, we need a **decryption algorithm**, a **decryption key**, and the **ciphertext.**
- These **reveal** the **original plaintext.**

Categories of Cryptography Algorithms

We can divide all the **cryptography algorithms (ciphers)** into **two groups**:

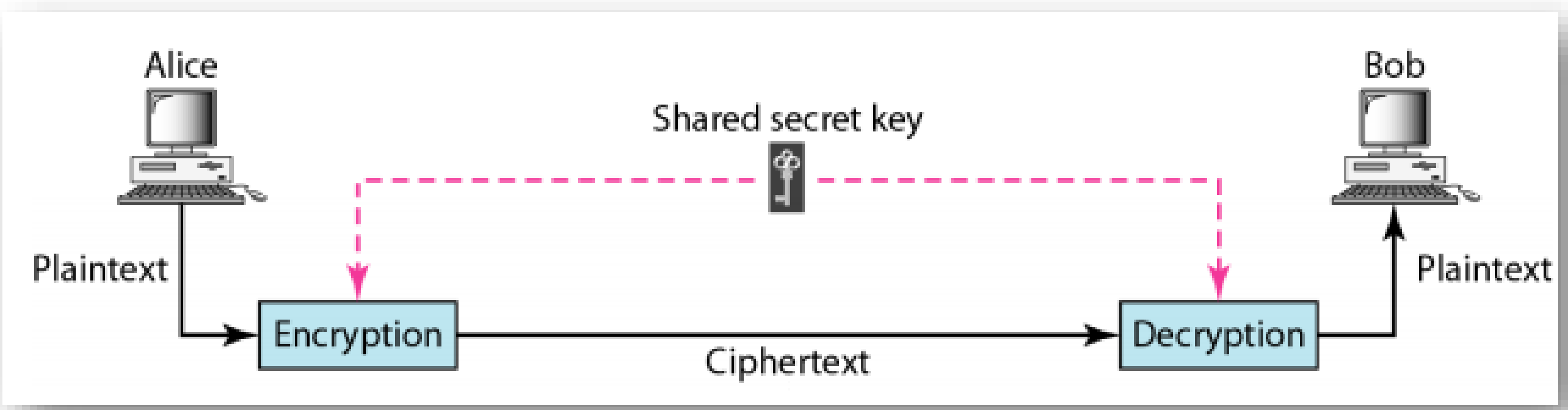
1. ***Symmetric key (also called secret-key) cryptography*** *algorithms*

2. ***Asymmetric (also called public-key) cryptography*** *algorithms.*



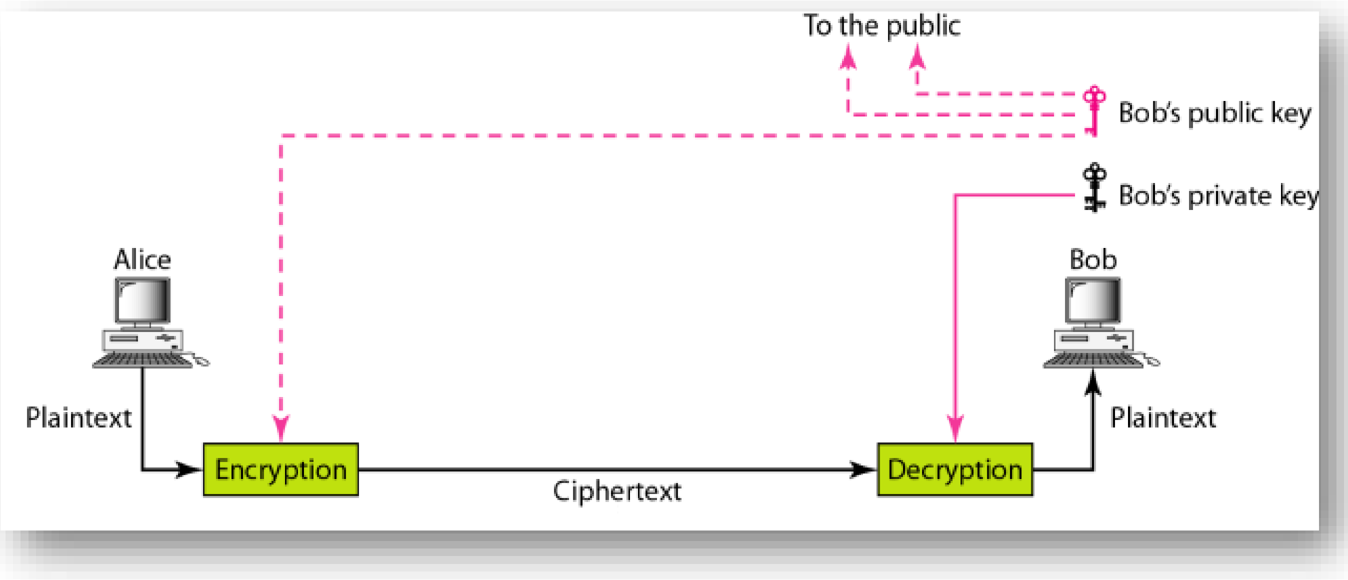
Symmetric Key Cryptography

- In **Symmetric-key cryptography**, the **same key** is used by **both parties.**
- The **sender** uses this **key** and an **encryption algorithm** to **encrypt data.**
- The **receiver** uses the **same key** and the corresponding **decryption algorithm** to **decrypt** the **data.**



Asymmetric-Key Cryptography

- In **Asymmetric** or **Public-key cryptography**, there are **two keys**: a ***private key*** and a ***public key.***
- The **private key** is **kept by** the **receiver.**
- The **public key** is **announced** to the **public** by the **receiver**.
- Imagine **Alice** wants to **send a message** to **Bob.**
- **Alice** uses the **Bob's public key** to **encrypt** the **message**.
- When the **message** is **received** by **Bob,** the **private key** is used to **decrypt** the **message.**

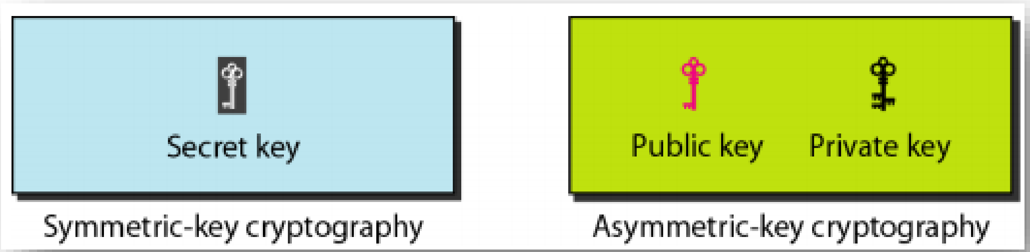


Asymmetric-Key Cryptography

- In **Asymmetric or public-key encryption/decryption**, the **public key** that is **used for encryption** is **different** from the **private key** that is **used for decryption.**
- The **public key** is **available** to the **public.**
- The **private key** is available **only** to an **individual.**

Three Types of Keys

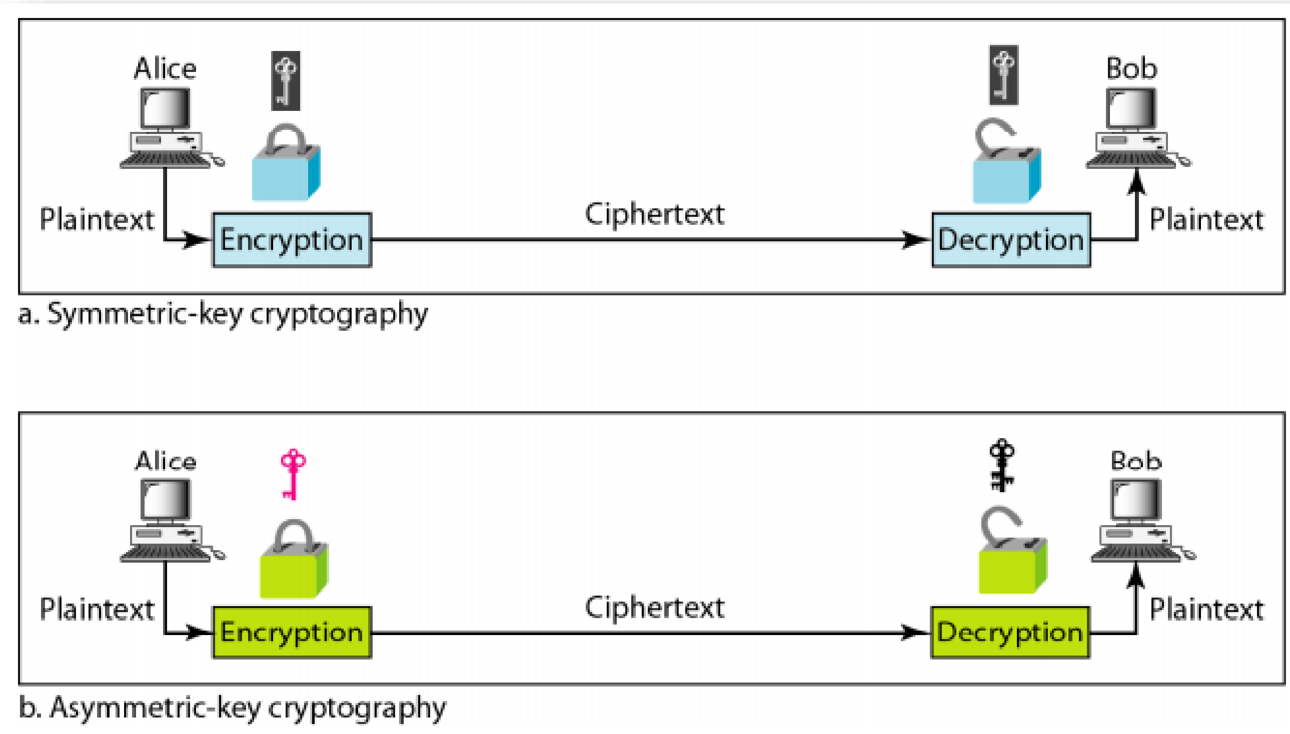
- We are dealing with **three types** of **keys** in **cryptography:**
  1. ***Secret key,***
  2. ***Public key,***
  3. ***Private key***
- The **first**, the **secret key**, is the **shared key** used in **symmetric-key cryptography**.
- The **second** and the **third** are the **public and private keys** used in **asymmetric-key cryptography.**



Symmetric-key vs. Asymmetric-key Cryptography

- **Encryption** can be **thought** of as **electronic locking** and **Decryption** as **electronic unlocking.**
- The **sender puts** the **message** in a **box** and **locks** the **box** by using a **key**;
- The **receiver unlocks** the **box** with a **key** and **takes out** the **message.**
- The **difference** lies in the **mechanism** of the **locking and unlocking** and the **type of keys used.**
- In **Symmetric-key cryptography**, the **same key locks and unlocks** the **box.**
- In **Asymmetric-key cryptography**, **one key locks** the **box,** but **another key is needed to unlock** it.

Symmetric-key vs. Asymmetric-key Cryptography



1.MESSAGE CONFIDENTIALITY

- The **concept** of **how to achieve** ***message confidentiality*** or ***privacy*** has not changed for thousands of years.
- The ***message*** must be **encrypted** at the **sender site** and **decrypted** at the **receiver site.**
- That is, the **message** must be **rendered unintelligible** to **unauthorized parties.**
- A **good privacy technique** **guarantees** to some extent that a **potential intruder (eavesdropper**) **cannot understand** the **contents** of the **message.**
- This can be done using either ***Symmetric-key cryptography*** or ***Asymmetric key cryptography.***

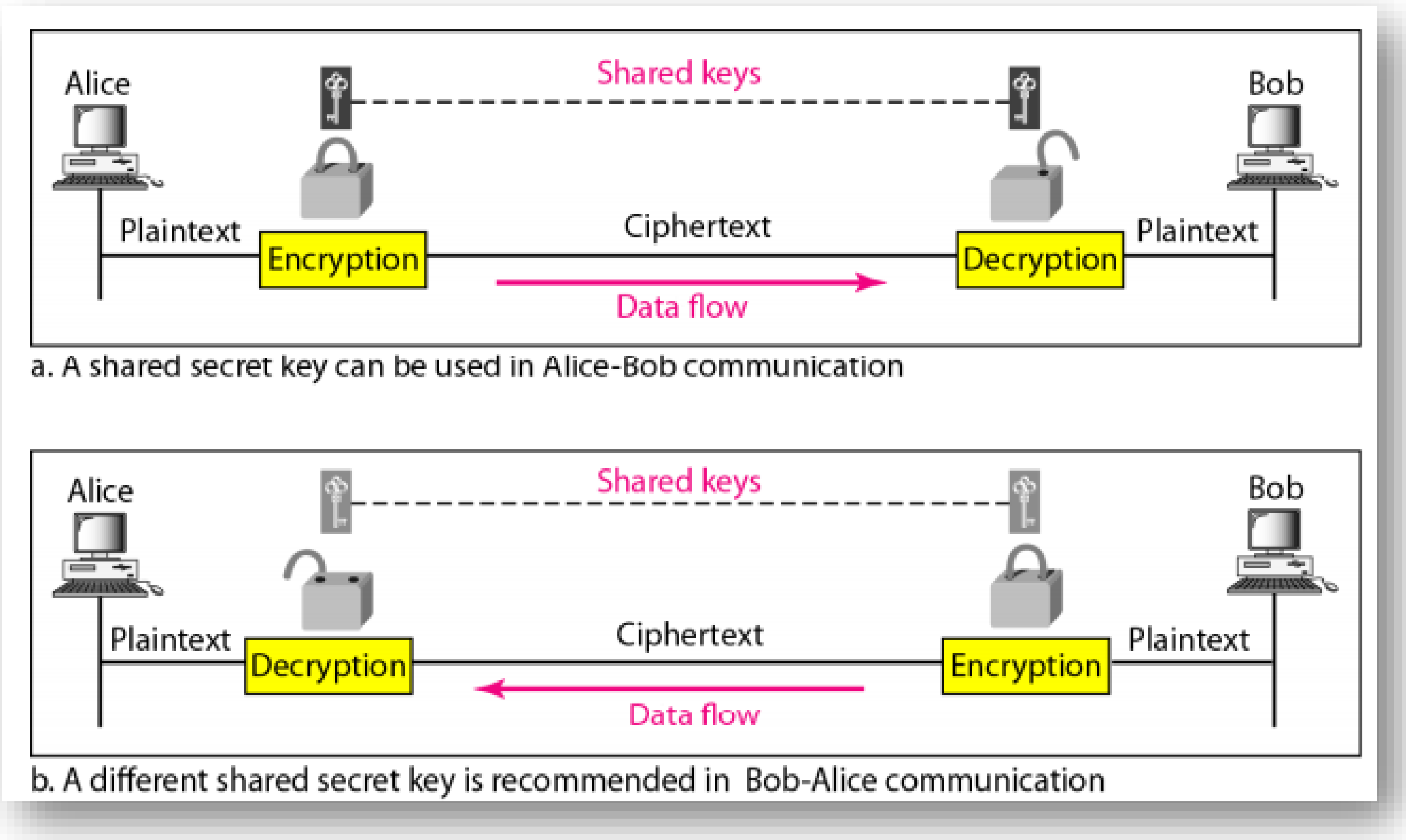
Confidentiality with Symmetric-Key Cryptography

- To provide **confidentiality** with **symmetric-key cryptography**, a **sender** and a **receiver** need to **share** a **secret key.**
- In the **past** when **data exchange** was between **two specific persons** (for example, two friends or a ruler and her army chief), it was **possible** to **personally exchange** the **secret keys.**
- Today's communication **does not** often provide this **opportunity.**
- A **person** residing in the **United States cannot meet** and **exchange** a **secret key** with a **person** living in **China.**
- Furthermore, the **communication** is between **millions of people**, not just a **few.**

Confidentiality with Symmetric-Key Cryptography

- To be **able** to use **symmetric-key cryptography**, we need to find a **solution** to the **key sharing.**
- This can be done using a **session key.**
- A **session key** is one that is **used only for** the **duration of one session**.
- The **session key** **itself** is **exchanged using** **asymmetric key cryptography .**
- Note that the **nature** of the **symmetric key** allows the **communication** to be carried on in **both directions** although it is not **recommended today.**
- Using **two different keys** is **more secure**, because **if one key** is **compromised**, the **communication** is **still confidential** in the **other direction.**

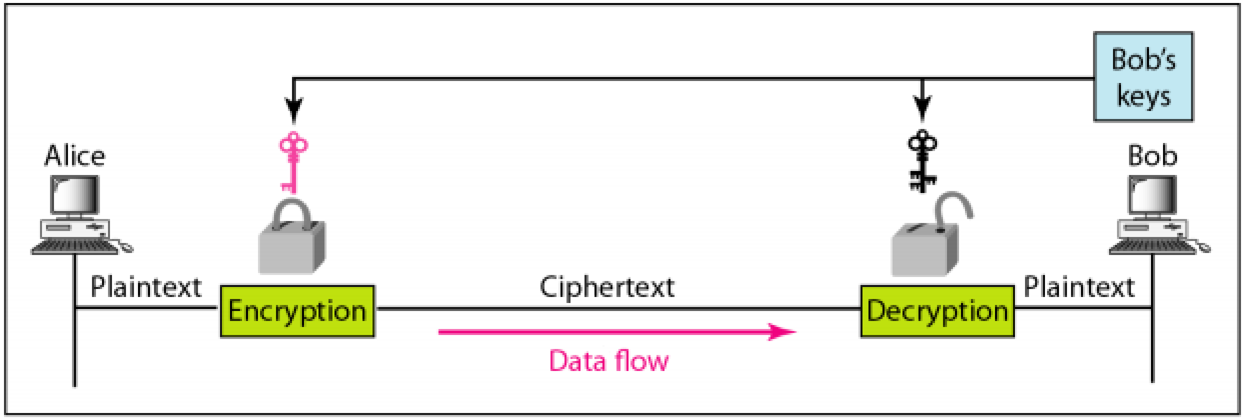
Message confidentiality using symmetric keys in two directions



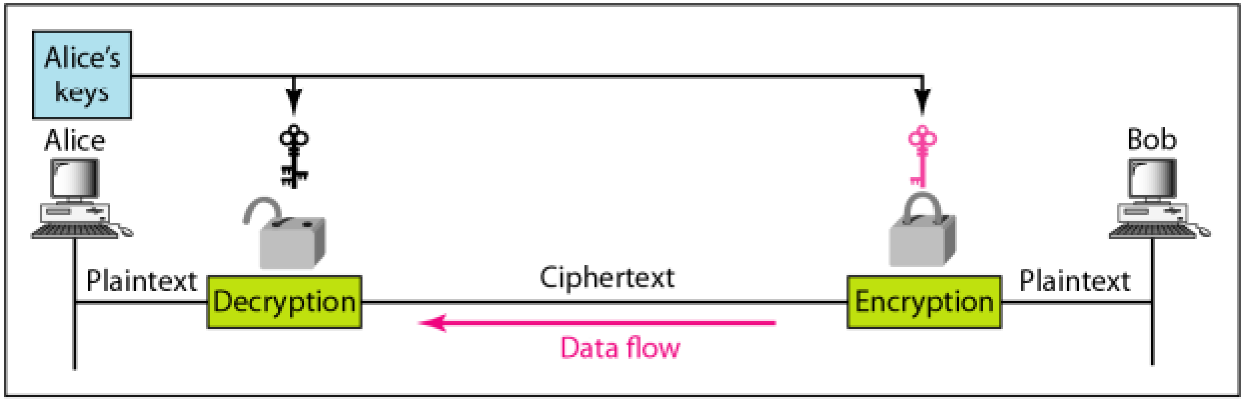
Confidentiality with Asymmetric-Key Cryptography

- In **Asymmetric-key cryptography**, there is **no key sharing**; there is a **public announcement.**
- **Bob** creates **two keys**: one **private** and one **public**.
- **He** keeps the **private key for decryption**; he **publicly announces the public key** to the **world.**
- The **public key** is used **only for encryption;** the **private key** is used **only for decryption.**
- The **public key** locks the **message**; the **private key** unlocks **it.**
- For a **two-way communication** between **Alice and Bob, two pairs of keys are needed.**
- When **Alice** sends a **message** to **Bob,** she uses **Bob's pair**; when **Bob** sends a **message** to **Alice**, he uses **Alice's pair.**

Message Confidentiality using Asymmetric keys



a. Bob's keys are used in Alice-Bob communication



b. Alice's keys are used in Bob-Alice communication

Message Confidentiality using Asymmetric Keys

- **Confidentiality** with **Asymmetric-key cryptosystem** has its own **problems.**
- **First,** the **method** is based on **long mathematical calculations** using **long keys.**
- This means that this **system** is **very inefficient for long messages**; it should be **applied only to short messages.**
- **Second,** the **sender** of the message still **needs to be certain about the public key of the receiver.**
- For **example**, in **Alice-Bob communication**, **Alice** needs to be **sure** that Bob's **public key is genuine**; **Eve** may have **announced her public key** in the **name of Bob.**

Examples of Cryptographic Algorithms

- Examples of **Symmetric Encryption** include:
  - DES(Data Encryption Standard), Triple DES- 1977
  - RC4, RC5, RC6(Rivest Cipher)- 1987
  - Blowfish and TwoFish-1993
  - AES(Advanced Encryption Standard)-2001
- Examples **of Asymmetric Encryption** include:
  - Diffie-Hellman exchange method- 1976
  - Rivest Shamir Adleman (RSA)-1977
  - Elliptical Curve Cryptography (ECC)-1985

Symmetric Ciphers

Plaintext	V	O	Y	A	G	E	R
Key	+3	+3	+3	+3	+3	+3	+3
Ciphertext	Y	R	B	D	J	H	U

Substitution Ciphers

Plaintext	V	O	Y	A	G	E	R
Key	+1	+2	+3	+1	+2	+3	+1
Ciphertext	W	Q	B	B	I	H	S

Plaintext	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Key	T	O	E	U	N	Z	I	A	G	X	P	Q	Y	R	H	V	S	M	D	F	C	J	W	B	K	L

Plaintext	V	O	Y	A	G	E	R
Ciphertext	J	H	K	T	X	N	M

V	O	Y	A	G	E	R
O	V	A	Y	E	G	R

Transposition Ciphers

2.MESSAGE INTEGRITY

- **Encryption** and **Decryption provide Confidentiality,** but **not Integrity.**
- However, on occasion we may **not** even **need secrecy**, but instead **must have integrity**.
- For **example, Alice** may write a **will** to distribute her **estate** upon her **death.**
- The **will** does **not need** to be **encrypted,** after her **death, anyone** can **examine** the **will.**
- The **integrity of the will**, however, **needs to be preserved**.
- **Alice does not want** the **contents of the will to be changed.**
- As another **example,** suppose **Alice** sends a **message** instructing her **banker, Bob,** to pay **Eve** for **consulting work.**
- The **message** does **not need to be hidden** from **Eve** because **she already knows** she is to be **paid.**
- However, the **message does need to be safe from any tampering, especially by Eve.**

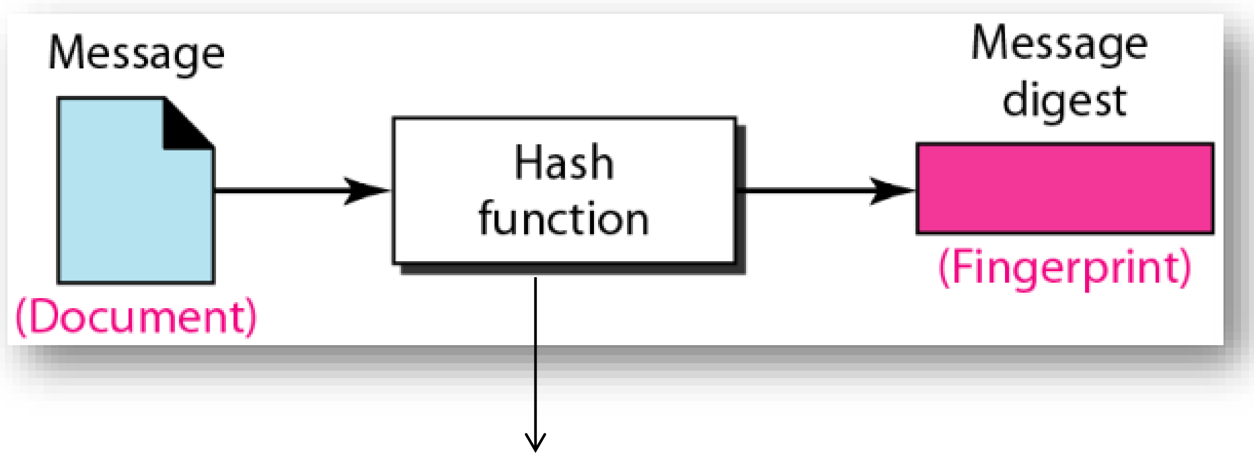
Document and Fingerprint

- One way to **preserve** the **integrity** of a **document** is through the **use** of a **fingerprint.**
- If **Alice** needs to be **sure** that the **contents** of her **document** will **not be illegally changed,** she can **put** her **fingerprint** at the **bottom** of the **document.**
- **Eve cannot modify** the **contents** of **this document** or **create a false document** because she **cannot forge Alice's fingerprint**.
- To **preserve the integrity** of a document, **both** the **document** and the **fingerprint** are **needed.**

Message and Message Digest

- The **electronic equivalent** of the **document** and **fingerprint pair** is the **message** and **message digest pair.**
- To **preserve the integrity** of a **message**, the **message is passed** through an **algorithm** called a **hash function**.
- The **hash function** creates a **compressed image of the message** that can be used as a **fingerprint.**
- The **document** and **fingerprint** are **physically linked together**; also, **neither needs** to be **kept secret**.
- However, the **message** and **message digest** can be **unlinked** (or sent) **separately** and, most importantly, the **message digest needs to be kept secret**.
- The **message digest** is either **kept secret** in a **safe place** or **encrypted** if we need to send it through a **communications channel**.

Message and Message Digest



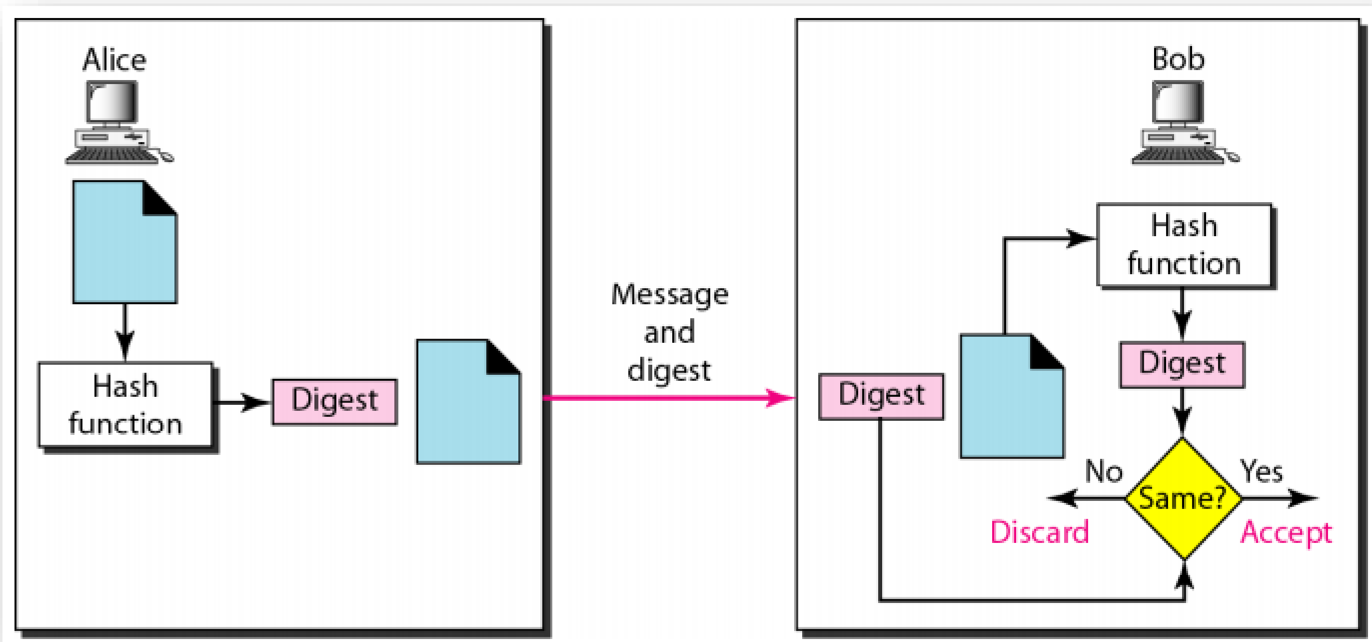
MDC
(modification detection code)

*__Hash function__ takes **input data** and uses it to create a **fixed-length output value** that's **unique** and **virtually irreversible.**

Creating and Checking the Digest

- The **message digest** is **created** at the **sender site** and is **sent with the message** to the **receiver.**
- To **check** the **integrity** of a **message,** or **document**, the **receiver uses** the **same hash function again** to generate the **local message digest** and **compares** the **local message digest** with the one **received**.
- If **both are the same,** the **receiver** is **sure** that the **original message has not been changed.**
- We are assuming that the **digest** has been **sent secretly.**
- **SHA-1(Secure Hash Algorithm 1)** is used for creating the **Digest.**

Checking integrity



3.MESSAGE AUTHENTICATION

- A **hash function guarantees** the **integrity** of a **message.**
- It **guarantees** that the **message has not been changed**.
- A **hash function**, however, **does not authenticate the sender** of the **message.**
- When **Alice** sends a message to **Bob**, **Bob** needs to know if the message is coming from **Alice** or **Eve**.
- To provide **message authentication**, **Alice** needs to provide **proof** that it is **Alice sending** the **message** and **not an imposter.**
- A **hash function** per se **cannot provide** such a **proof.**
- The **digest** created by a **hash function** is normally called a **modification detection code (MDC).**
- The **MDC can only detect** any **modification** in the **message**.

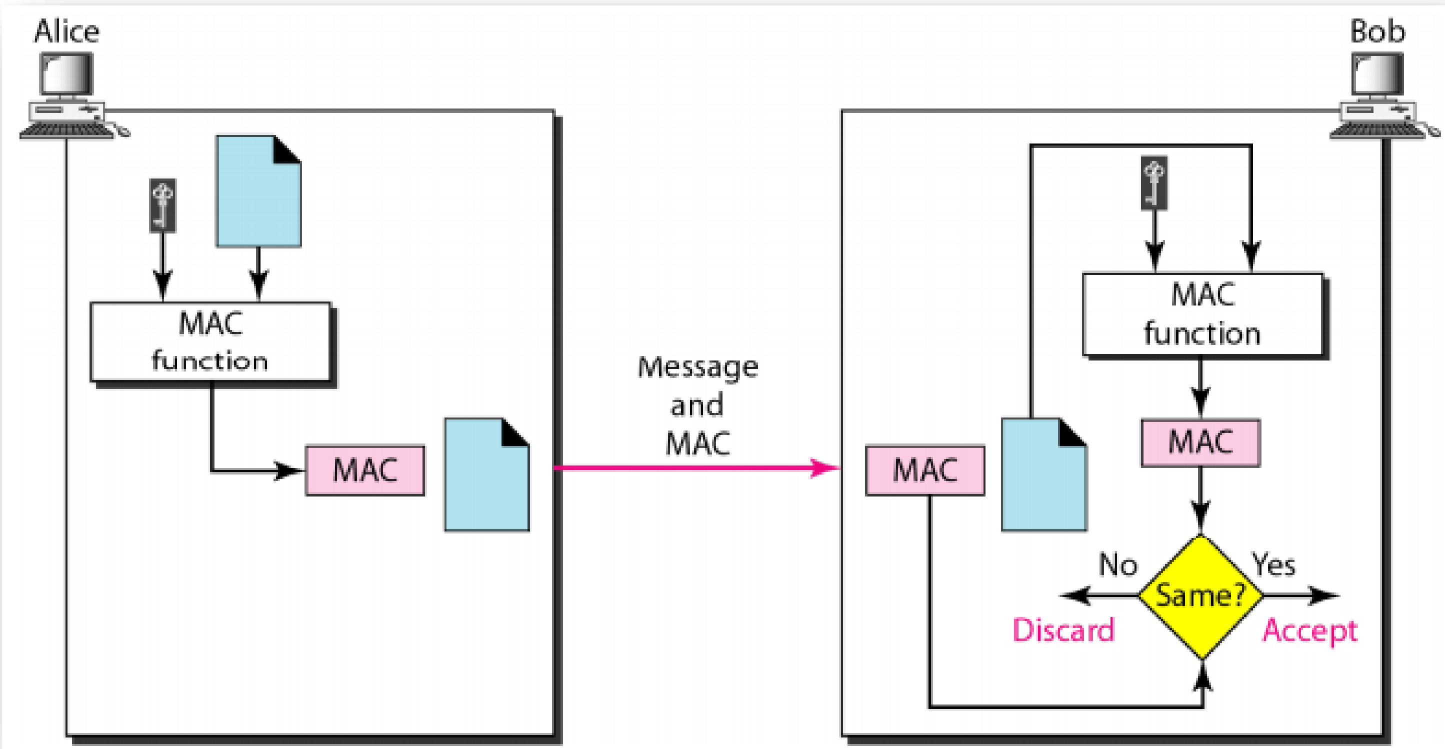
Message Authentication Code (MAC)

- To provide **message authentication**, we need a **Message Authentication Code (MAC).**
- An **MDC** uses a **keyless hash function** ; a **MAC uses a keyed hash function**.
- A **keyed hash function** includes the **symmetric key between** the **sender** and **receiver** when **creating the digest.**
- **Alice** uses a **keyed hash function** to **authenticate her message** and how **Bob** can **verify** the **authenticity** of the message.
- **Alice,** using the **symmetric key** between herself and **Bob *(K_{AB})*** *and a* ***keyed hash* function**, generates a **MAC.**
- She then **Concatenates** the **MAC with the original message** and sends the **two** to **Bob.**
- **Bob** receives the **message** and the **MAC.**

Message Authentication Code (MAC)

- **Bob separates** the **message** from the **MAC**.
- He **applies** the **same keyed hash function** to the **message** using the **symmetric key** K_{AB} to get a **fresh MAC.**
- He then **compares** the **MAC** sent by **Alice** with the **newly generated MAC.**
- If the **two MACs** are **identical,** the **message** has **not been modified (message integrity)** and the **sender** of the **message** is **definitely Alice (message authentication).**

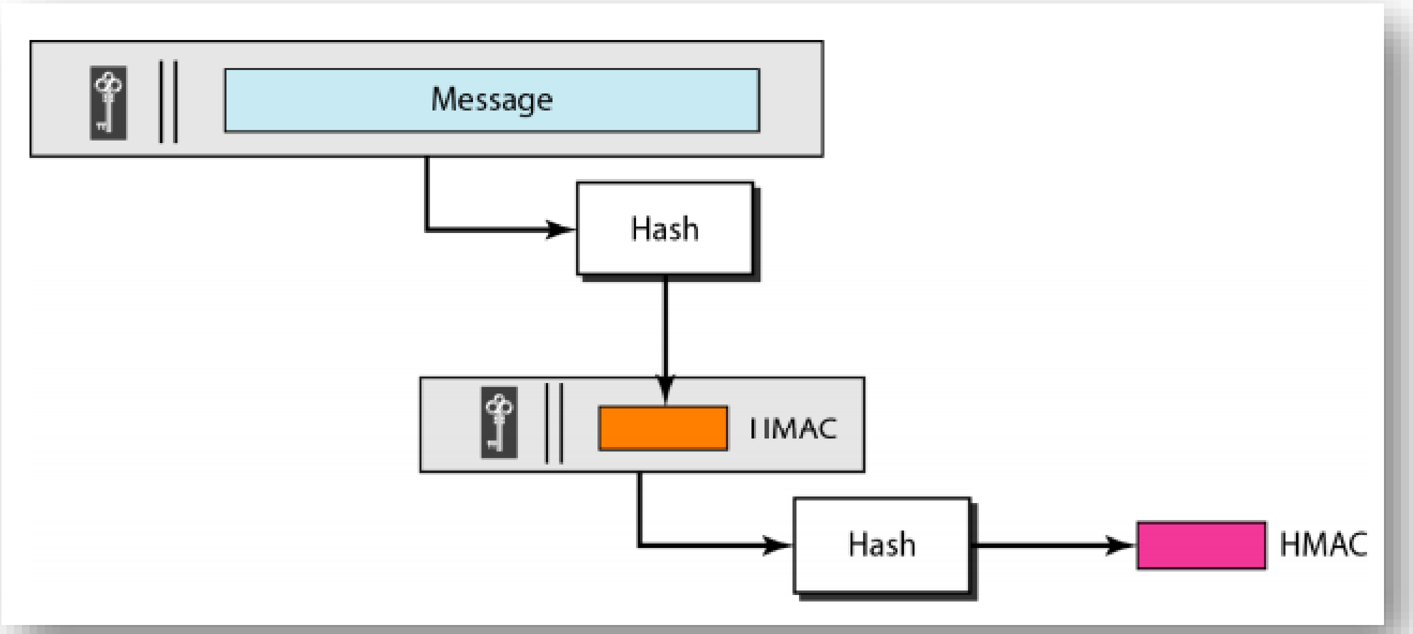
MAC, created by Alice and checked by Bob



HMAC(hashed MAC)

- **HMAC(hashed MAC)** can be used to create the **MAC.**
- **HMAC** can use any standard **keyless hash function** such as **SHA-1.**
- **HMAC** creates a **nested MAC** by applying a **keyless hash function** to the **concatenation** of the **message** and a **symmetric key.**
- A copy of the **symmetric key** is **prepended** to the **message.**
- The **combination** is **hashed** using a **keyless hash function**, such as **SHA-1.**
- The **result** of this **process** is an **intermediate HMAC** which is **again prepended** with the key (the same key), and the result is **again hashed** using the **same algorithm**.
- The **final result** is an **HMAC.**
- The **receiver** receives this **final HMAC** and the **message.**
- The **receiver creates** its **own HMAC** from the **received message** and **compares the two HMACs** to validate the **integrity** of the **message** and **authenticate** the **data origin**.

HMAC



DIGITAL SIGNATURE

- Although a **MAC can provide message Integrity** and **message Authentication**, it has a **drawback.**
- It **needs a symmetric key** that must be **established** between the **sender** and the **receiver.**
- A **digital signature**, on the other hand, can **use a pair of asymmetric keys** (a public one and a private one).
- **Sender sign a document** to show that it **originated** from him or was approved by him.
- The **signature is proof** to the **recipient** that the **document comes** from the **correct entity.**

DIGITAL SIGNATURE

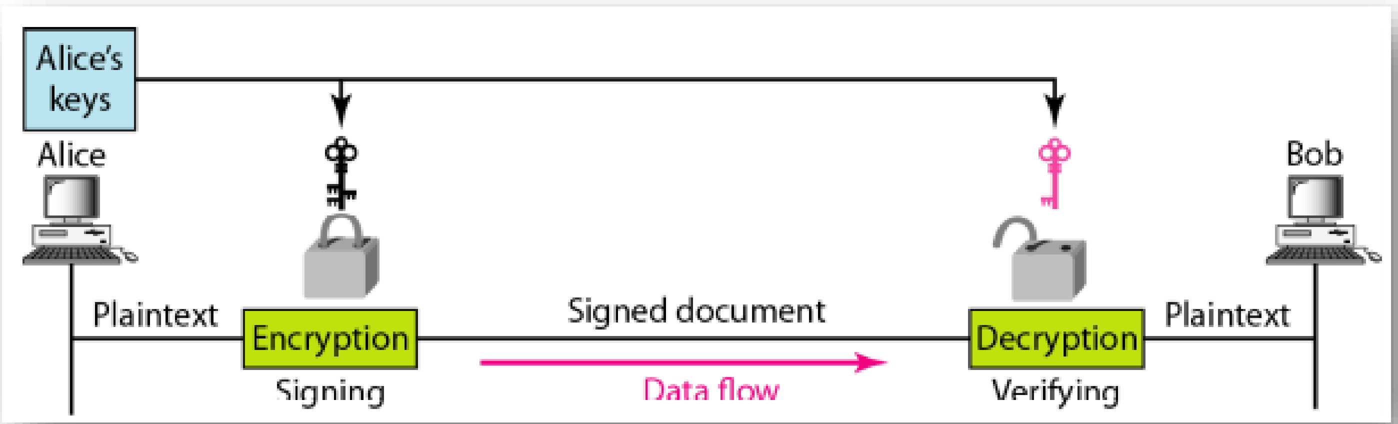
- In other words, a signature **on a document**, when verified, is a **sign** of authentication; the **document** is **authentic.**
- When Alice sends a **message** to Bob,
- Bob needs to **check** the authenticity of the **sender;** he needs to be sure that the **message** comes from Alice and not Eve.
- Bob can ask Alice to **sign** the **message electronically.**
- In other words, an electronic signature **can prove the** authenticity **of** Alice **as the** sender **of the** message.
- We **refer** to this **type of signature** as a Digital Signature.

Process

Digital Signature can be achieved in **two ways**: **signing the document** or **signing a digest** of the **document.**

Signing the Document

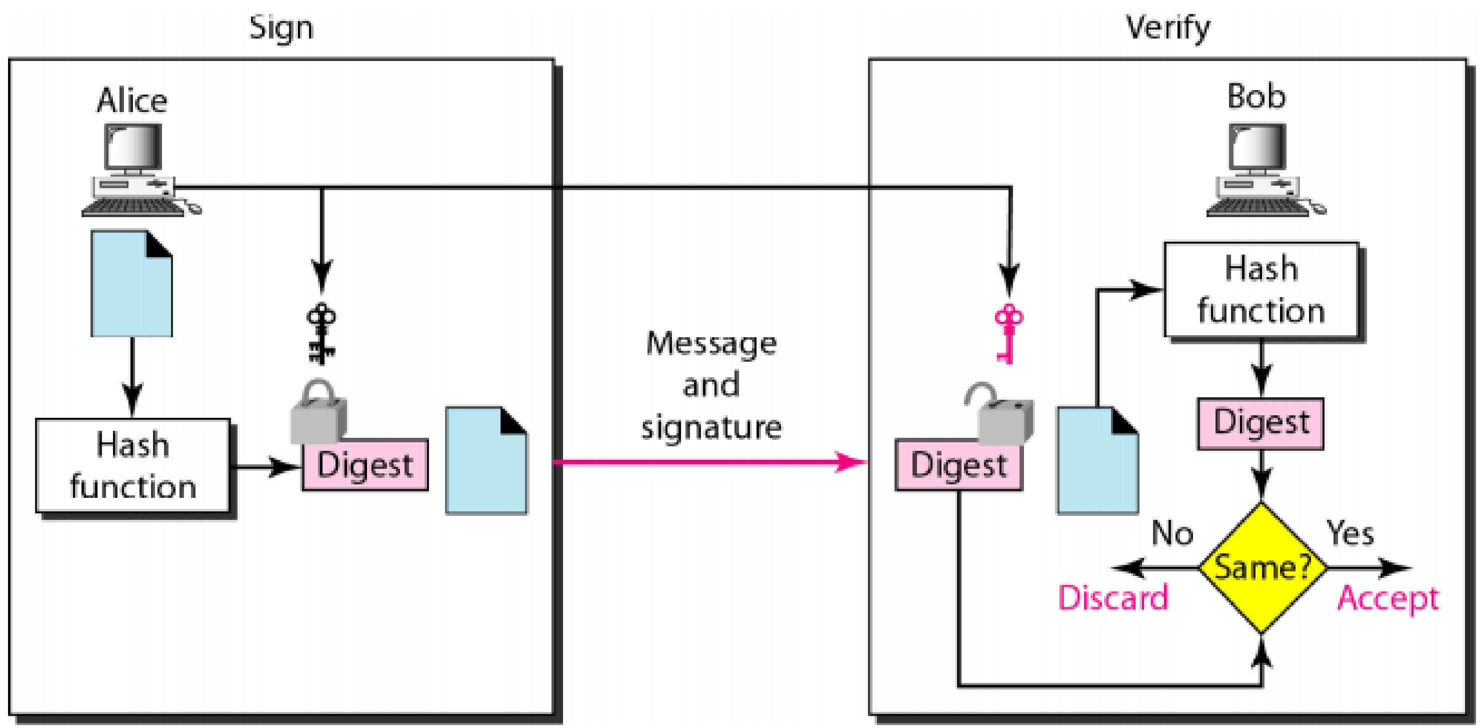
- **Signing a document** is **encrypting it with the private key of the sender**;
- **Verifying the document** is **decrypting it with the public key of the sender**.
- In a **cryptosystem**, we use the **private and public keys** of the **receiver**; in **digital signature,** we use the **private and public key** of the **sender.**



Signing the Digest

- **Public key cryptography is very inefficient** in a cryptosystem **if we are dealing with long messages.**
- In a **digital signature system**, our **messages** are **normally long**, but we have to use **public keys.**
- The **solution** is **not to sign the message itself**; instead, we **sign a digest of the message**.
- The **sender** can **sign** the **message digest**, and the **receiver can verify the message digest.**
- A ***digest*** is made out of the **message** at **Alice's site**.
- The ***digest*** then goes through the **signing** process using **Alice's private key.**
- **Alice** then **sends** the **message** and the **signature** to **Bob**.

Signing the digest in a digital signature



Services by Digital Signature

- A **Digital signature** can provide **three out of the five services** we mentioned for a **security system:**

  1. ***Message Integrity,***
  2. ***Message Authentication,***
  3. ***Nonrepudiation.***

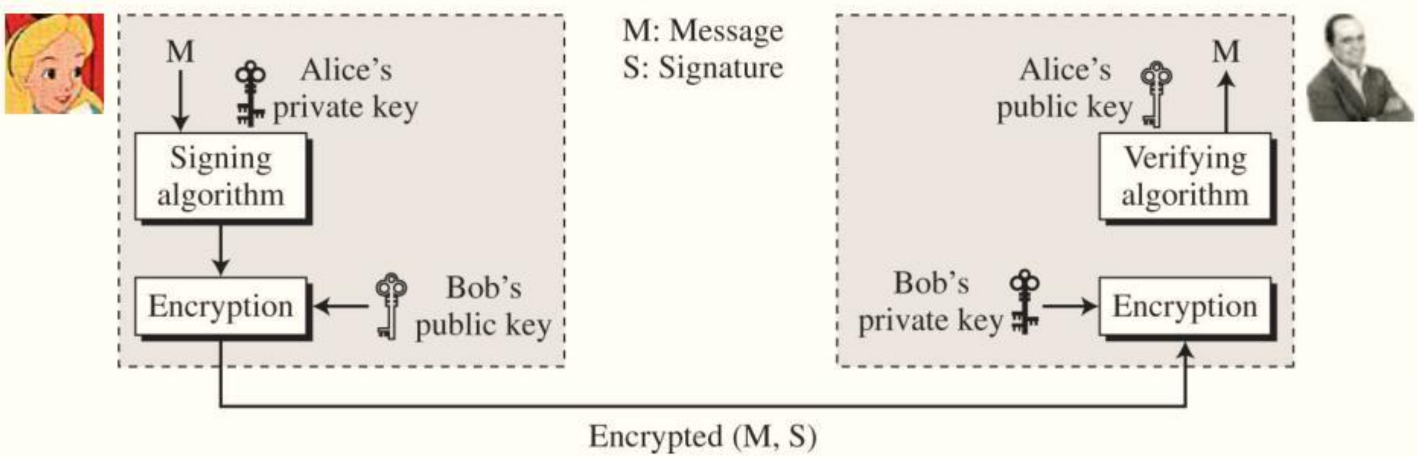
- A **digital signature** scheme **does not provide confidential** communication.
- If **confidentiality** is **required**, the **message** and the **signature** must be **encrypted** using either a **secret-key** or **public-key cryptosystem.**

Digital Signatures and Confidentiality

- Sender:
  - Signs message with <u>sender private key</u>
  - Encrypts message with <u>recipient public key</u>
- Recipient
  - Decrypts message with <u>recipient private key</u>
  - Verifies signature with <u>sender public key</u>

Authentication

Confidentiality



5. ENTITY AUTHENTICATION

- **Entity authentication** is a **technique** designed to let **one party prove** the **identity** of **another party.**
- An **Entity** can be a **person,** a **process,** a **client,** or a **server.**
- The **entity whose identity** needs to be **proved** is called the **claimant;**
- The **party that tries to prove** the **identity** of the **claimant** is called the **verifier.**
- When **Bob** tries to **prove** the **identity** of **Alice, Alice** is the **claimant,** and **Bob** is the **verifier.**
- There are **two methods** for **entity authentication**:

  1. ***Password***
  2. ***Challenge Response***

Passwords

- The **simplest** and the **oldest method** of **entity authentication** is the **password,** something that the **claimant** ***possesses.***
- *A* ***password*** *is used when a* ***user*** *needs to* ***access*** *a* ***system*** *to* use the **system's resources (*log-in*).**
- **Each user** has a **user identification** that is **public** and a **password** that is **private**.
- We can divide this **authentication scheme** into **two** separate **groups:**
  - ***Fixed password***
  - ***One-time password***

Challenge-Response

- In **password authentication**, the **claimant proves** her **identity** by **demonstrating** that she **knows** a **secret,** the **password.**
- However, since the **claimant reveals** this **secret**, the **secret** is **susceptible to interception** by the **adversary.**
- In ***challenge-response authentication***, the **claimant proves** that she ***knows*** *a* ***secret without revealing it.***
- *In other words, the* **claimant does not reveal the secret** to the **verifier**; the **verifier either** has **it** or **finds it.**
- The **challenge** is a **time-varying value** such as a **random number** or a **timestamp** which is **sent by** the **verifier.**
- The **claimant applies a function** to the **challenge** and **sends the result**, called a ***response, to the verifier.***
- *The **response shows** that the **claimant knows*** **the secret**.

Challenge-Response

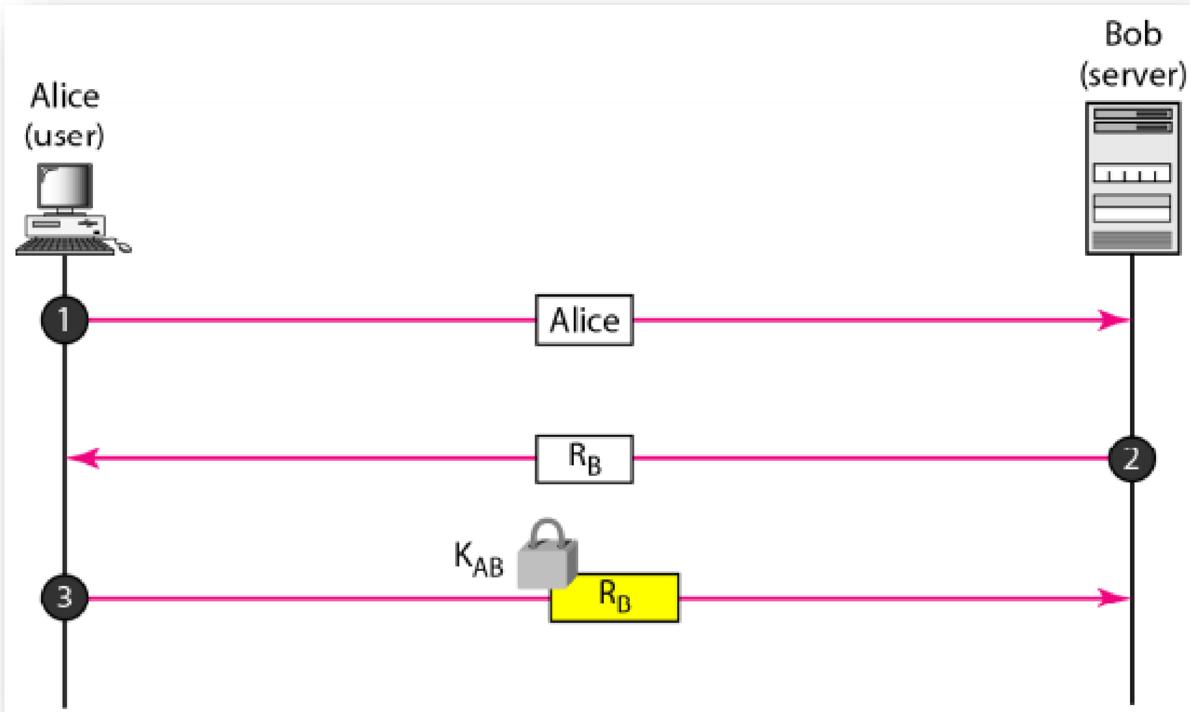
- The **Challenge-response authentication** can be achieved in **four different ways:**

1. **Challenge-response authentication** using using a **Symmetric-Key Cipher**
   a. ***Using a Nonce***
   b. ***Using a Timestamp***
2. **Challenge-response authentication** using a **Keyed-hash function**
3. **Challenge-response authentication** using an **Asymmetric-Key Cipher**
4. **Challenge-response authentication using Digital Signature**

Challenge-Response Using a Symmetric-Key Cipher (Using Nonce)

- In the **first category,** the **challenge-response authentication** is achieved using **symmetric key encryption.**
- The **secret** here is the **shared secret key**, **known by both** the **claimant** and the **verifier.**
- The **function** is the **encrypting algorithm** applied on the **challenge.**
- The **first message** is **not part** of **challenge-response,** it only **informs** the **verifier** that the **claimant** wants to be **challenged.**
- The **second message** is the **challenge which contain the nonce R_B randomly chosen** by the **verifier** to **challenge the claimant**.
- The **claimant encrypts the nonce** using the **shared secret key** known only to the **claimant** and the **verifier** and **sends** the **result** to the **verifier.**

Challenge-Response Using a Symmetric-Key Cipher(Using Nonce)



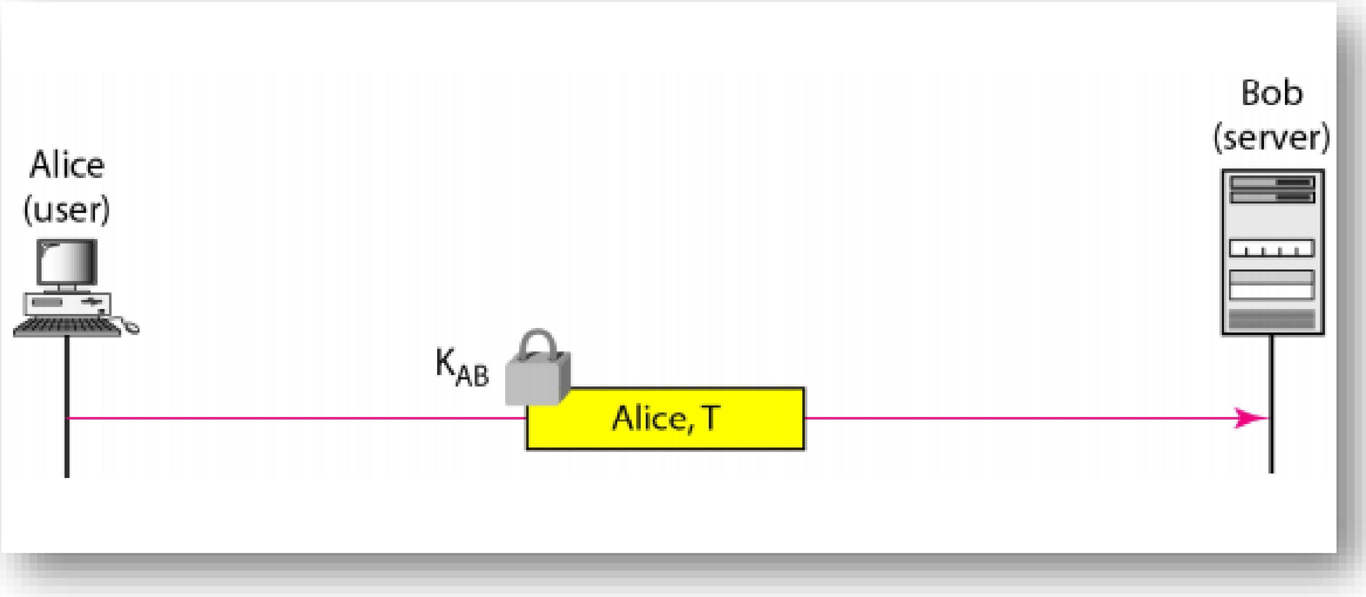
Challenge-Response Using Symmetric-Key Cipher (Using Nonce)

- The **Verifier decrypts** the **message.**
- If the **nonce** obtained from **decryption** is the **same** as the one **sent by the verifier, Alice** is **granted access.**
- **Note** that in this **process,** the **claimant** and the **verifier** need to keep the **symmetric key** used in the process **secret.**
- The **verifier** must also keep the value of the **nonce** for **claimant identification** until the **response** is **returned.**
- **Eve** cannot **replay** the **third message** and **pretend** that it is **by Alice** because **Eve** doesn't know about the **shared secret key** K_{AB}**.**

Challenge-response using a Symmetric-Key Cipher (using Timestamp)

- In this approach, the **time-varying value** is a **timestamp,** which obviously **changes with time.**
- In this approach the **challenge message** is the **current time** sent from the **verifier to the claimant.**
- However, this supposes that the **client and the server clocks are synchronized;** the **claimant knows the current time**.
- This means that there is **no need** for the **challenge message.**
- The **first** and **third messages** can be **combined.**
- The **result** is that **authentication** can be done using **one message**, the **response to an implicit challenge**, the **current time encrypted** using the **shared secret key.**

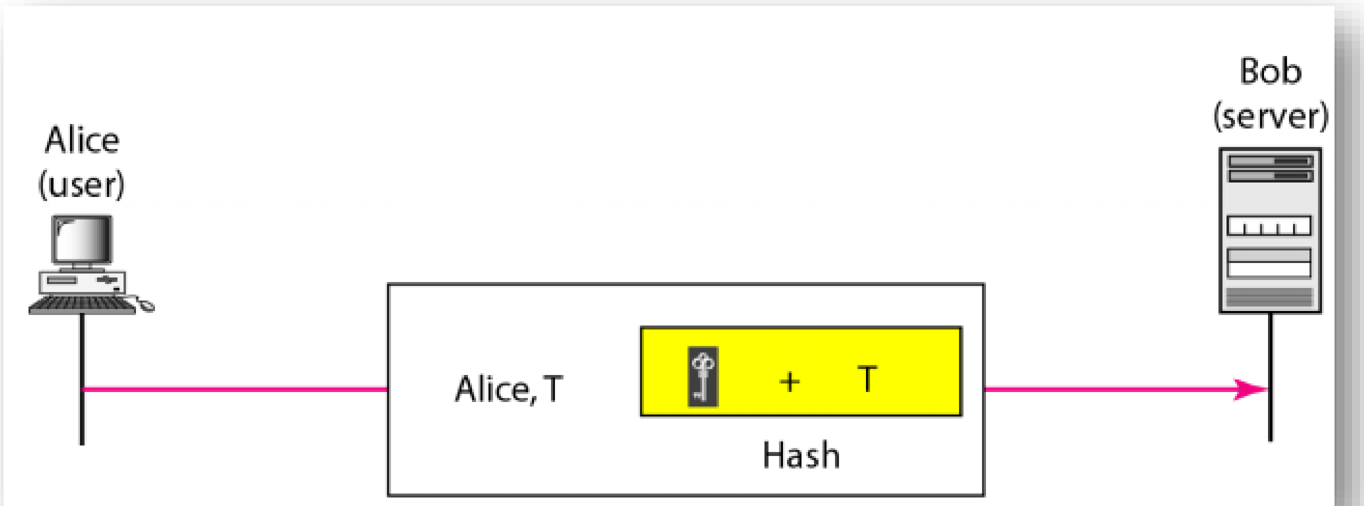
Challenge-response using a Symmetric-Key Cipher (using Timestamp)



Challenge-response using a Keyed-Hash function

- Instead of using **encryption and decryption** for **entity authentication**, we can use a **keyed-hash function (MAC).**
- There are **two advantages** to this scheme.
- **First**, the **encryption/decryption algorithm** is **not exportable to some countries**.
- **Second,** in using a **keyed-hash function**, we can **preserve the integrity of challenge** and **response messages** and at the **same time** use a **secret**, the **key.**
- Note that in this case, the **timestamp is sent both** as **plaintext T** and as **text scrambled by** the **keyed-hash function.**
- When **Bob** receives the **message,** he takes the **plaintext T**, applies the **keyed-hash function**, and then **compares** his **calculation** with what he **received** to determine the **authenticity** of **Alice.**

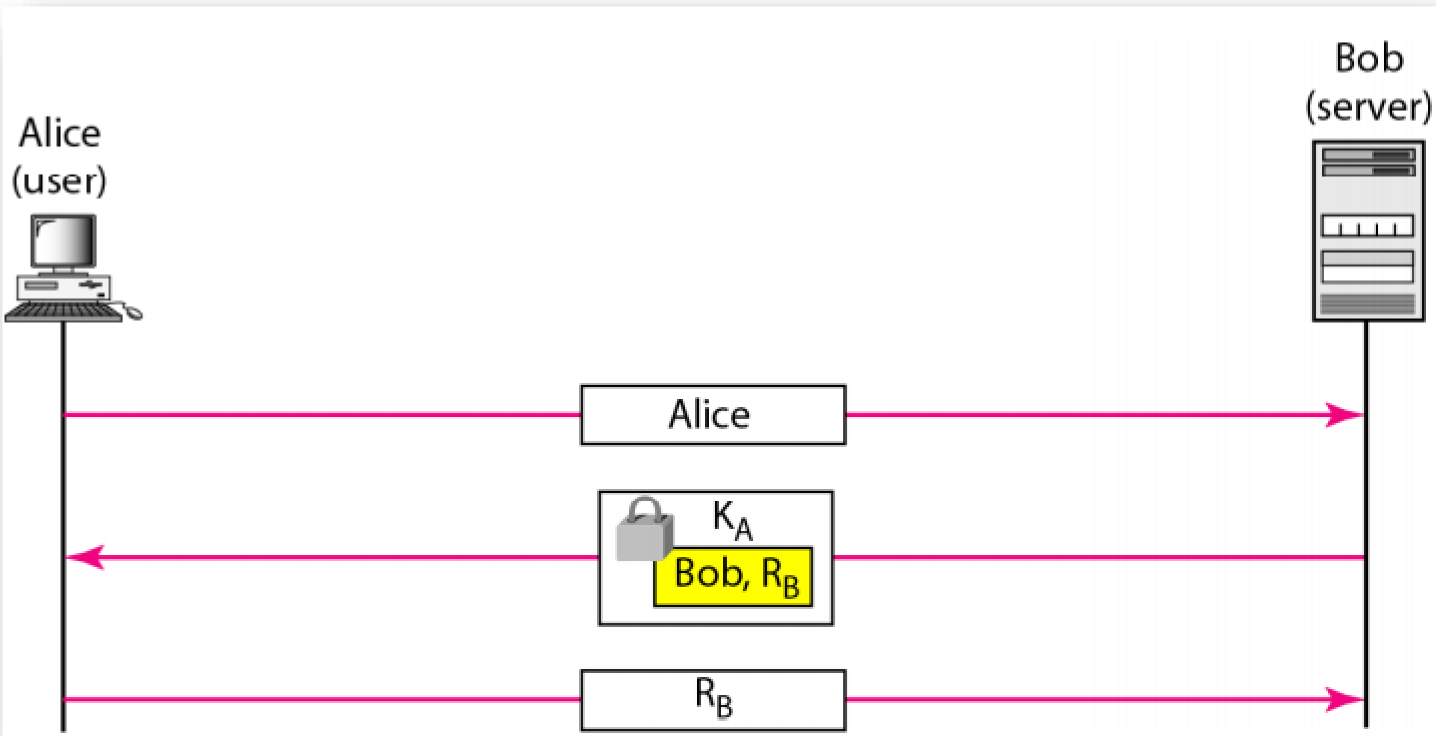
Challenge-response using a Keyed-Hash function



Challenge-response using an Asymmetric-Key Cipher

- In **Asymmetric-key cipher** for **entity authentication** approach the **secret** must be the **private key of the claimant**.
- The **claimant** must show that she owns the **private key** related to the **public key** that is **available** to **everyone.**
- This means that the **verifier** must **encrypt the challenge** using the **public key** of the **claimant**; the **claimant** then **decrypts** the **message** using her **private key.**
- The **response** to the **challenge** is the **decrypted challenge**.
- **Bob encrypts** the **challenge** using **Alice's public key.**
- **Alice decrypts** the **message** with **her private key** and **sends** the **nonce** to **Bob.**

Challenge-response using an Asymmetric-Key Cipher

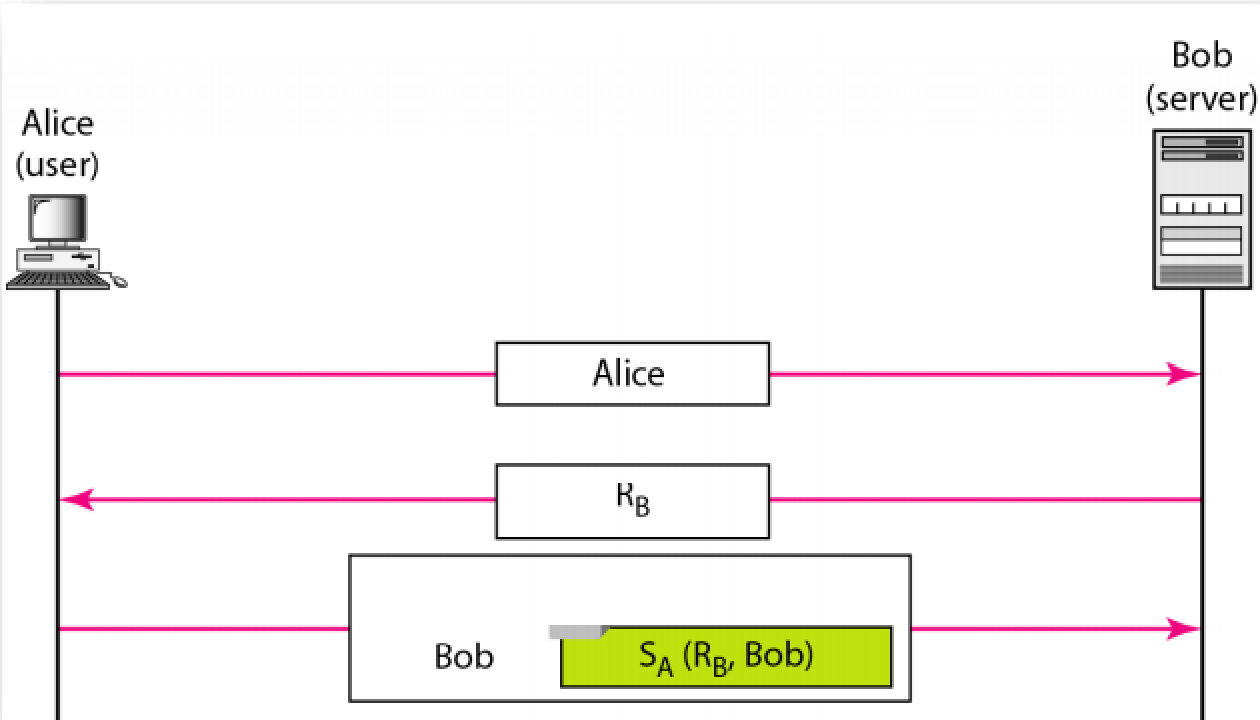


Challenge-response using Digital Signature

- We can use **Digital signature** for **entity authentication.**
- In this method, we let the **claimant use her private key for signing** instead of using it for **decryption.**
- There are **four steps**:

1. ***Initialization.*** The **claimant** tells the **verifier** it wishes **to be authenticated**.
2. ***Challenge***. The **verifier generates a challenge** and issues it to the **claimant.**
3. ***Response.*** **The claimant signs** the **challenge (using his private key)** and returns it to the **verifier.**
4. ***Verification***. The **verifier** verifies that the **signed version** of the **challenge** matches the **issued challenge**(using claimant public key) and **grants access** to the **client.**

Challenge-response using Digital Signature



KEY MANAGEMENT

- How **Secret keys** in **Symmetric-key cryptography** and how **Public keys** in **Asymmetric-key cryptography** are **distributed** and maintained.
- Thus **two issues** are:

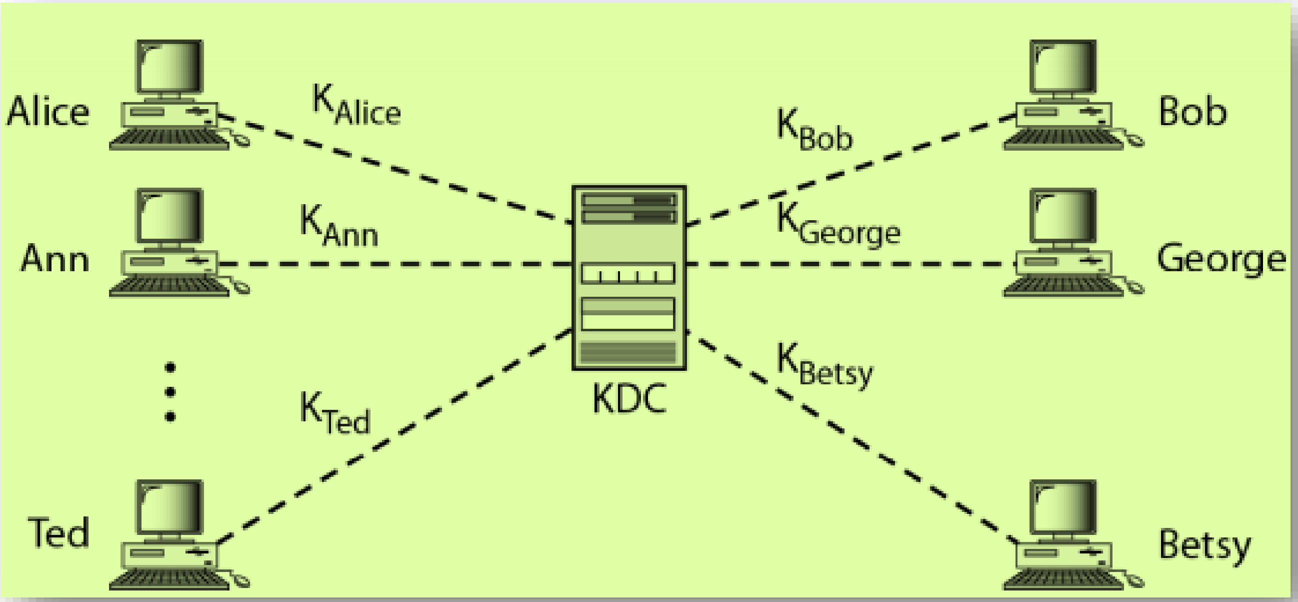
  1. ***Distribution of Symmetric keys***
  2. ***Distribution of Asymmetric keys***

Symmetric-Key Distribution

- **Symmetric key cryptography**, however, needs a **shared secret key** between **two parties.**
- If **Alice** needs to exchange **confidential messages** with ***N people***, *she **needs** **N different*** **keys.**
- What if ***N people*** *need to communicate with one another?*
- *A **total** of* ***N(N - 1)/2*** ***keys** are* needed.
- **Each person** needs to have ***N - 1 keys*** *to communicate with each of the other* people, but because the keys are shared, we need only ***N(N - 1)/2.***
- If **Alice** and **Bob** want to communicate, they need to somehow **exchange** a **secret key**; if **Alice** wants to communicate with **1 million people**, how can she exchange **1 million keys** with **1 million people?**
- Using the **Internet** is definitely **not a secure method**.

Key Distribution Centre: KDC

- A **practical solution** is the use of a **trusted party**, referred to as a **key distribution centre (KDC).**
- To **reduce** the **number of keys**, **each person establishes** a **shared secret key** with the **KDC.**



Key Distribution Centre: KDC

- A **secret key** is **established between KDC** and **each member**.
- **Alice** has a **secret key with KDC**, which we refer to as K_{Alice}.
- **Bob** has a **secret key with KDC**, which we refer K_{Bob} and so on.
- How can **Alice send** a **confidential message** to **Bob?**
- The **process** is as follows:

1. **Alice** sends a request to **KDC,** stating that **she** needs a **session (temporary) secret key** between **herself and Bob**.
2. **KDC** informs **Bob** of **Alice's request**.
3. If **Bob agrees**, a **session key** is **created** between the two.
4. **KDC** share the **Session key** with **Alice.**

After **communication is terminated**, the **session key is no longer valid**.

Setting up a One-time Session key using a KDC

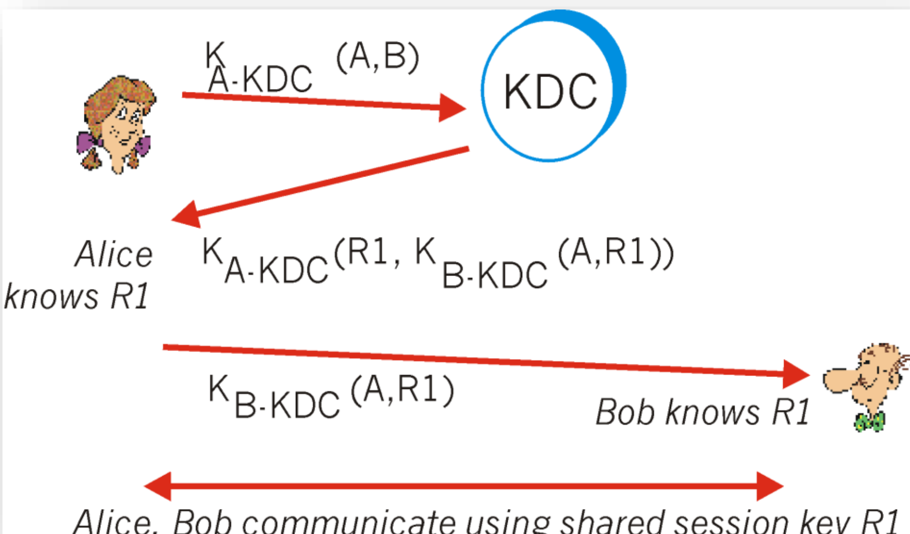
- Suppose that **Alice** and **Bob** are **users** of the **KDC**; they only know **their individual key,** $K_{A\text{-}KDC}$ and $K_{B\text{-}KDC}$, respectively, for communicating securely with the **KDC.**
- **Alice** takes the **first step**, and **sends** an **encrypted message** using key $K_{A\text{-}KDC}$ to the **KDC** saying she **(*A*)** wants to communicate with **Bob** **(*B*)** as $K_{A\text{-}KDC}(A,B)$.
- The **KDC**, knowing $K_{A\text{-}KDC}$, **decrypts** $K_{A\text{-}KDC}(A,B)$, *in this way* **KDC** authenticated **Alice.**
- The **KDC** then **generates** a **random number, *R1***, which is the **shared key** value that **Alice** and **Bob** will use to perform **symmetric encryption** when they communicate with each other.
- This **key** is referred to as a **one-time session key** , as **Alice** and **Bob** will use this **key** for only this **one session** that they are currently setting up.

Setting up a One-time Session key using a KDC

- The **KDC** now needs to inform **Alice** and **Bob** of the value of ***R1***.
- The **KDC** thus **sends back** an **encrypted message** to **Alice** containing the following:
- ***R1***: the **one-time session key** that **Alice** and **Bob** will use to communicate.
- $K_{B\text{-}KDC}(A,R1)$: A pair of values: **A**, and ***R1***, encrypted by the **KDC** using **Bob's key,** $K_{B\text{-}KDC}$.
- These items are **put** into a **message** and **encrypted** using **Alice's shared key** and transmitted to **Alice** from **KDC** as $K_{A\text{-}KDC}(R1 , K_{B\text{-}KDC}(A,R1))$.
- **Alice** receives the message from the **KDC**, **extracts** ***R1*** from the message and saves it.
- **Alice** now knows the **one-time session key**, ***R1***.
- **Alice** also extracts $K_{B\text{-}KDC}(A,R1)$ and forwards this to **Bob.**
- **Bob** **decrypts** the received message, $K_{B\text{-}KDC}(A,R1)$, using $K_{B\text{-}KDC}$ and extracts **A** and ***R1***.

Setting up a One-time Session key using a KDC

- **Bob** now knows the **one-time session key, *R1*,** and the person with whom he is sharing this key, ***A***.
- First **Bob** authenticate **Alice** using ***R1*** before proceeding any further.



$K_{A\text{-}KDC}$:Shared secret key between Alice and KDC

$K_{B\text{-}KDC}$:Shared secret key between BOB and KDC

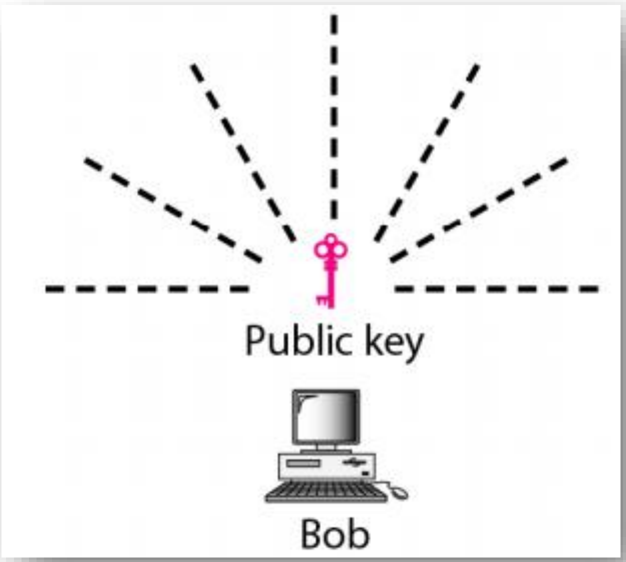
R1: Session Key between Alice and Bob.

Public-Key Distribution

- In **Asymmetric-key cryptography**, people do not need to know a **symmetric shared key.**
- If **Alice** wants to send a message to **Bob,** she only needs to know **Bob's public key**, which is **open to the public** and **available to everyone.**
- If **Bob** needs to send a message to **Alice,** he only needs to know **Alice's public key,** which is also **known to everyone.**
- In **public-key cryptography**, everyone **shields** a **private key** and **advertises a public key.**
- **Public keys**, like **secret keys**, need to be **distributed** to be useful.

Public Announcement

- The **naive approach** is to **announce public keys** publicly.
- **Bob** can put his **public key** on his **website** or **announce** it in a **local or national newspaper.**
- When **Alice** needs to send a **confidential message** to **Bob,** she can obtain **Bob's public key** from **his site** or from the **newspaper,** or she can even **send a message** to ask for it.



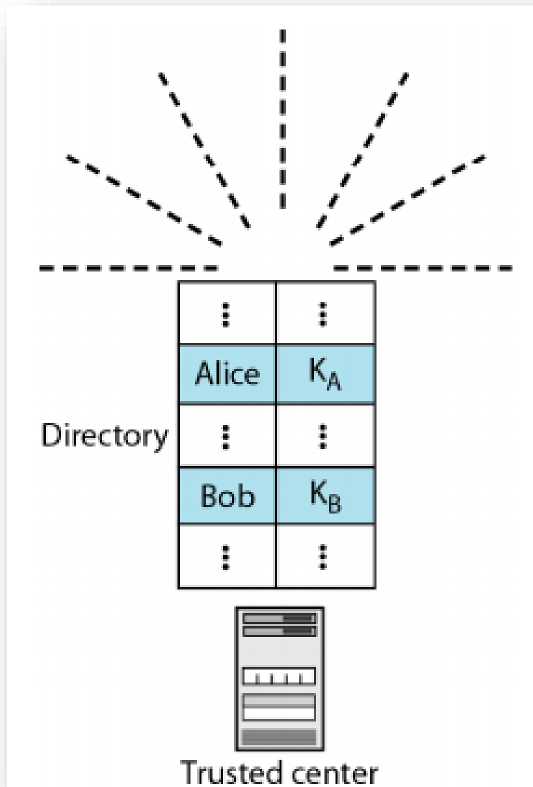
Public Announcement

- This approach, however, is **not secure**; it is subject to **forgery**.
- For **example**, **Eve** could make such a **public announcement.**
- Before **Bob** can react, **damage** could be done.
- **Eve** can fool **Alice** into **sending her a message** that is **intended** for **Bob.**
- **Eve** could also **sign a document** with a corresponding **forged private key** and make **everyone believe** it was **signed by Bob.**
- The **approach** is also **vulnerable** if **Alice** directly requests **Bob's public key**.
- **Eve** can **intercept Bob's response** and **substitute** her own **forged public key** for **Bob's public key.**

Trusted Center

- A **more secure approach** is to have a **trusted center** retain a **directory of public keys.**
- The **directory,** like the one used in a **telephone system**, is **dynamically updated**.
- Each user can **select a private/public key**, keep the **private key**, and deliver the **public key** for **insertion** into the **directory**.
- The **center requires** that each **user register in the center** and **prove** his or her **identity.**
- The **directory** can be **publicly advertised** by the **trusted center**.
- The **center** can **also respond to any inquiry** about a **public key.**

Trusted Center



Controlled Trusted Center

- A **higher level of security** can be **achieved** if there are **added controls** on the **distribution** of the **public key.**
- The **public-key announcements** can **include a timestamp** and be **signed by an authority** to **prevent interception** and **modification of the response**.
- If **Alice** needs to know **Bob's public key,** she can **send a request** to the **center** including **Bob's name** and a **timestamp.**
- The **center responds** with **Bob's public key,** the **original request**, and the **timestamp signed with the private key of the center**.
- **Alice** uses the **public key of the center**, known by all, to **decrypt the message** and **extract Bob's public key.**

Controlled Trusted Center