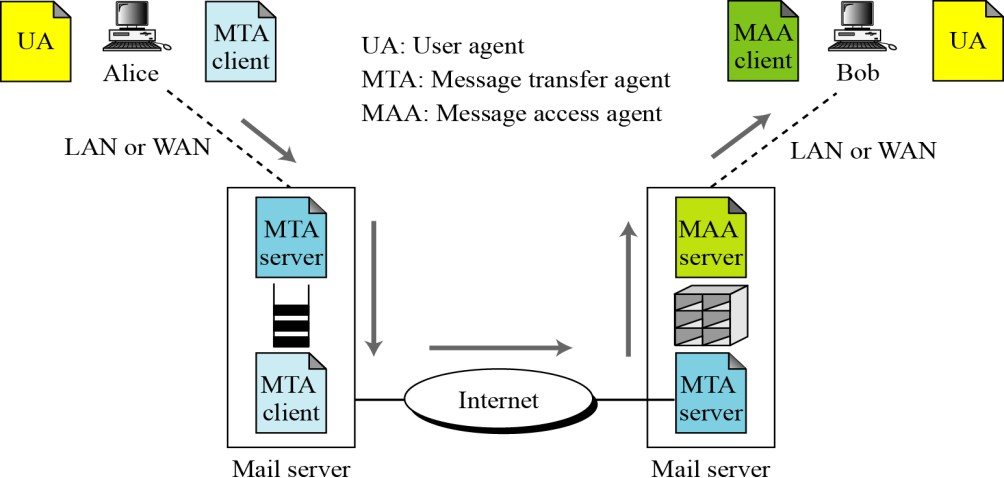
# UNIT 5

**PART1: Security at application layer**

* 1. **E-mail Architecture**

X

## PGP

* + - Pretty Good Privacy (PGP) is a protocol used to secure E-mail applications. It is developed by Phil Zimmermann. It is open source software.
    - PGP provides a **confidentiality** and **authentication** services that can be used for

**electronic mail** and **file storage** applications

* + - It is available on Unix, PC, Macintosh and Amiga operating systems.

**Note**: In e-mail security, the encryption/decryption is done using a symmetric-key algorithm, but the secret key to decrypt the message is encrypted with the public key of the receiver and is sent with the message

**PGP services/operations :**

The operation of PGP consists of four services:

* + - **Authentication** – provided by using digital signature.

#### Integrity - ”

* + - **Confidentiality** – provided by using symmetric block encryption algorithms.
    - **Compression** – provided by using compressed methods.
    - **Code conversion**– provided by using radix64 encoding scheme.
    - **Segmentation –** Automatically done. No specific method used.
  1. **PGP services/operations**

SENDER: Alice RECEIVER: Bob

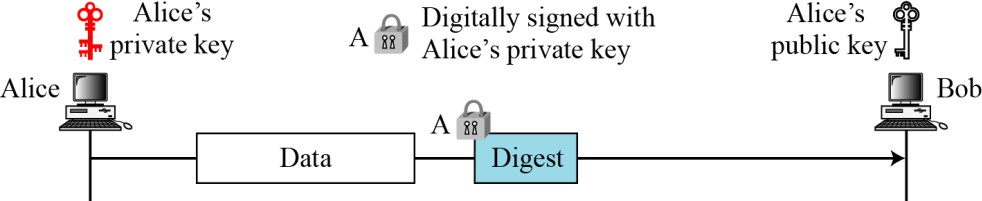
#### Authentication and Integrity –

To provide Authentication and Message Integrity

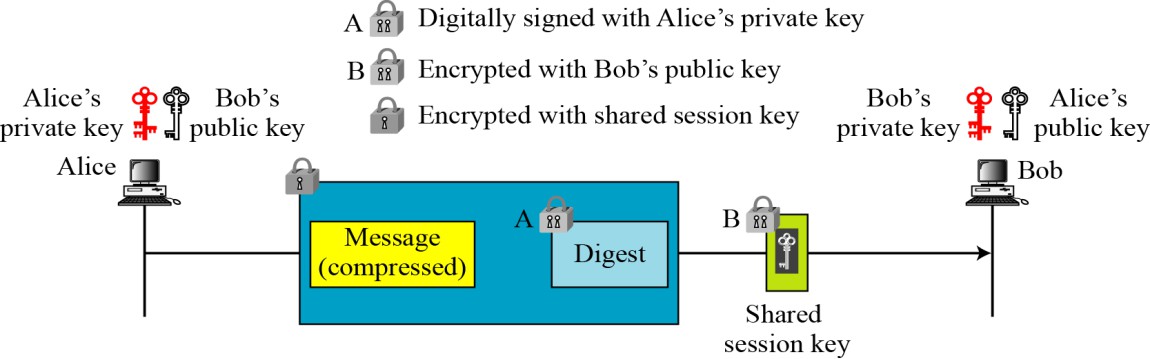
* + Alice creates a digest of the message and signs it with her private-key

Signing means digest is encrypted using senders private-key.

* + When Bob receives the message he verifies the message by using Alice’s public-key. Here, two keys are needed – Alice’s private- key and Bob’s public-key

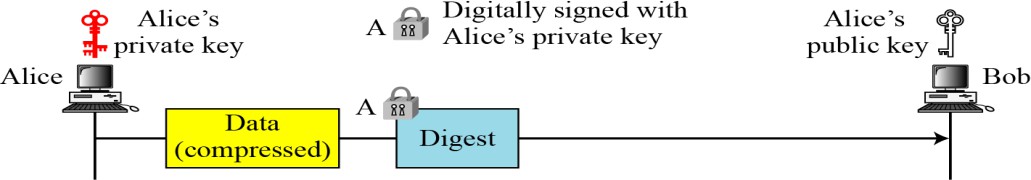


1. **Confidentiality** – provided by using symmetric block encryption algorithms.



#### Compression

* + After encrypting the digest, Alice compresses the Data using compression method. Then transmits encrypted digest and compressed data to Bob.
  + There is no security benefit. But, it eases the traffic since the packet size is reduced.



* + PGP compresses the message after applying the signature but before encryption. This has the benefit of saving space both for e-mail transmission and for file storage.
  + Z for compression and Z–1 for decompression
  + The signature is generated before compression for two reasons:
    - It is preferable to sign an uncompressed message so that one can store only the uncompressed message together with the signature for future verification.
    - If you generate signature after compression then there is a need recompression for message verification, PGP’s compression algorithm presents a difficulty.
  + Message encryption is applied after compression to strengthen cryptographic security. Therefore cryptanalysis is more difficult.
  + The compression algorithm used here is ZIP Algorithm

#### Code conversion

* + All the characters in e-mail message are converted into ASCII characters
  + To translate other characters not in ASCII set PGP uses **Radix-64** conversion.

#### Segmentation

* + E-mail facilities often are restricted to a maximum length. To accommodate this, PGP automatically subdivides a message that is too large into segments that are small enough to send via e-mail.
  + The segmentation is done after all of the other processing, including the radix-64 conversion.

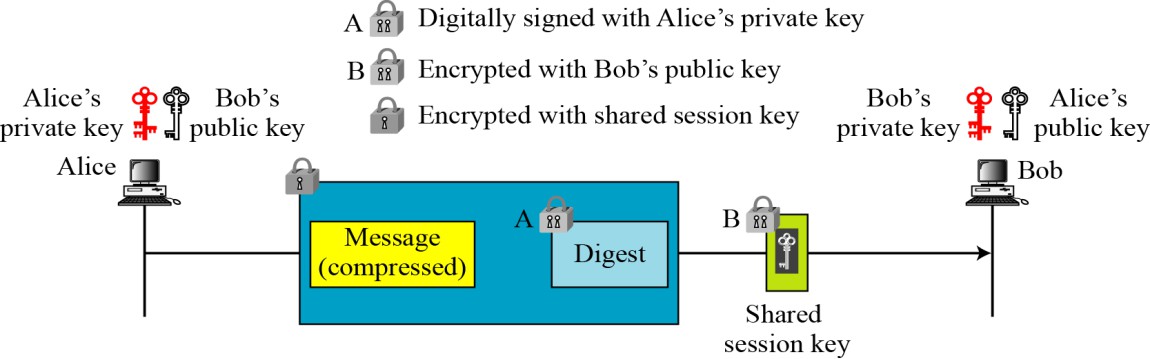
x

* 1. **PGP Working**

##### Confidentiality with One-Time Session Key

* Pretty Good Privacy uses a variation of the public key system. In this system, each user has an encryption key that is publicly known and a private key that is known only to that user.
* In general we encrypt a message that we want to send to someone else using their public key. When they receive it, they decrypt it using their private key.
* Since encrypting an entire message can be time-consuming, PGP uses a faster encryption algorithm to encrypt the message and then uses the public key to encrypt the session-key that was used to encrypt the entire message. Both the encrypted message and the session-key are sent to the receiver who first uses the receiver's private key to decrypt the session-key and then uses that session-key to decrypt the message.

##### A confidential message



Suppose if Alice wants to send an e-mail message, then she

1. Alice creates an E-mail message first.
2. For that message, she generates **Message Digest/ Digest** using the hash algorithm.
3. She signs the **Digest** using her private key, called Digital Signature. i.e encrypts the **Digest**

using private key.

1. She creates a **Session-key** and then encrypts it using Bob’s public-key. This **Session-key** is used to decrypt the message at receiver site.
2. She compresses the original message using compressing algorithm.
3. She appends compressed message and signed digest, then encrypts it using Bob’s public-key and sends it along with the encrypted **Session-key** to the Bob.

#### The algorithms used in PGP are…….

* 1. Public-key algorithms ---
     + For signing the digest (digital signature)
       - Ex: DSS, RSA (for signing only)
     + For encrypting the compressed message and signed digest using receiver’s public-key
       - Ex: RSA (for Encryption only), RSA (Encryption & signing )
  2. Symmetric-key algorithms
     + For encrypting the session-key created by the sender of the message
       - Ex: IDEA, Triple DES
  3. Hash algorithms
     + For the digest creation
       - Ex: SHA-1, MD5
  4. Compression methods
     + For compression
       - Ex: ZIP, ZLIP

|  |  |
| --- | --- |
|  |  |
|  |  |

X

* 1. **PGP Key Rings**

Each PGP user has a pair of key rings:

#### Example:

* Public-key ring contains all the public-keys of other PGP users known to this user, indexed by key ID
* Private-key ring contains the public/private key pair(s) for this user, indexed by key ID & encrypted keyed from a hashed passphrase

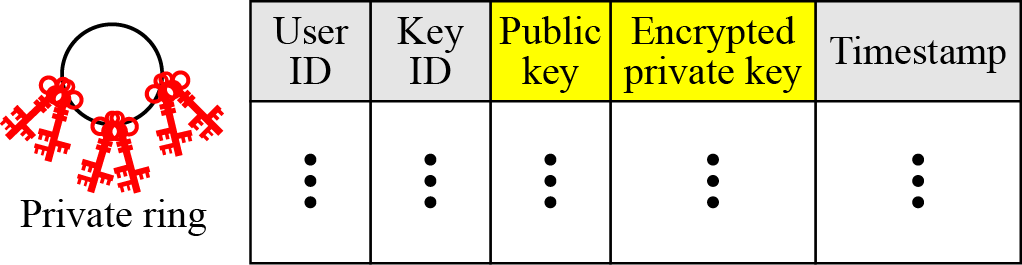
Alice has several pairs of private/public keys belonging to her and public keys belonging to other people.

**Private-key ring:**

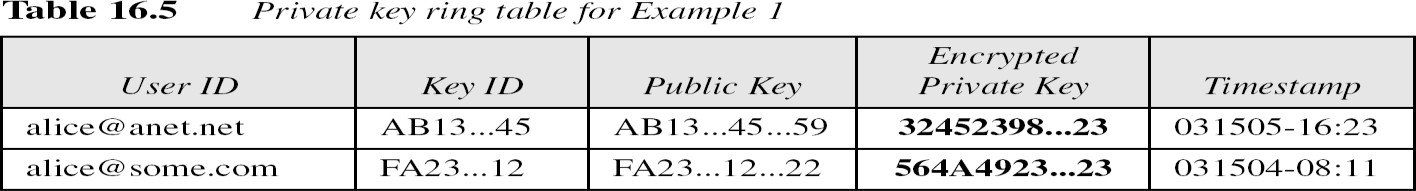
* + We can view the ring as a table in which each row represents one of the public/private key pairs owned by this user.
  + Each row contains the entries:
    - **User ID**: It is the e-mail address of the user. Apart from e-mail address it may store alias names for each key pair. (e.g., [stallings@acm.org](mailto:stallings@acm.org)).
    - **Key ID**: The least significant 64 bits of the public key. i.e (public-key mod 264). It is used to uniquely identify a public-key among the use’s public-keys.

The **Key ID** is sent along with the message to the recipient, to use the public-key of the sender from public key ring of the recipient.

Sending the public key along with the message is tough task.

* + - **Public key**: It lists the public-key of a particular private/public key pair.
    - **Encrypted Private key**: Holds the encrypted value of the private-key in the private/public key pair.
    - **Timestamp**: The date and time of the key pair was generation.

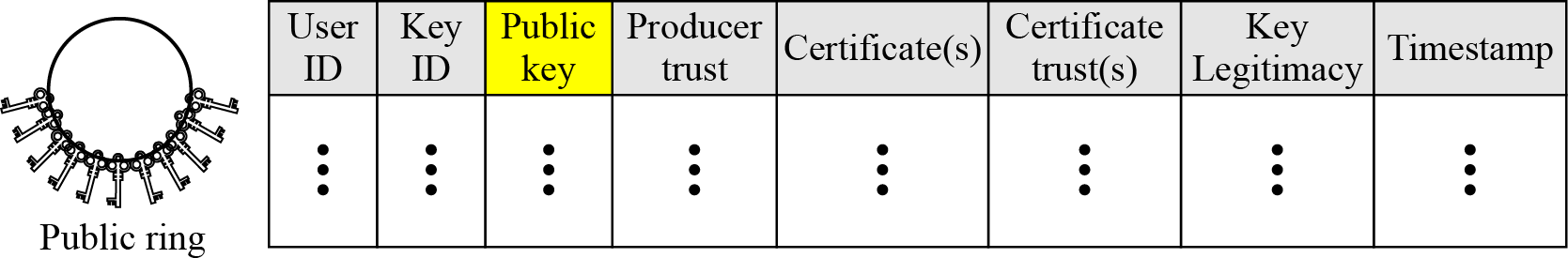
#### Example:



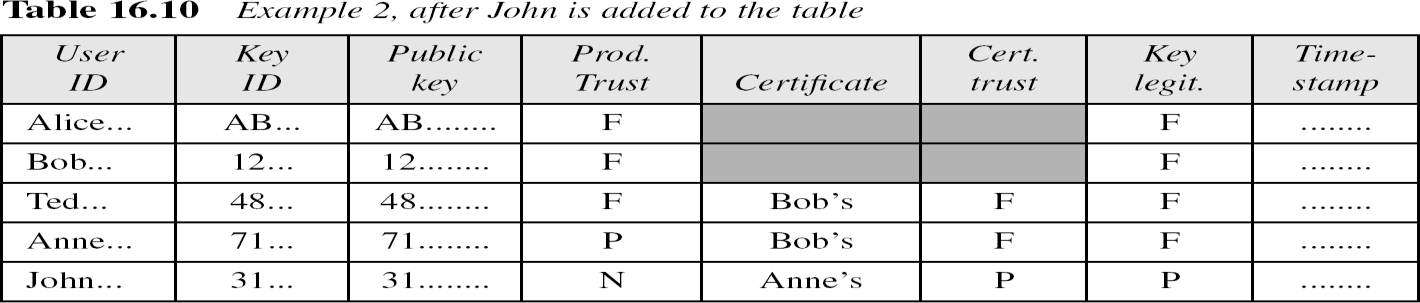
**Public-key ring:**

This ring is used to store public keys of other users that are known to this user.

* **User ID**: Identifies the owner of this key. Multiple user IDs may be associated with a single public key.
* **Key ID**: The first 64 bits of the public key.
* **Public Key**: The public key for the entity.
* **Producer Trust**: Defines producer level of trust. It can be one of 3 possible values: none partial or full.
* **Certificate(s)**: This column holds the certificate or certificate(s) signed by other entities for this entity. One user ID may have than one certificate.
* **Certificate trust(s):** This column holds the value of user’s trust on the certificate introducer to this entity.
* **Key Legitimacy:** This value this calculated based on the value of the certificate trust and predefined weight for each certificate trust.
* **Timestamp**: The date/time of the column creation.

The public key can be indexed by either User ID or Key ID.

Example:



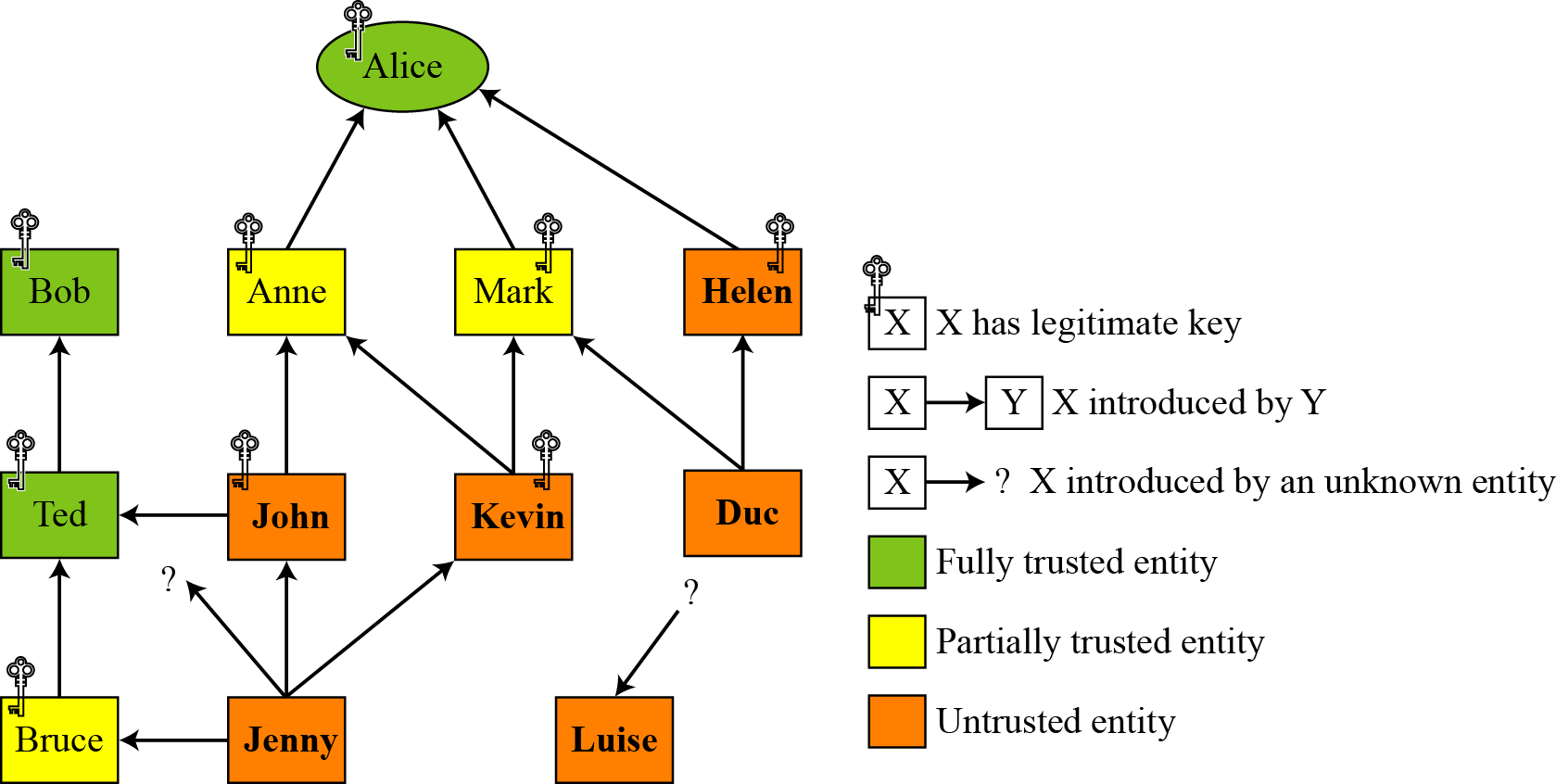
X

#### Trusted Model in PGP

* **PGP** is widely used for exchanging secure e -mail over Internet. **Trust** in **PGP** is achieved using the web of **trust model**. The underlying idea of this **model** is that you accept the public key of a **PGP** user if it has been signed by one or more other trustworthy **PGP** users.
* We can create a trust model for any user based on a public-key ring table. The trust model may change with any changes in the public-key ring table.
* When a new public key is inserted on public-key ring, it assigns the trust value of the key’s owner given by the user. The trust value for each signature attached at the key is given to the OwnerTrust value for the owner of the signature;

If the owner for the signature is not in the key ring, it is given to a unknown user value.

**Example:** Consider web of trust model of the user Alice.



* + This is the trust model at Alice site.
  + Each square represents one user. And each user maintains their own trust model based on their public-key table.
  + The trust model of Alice is generated based on the public-key ring. It consists of
* 3 entities in Alice's ring with **full trust** - **Alice herself, Bob, and Ted**
* 3 entities with **partial trust** - Anne, Mark and Bruce **and** 6 entities with **no trust**.
* Nine entities have a **legitimate key**.
* three entities do not have any legitimate keys with Alice
  + Alice can **encrypt** a message to any one of these entities or **verify** a signature received from one of these entities.
  + Bob, Anne, and Mark have made their keys legitimate by sending their keys by e-mail and verifying their fingerprints by phone.
  + Helen, has sent a certificate from a CA because she is not trusted by Alice and verification on the phone is not possible.
  + Ted is fully trusted, he has given Alice a certificate signed by Bob.
  + John has sent Alice two certificates, one signed by Ted and one by Anne.
  + Kevin has sent two certificates to Alice, one signed by Anne and one by Mark. Each of these certificates gives Kavin half a point of legitimacy; therefore, Kavin's key is legitimate.
  + Duc has sent two certificates to Alice, one signed by Mark and other by Helen. Since Mark is half-trusted and Helen is not trusted, Duc does not have a legitimate key.
  + Jenny has sent four certificates, one signed by half-trusted entity, two by un-trusted entities, and one by an unknown entity. Jenny does not have enough points to make her key Legitimate.
  + Luise has sent one certificate signed by an unknown entity.

Note: Alice may Luise's name in the table in case future certificates for Luise arrive.

X

* 1. **PGP Certificates**

*In PGP, there is no need for CAs; anyone in the ring can sign a certificate for anyone else in the ring.*

For the same public key one user can get any number of certificates from other users.

**Example**: Alice can obtain certificates for the public-key K1, from Anne, Joe, Roy and Shan

##### The entire operation of PGP is based on

* + - *Introducer trust level*
    - *Certificate trust level and*
    - *Legitimacy*

#### Introducer trust

Every user can issue any number of public-key certificates to other users. He/she is called an introducer. PGP allows different trust levels to maintain in key rings**: none, partial and full.**

The **i**ntroducer trust level specifies the trust levels issued by the introducer for other people in the ring.

##### Example:

Alice may fully trust Bob, partially trust Anne and not trust John. There is no specific mechanism to determine trust level in PGP.

#### Certificate trust level

When user receives certificate(s) from others, assigns a level of trust (i.e one of the three: none, partial or full).

#### Example:

Assume that Alice fully trust **Bob,** partially trusts **Anne** and **Janette,** and has no trust in

#### John.

The following scenarios can happen…

* 1. Bob issues two certificates --- for---Linda and Lesley. So, Alice receives and stores public-key certificates of Linda and Lesley, and assigns trust level as “**full**”. Since, Alice has **full** trust in **Bob.**
  2. Anne issues a certificate --- for --- John. So, Alice stores certificate of John and assigns trust level as “**partial**”. Since, Alice has **partial** trust in **Anne.**
  3. Janette issues two certificates --- for---John and Lee. So, Alice stores certificate of John and Lee, and assigns trust level as “**partial**”. Since, Alice has **partial** trust in **Janette.**

Now, John has two certificates – issued by --

* 1. John issues a certificate --- for ---Liz. Therefore, Alice can discard or keep certificate of Liz. If stores, assigns trust level as “**none**”. Since, Alice has **no** trust in **John.**

#### Key Legitimacy

PGP provides procedure for determining key legitimacy. Key legitimacy is required to trust the entity for sending messages. Sender can send messages only when the receiver is having either one fully trusted certificate or two partially trusted certificates.

Key legitimacy level = weighted trusted level Weights of certificates are….

1. Certificate with No trust – weight = 0
2. Certificate with partial trust – weight = 1/2
3. Certificate with full trust – weight = 1

***Key-revocation:***

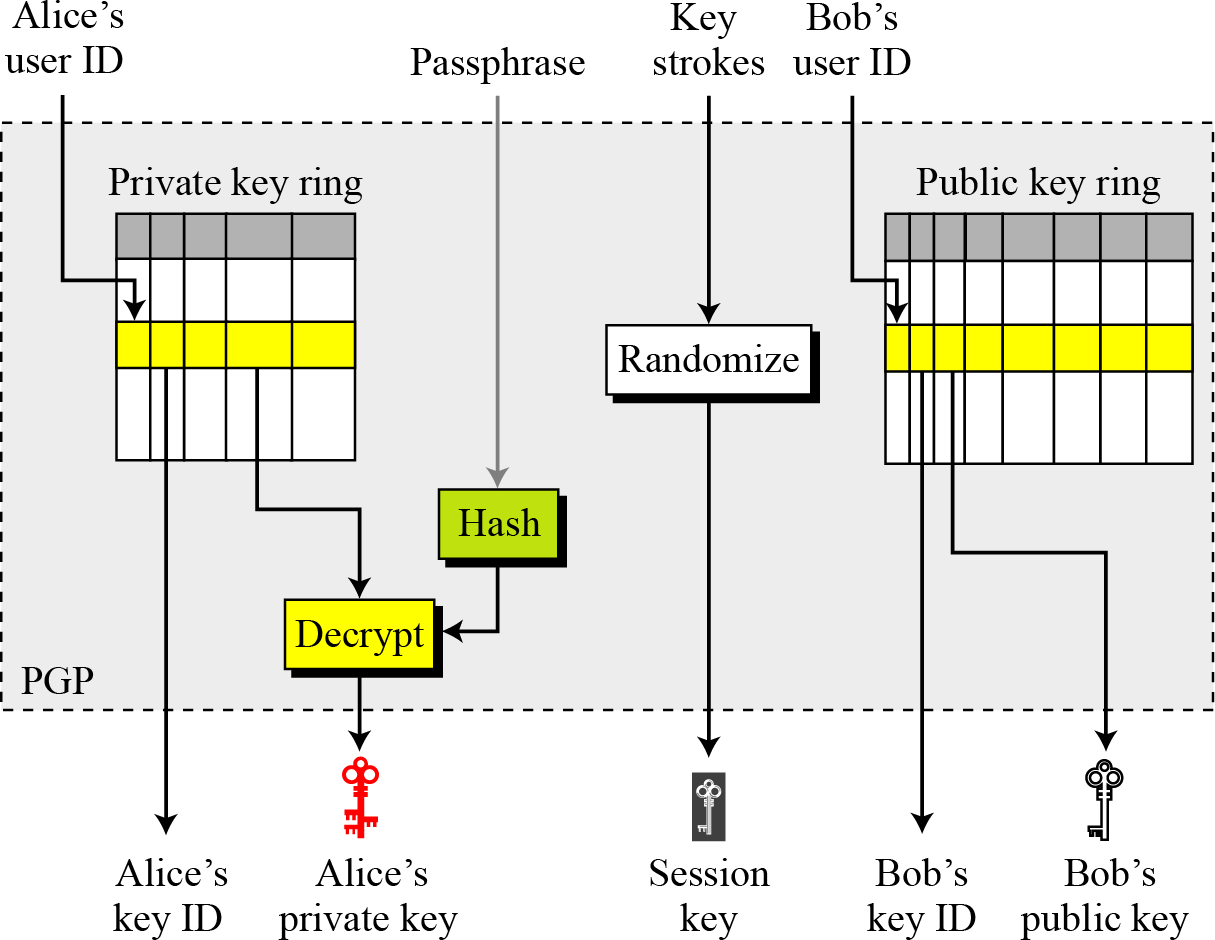
Key-revocation is happened

* + If the owner of the key feels that the key is compromised (stolen, for example) or
  + If the key is too old

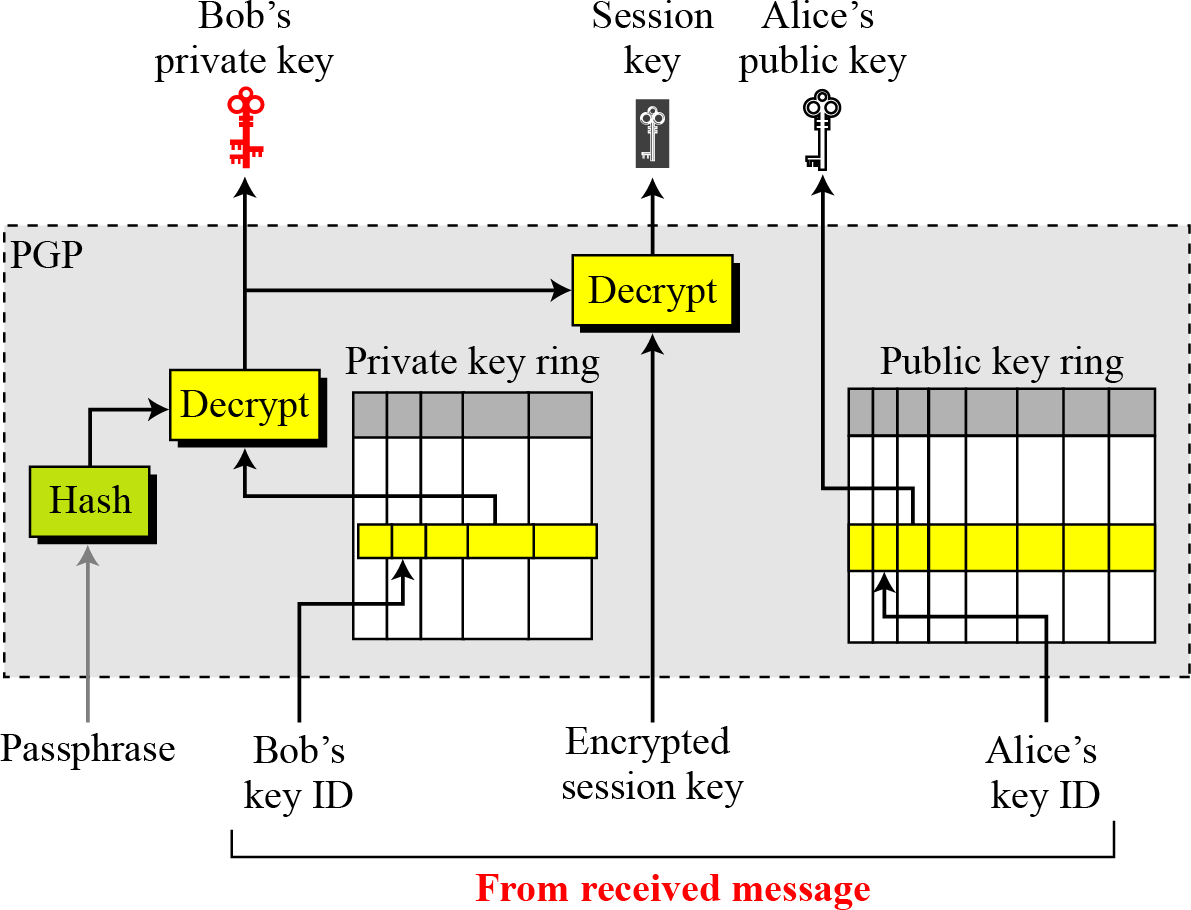
X

* 1. **Extracting Information from Rings**

##### Extracting information at the sender site



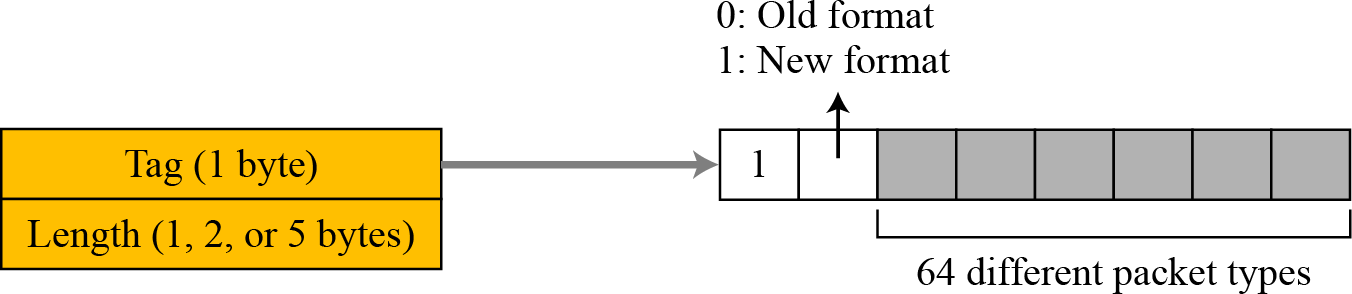
***Extracting information at the receiver site***



X

* 1. **PGP Packets**
     + A message in PGP packets consists of one or more packet.
     + During the evolution of PGP, the format and number of packets types have changed.
     + PGP has a generic header that applies to every packet.
     + The generic header in the most recent version has only two fields.

**Format of packet header**



* + - **Tag**:
      * Tag is an 8-bit flag.
      * The first bit (most significantly) is 1.
      * The second bit is 1 if we are using latest version.
      * The remaining six bits can define up to 64 different packet types.

#### Some commonly used packet types

|  |  |
| --- | --- |
| Value | Packet type |
| 1 | Session key packet encrypted using a public key |
| 2 | Signature packet |
| 5 | Private-key packet |
| 6 | Public-key packet |
| 8 | Compressed data packet |
| 9 | Data packet encrypted with a secret key |

|  |  |
| --- | --- |
| 11 | Literal data packet |
| 13 | User ID packet |

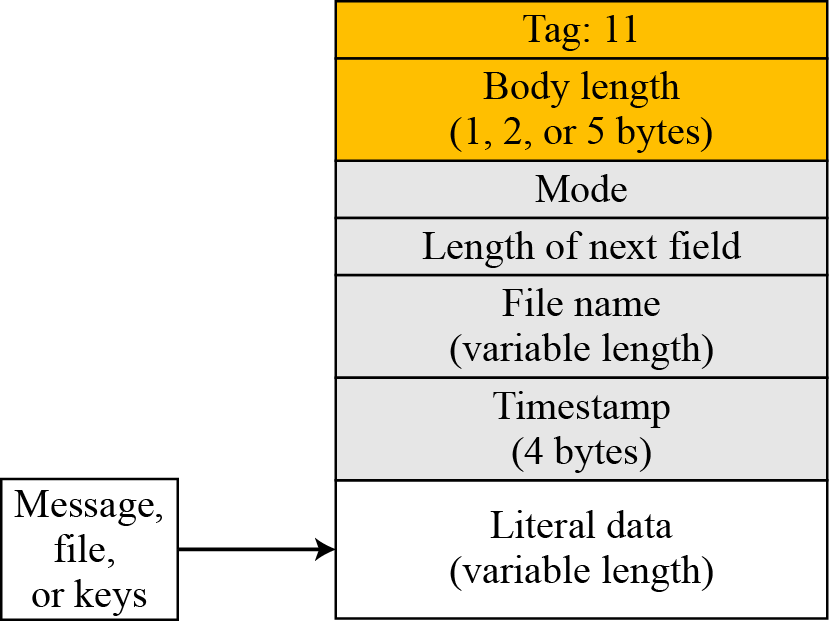
* + - **Length**:
      * Length defines the length of the entire packet in bytes.
      * The size of this field is variable; it can be 1,2 or 5 bytes.
      * The receiver can determine the number of bytes of the length field by looking at the value of the byte immediately following the tag field.

**Types of Packets**

#### Literal Data Packet:-

* + This packet carries or holds the **actual data** that is being transmitted or stored.
  + This packet is the most elementary type of message; i.e, it cannot carry any other packet.

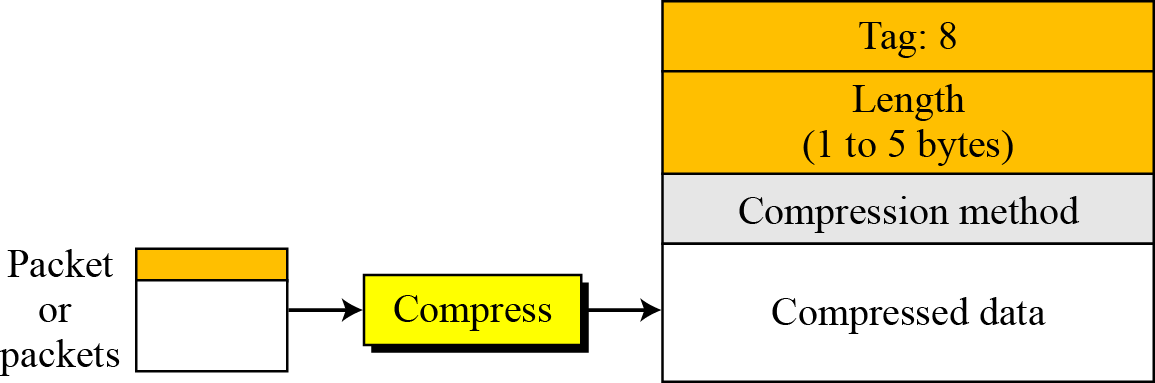
The format is shown in below fig.



* **Mode:-** This one-byte field defines how data is written to the packet. The value of this field can be “b” for binary, “t” for text, or any other locally defined value.
* **Length of next field:**- This is one-byte field defines the length of the next field (file name field).
* **File name:**- This variable-length field defines the name of the file or message as an ASCII string.
* **Time Stamp:**- This four-byte field defines the time of creation or last modification of the message. The value can be 0, which means that user chooses not specify a time.
* **Literal data:**- This variable-length carries the actual data(file or message) in text or binary(depending on the value of the mode field).

#### Compressed Data Packet:-

* + This packet carries compressed data packet. Figure shows the format of a compressed data packet.



#### Compression Method:-

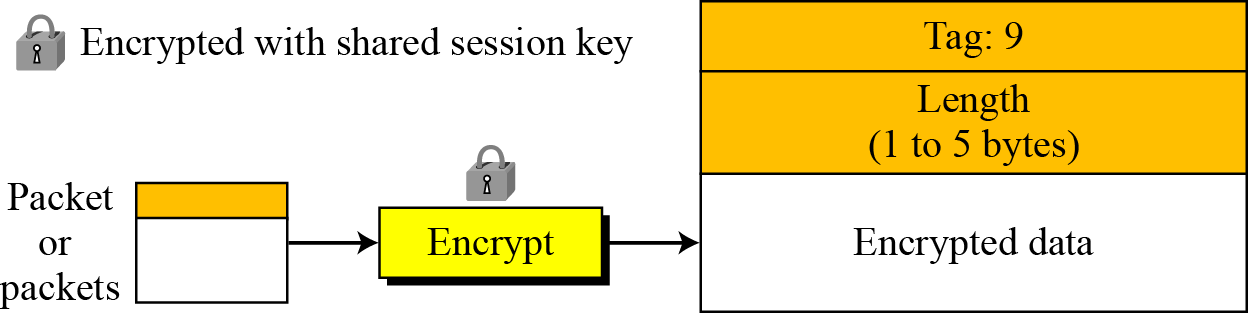
* + This one-byte field defines the compression method used to compress the data
  + The value defines for this so far are 1(ZIP) and 2(ZLIP).
  + Also, an implementation can use other experimental compression methods.

#### Compressed Data:-

* + This variable-length field carries the data after compression.
  + **Note**: The data in this field can be one or the concatenation of two or more packets.
  + **Example**: A single literal data packet or a combination of a signature packet followed by a literal data packet.

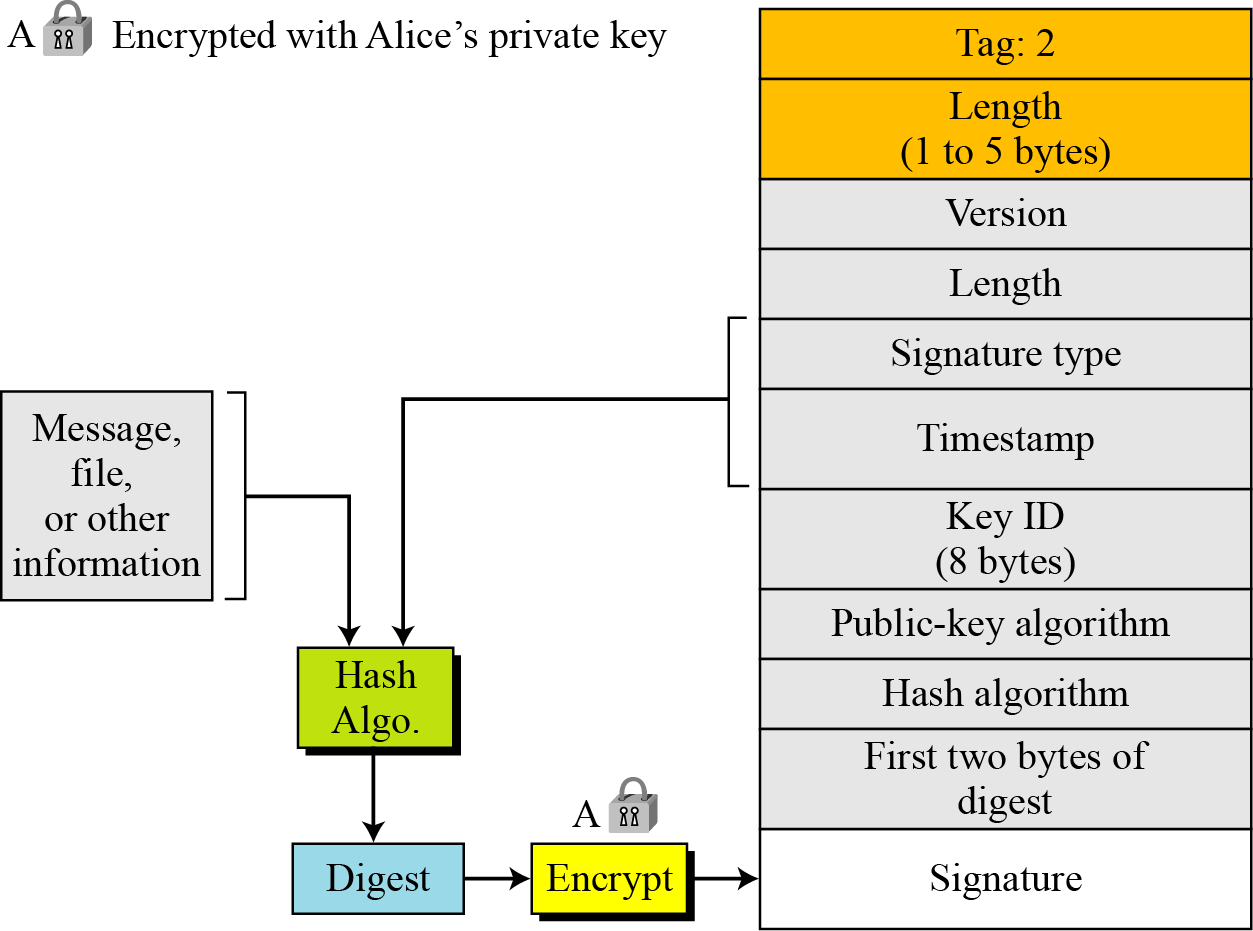
#### Data Packet Encrypted with Secret Key:-

* + This packet carries data from one packet or a combination of packets that have been encrypted using a conventional symmetric-key algorithm.
  + **Note**: A packet carrying the one-time session key must be sent before this packet. Figure shows the format of the encrypted data packet.



#### Signature Packet

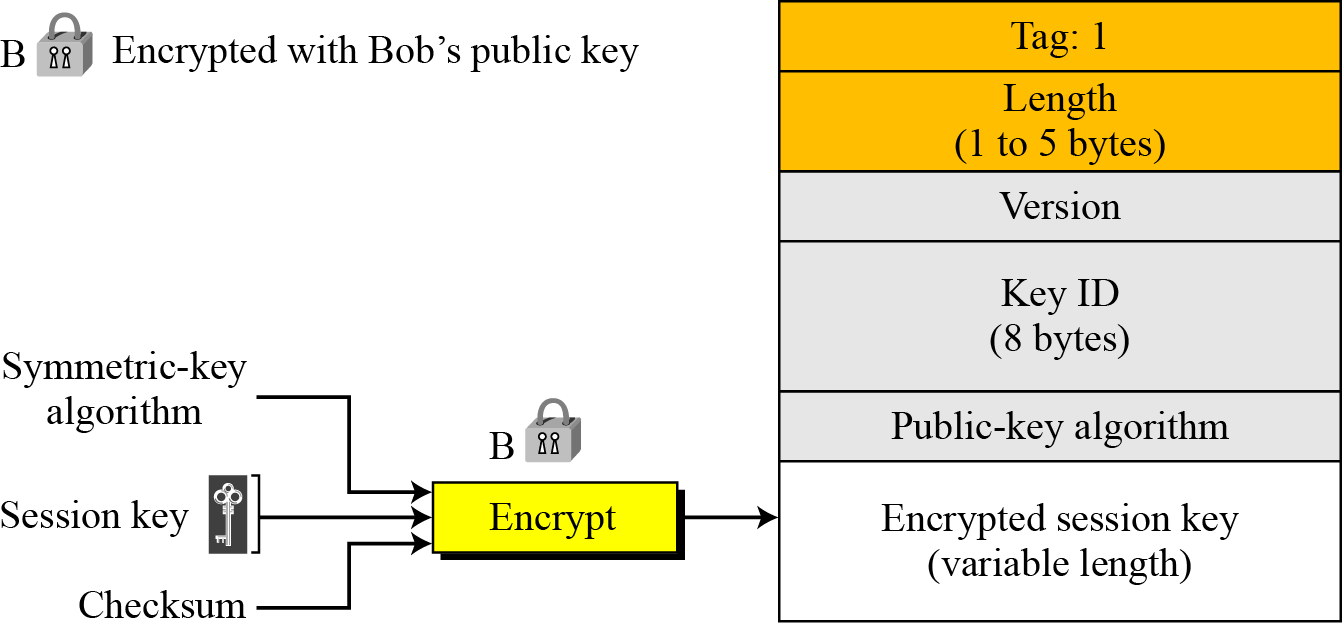
It protects the integrity of the data.



* + **Version**: one-byte field defines the PGP version
  + **Length**: shows the length of the next two fields.
  + **Signature Type**: one-byte field to hold signature type
  + **Timestamp**: four-byte field defines the time signature was calculated.
  + **Key ID**: 8-byte field defines public-key ID of the signer
  + **Public-key Algorithm**: one-byte field gives the code for the public-key algorithm used to encrypt the digest.
  + **Hash – Algorithm**: one-byte filed gives the code for the hash algorithm
  + **First two bytes of Message Digest:** used as a kind of checksum
  + **Signature:** variable length field holds encrypted digest signed by the sender.

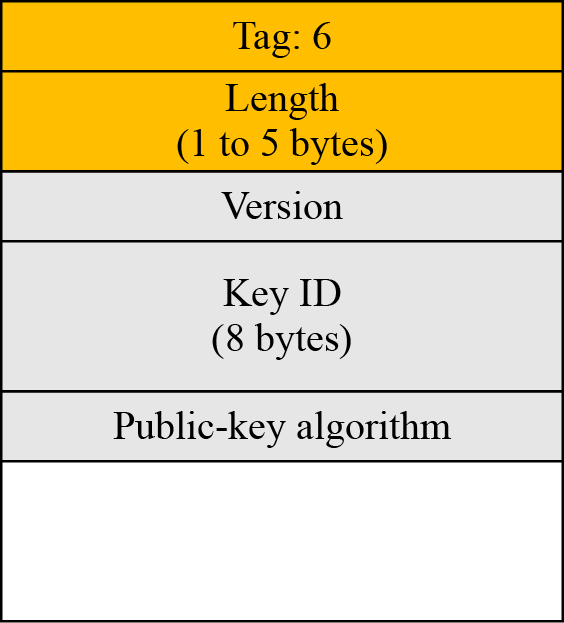
##### Session-key packet Encrypted with Public Key

* + **Version**: one-byte field defines the PGP version
  + **Key ID**: 8-byte field defines public-key ID of the signer
  + **Public-key Algorithm**: one-byte field gives the code for the public-key algorithm used to encrypt the digest.
  + **Encrypted session:** variable – length field holds encrypted value of the session key created by the sender and sent to the receiver



##### Public-key packet

This packet contains the public key of the sender

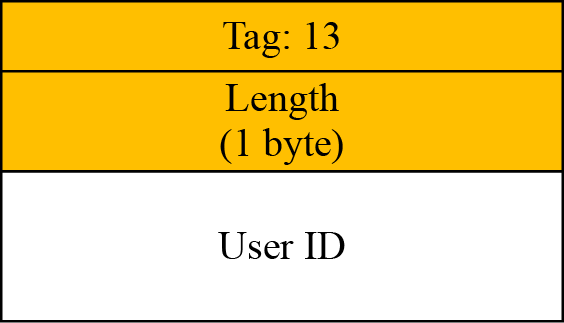


* + **Version**: one-byte field defines the PGP version
  + **Timestamp**: four-byte field defines the time signature was calculated.
  + **Validity**: two-byte field shows the number of days the key is valid.
  + **Public-key Algorithm**: one-byte field gives the code for the public-key algorithm.
  + **Public-key:** variable length field holds the public key itself

##### User ID packet

It identifies the user.

**User ID**: variable – length string defines the user ID of the sender.



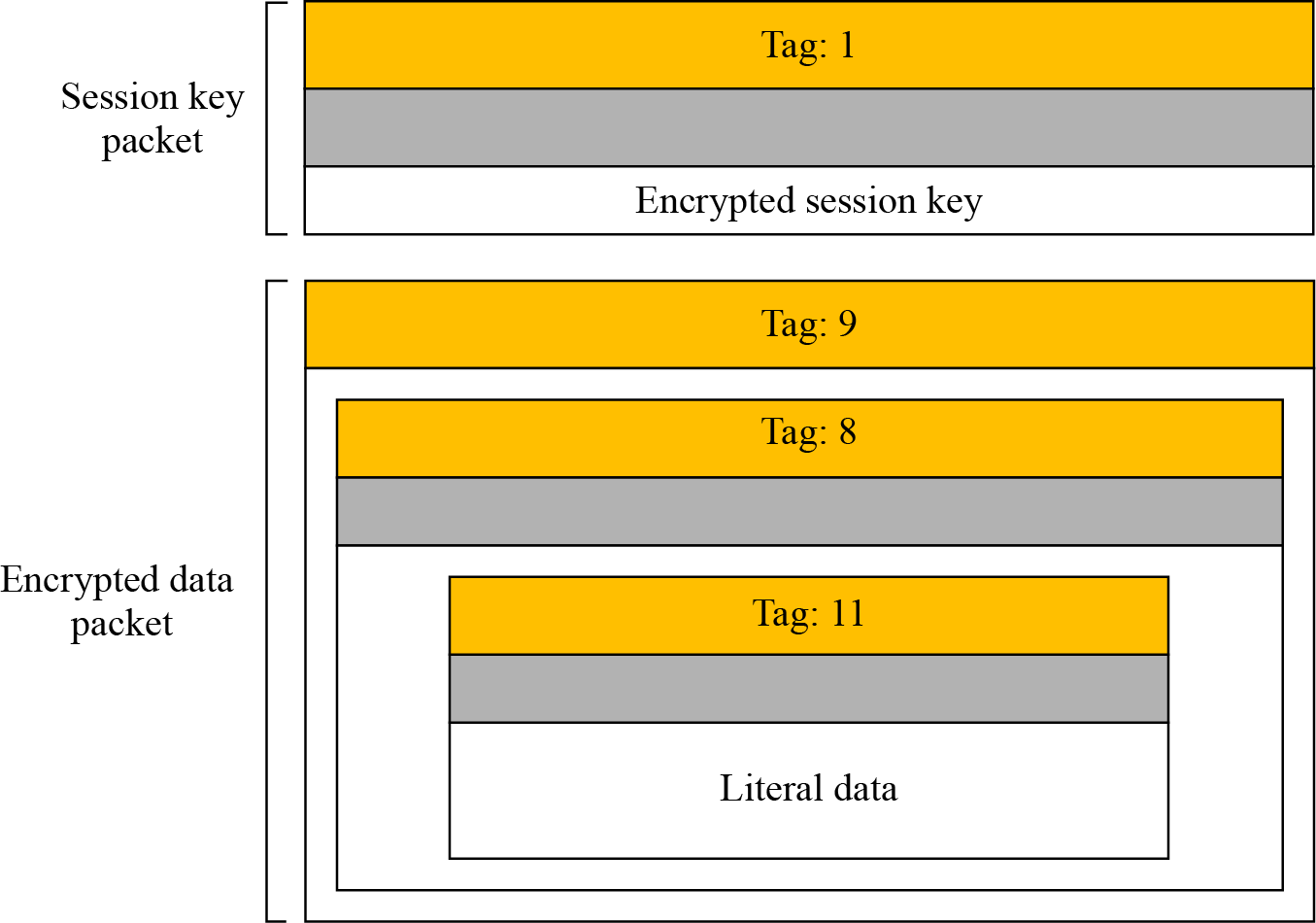
X**-**-----------------------------------------------------------------

## PGP MESSAGES

A message in PGP is a combination of sequenced and/or nested packets. Even though not all combinations of packets can make a message, the list of combinations is still long.

#### Encrypted Message

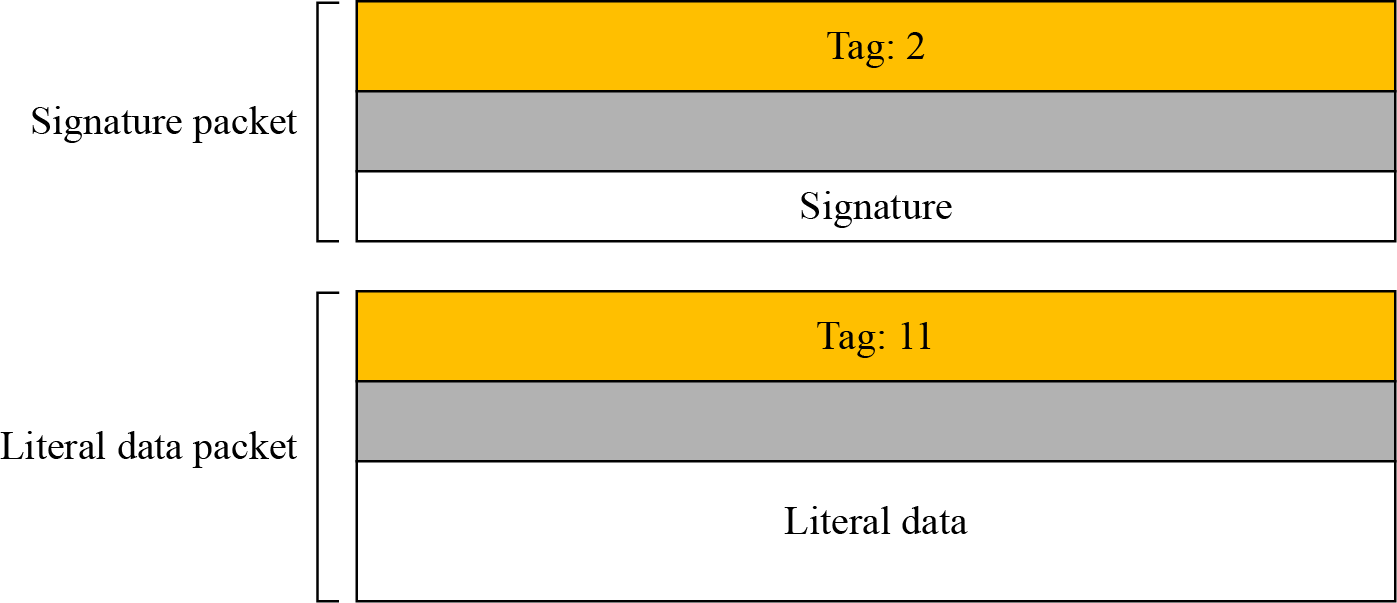
An Encrypted message can be a sequence of two packets, a session-key packet and a symmetrically encrypted packet. The latter is normally nested packet.



**Note**: the session-key packet is just a single packet. The encrypted data packet is made of a compressed packet. The compressed packet is made of a literal data packet. The last one holds the literal data.

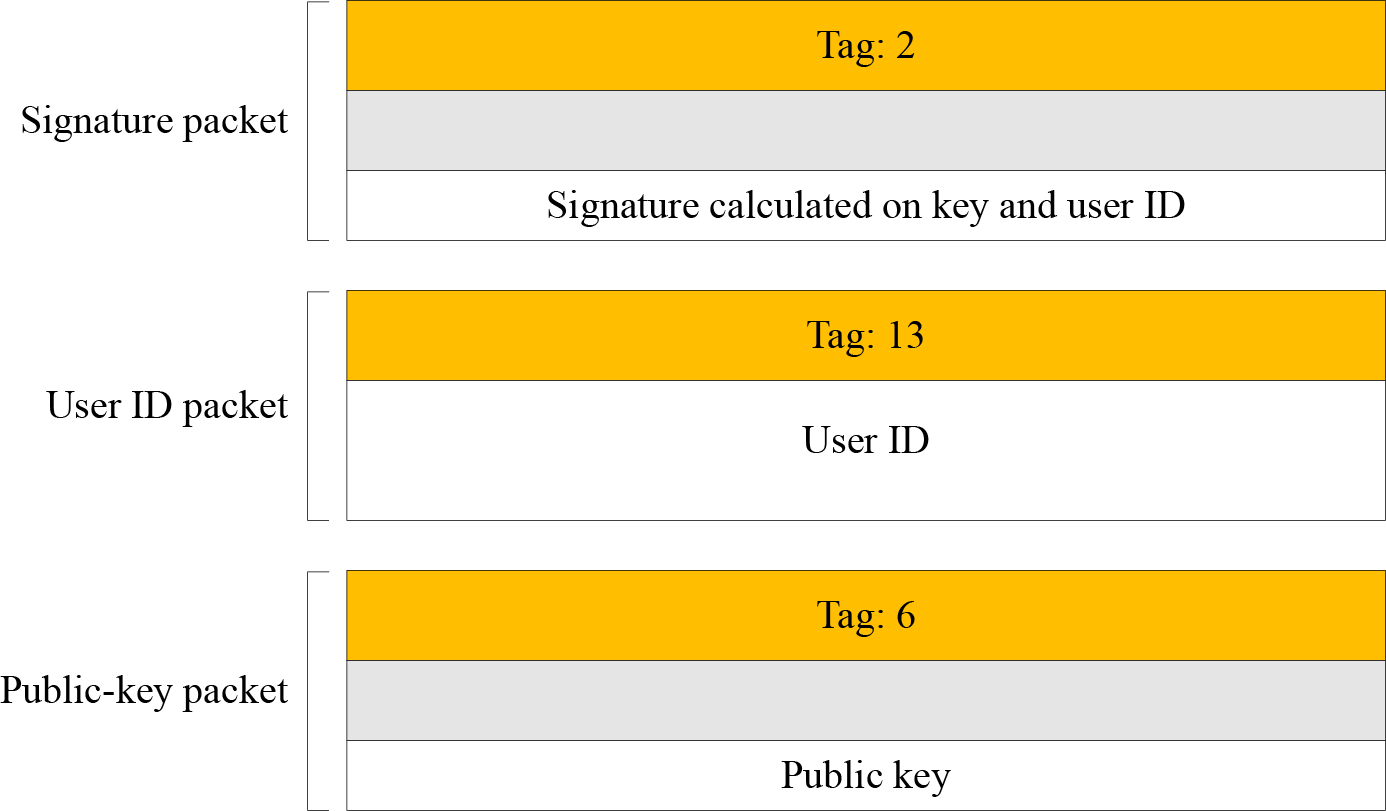
#### Signed Message

A signed Message can be the combination of a signature packet and literal packet.



#### Certificate Message

Although a certificate can take many forms, one simple example is the combination of a user ID packet and a Public-Key packet. The signature is then calculated on the concatenation of the Key and user ID.



#### Applications of PGP

PGP has been extensively used for personal emails.

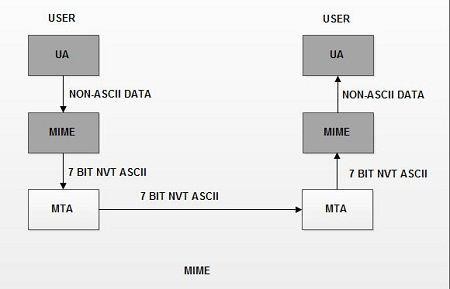
**-----------------------------------------------------------x------------------------------------------------------------------**

## MIME

**MIME** stands for *(****Multipurpose Internet Mail Extensions****).* It is widely used internet standard for coding binary files to send them as e-mail attachments over the internet. MIME allows an E- mail message to contain a non-ASCII file such as a video image or a sound and it provides a mechanism to transfer a non text characters to text characters.

#### MIME was invented to overcome the following limitations of SMTP:

1. SMTP cannot transfer executable files and binary objects.
2. SMTP cannot transmit text data of other language, e.g. French, Japanese, Chinese etc, as these are represented in 8-bit codes.
3. SMTP services may reject mails having size greater than a certain size.
4. SMTP cannot handle non-textual data such as pictures, images, and video/audio content.



**MIME** defines five headers that can be added to the original e-mail header section to define the transformation parameters:

* 1. MIME Version
  2. Content Type
  3. Content Type Encoding
  4. Content Id
  5. Content description

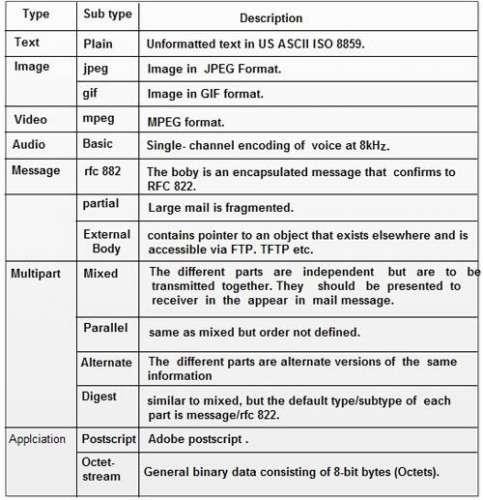
#### MIME Header

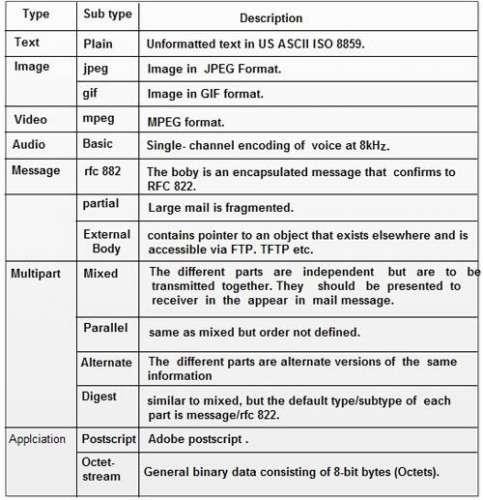
The five header fields defined in MIME are as follows:

1. **MIME-version**. It indicates the MIME version being used. The current version is 1.1. It is represented as : MIME-version: 1.1.
2. **Content-type**. It describes the type and subtype of the data in the body of the message. The content type and content subtype are separated by slash. This field describes how the object in the body is to be interpreted. The default value is plaintext in US ASCII. Content type field is represented as:

#### Context-type: <type/subtype; parameters>

There are seven different types and fourteen sub-types of content.

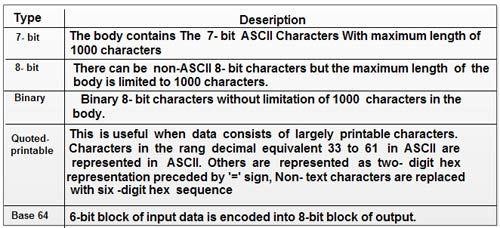




1. **Content-transfer encoding.** It describes how the object within the body has been encoded to US ASCII to make it acceptable for mail transfer. Thus it specifies the method used to encode the message into 0s and 1s for transport. The content transfer encoding field is represented as :

**Content-transfer-encoding** : <type>

The various encoding methods used are given in the table below:



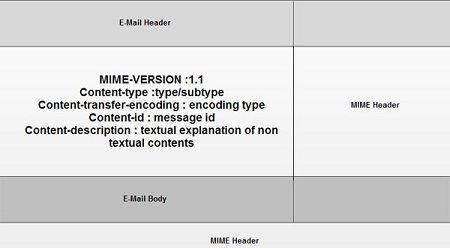
1. **Content-Id**. It is used to uniquely identify the MIME entities in multiple contexts i.e. it uniquely identifies the whole message in a multiple message environment. This field is represented as:

#### Content-id : id = <content-id>

1. Content-description. It is a plaintext description of the object within the body; It specifies whether the body is image, audio or video. This field is represented as:

#### Content-description: <description>

The various fields in the MIME header are



**-----------------------------------------------------------x------------------------------------------------------------------**

## S/MIME:-

S/MIME. adds some new Content types to include security services to the MIME. All of these new types include the parameter "applicationlpkcs7-mime", in which "pkcs" defines "Public Key Cryptography Specification."

#### Cryptographic Message Syntax (CMS):-

To define how security services. such as confidentiality or integrity, can be added toMIME content types. S/MIME has defined **Cryptographic Message Syntax (CMS)**.The syntax in each case defines the exact encoding scheme for each content The following describe the type of message and different subtypes that are created from these messages. For details, the reader is referred to RFC3369 and 3370.

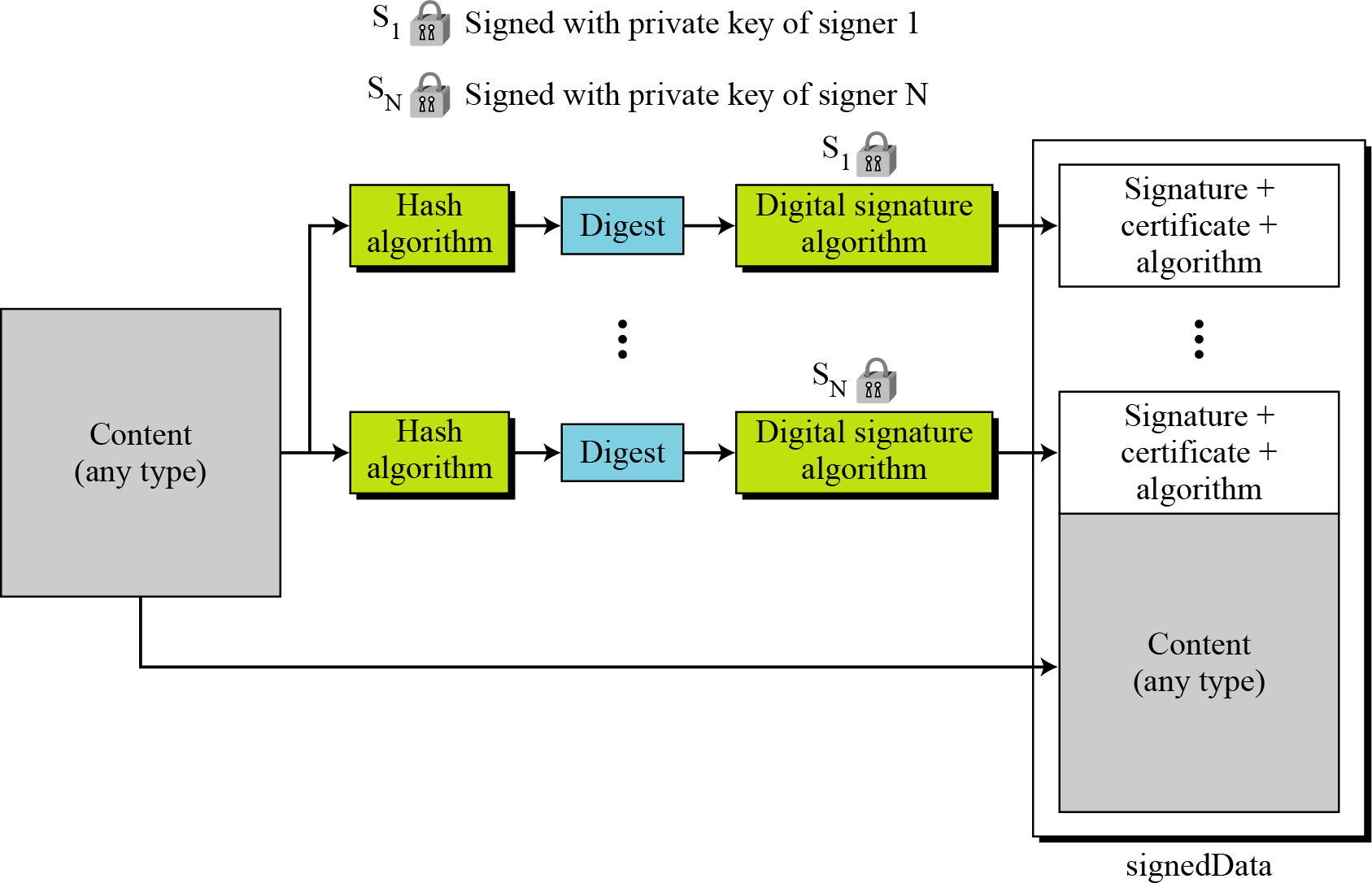
1. **Data Content Type:-**This an arbitrary string. object created is called Data.

#### Signed-Data Content Type:-

* + It provides only integrity of data.
  + It contains any type and zero or more signature values.
  + The encoded result is an object called Signed Data The following are the steps to create **Signed-Data**:

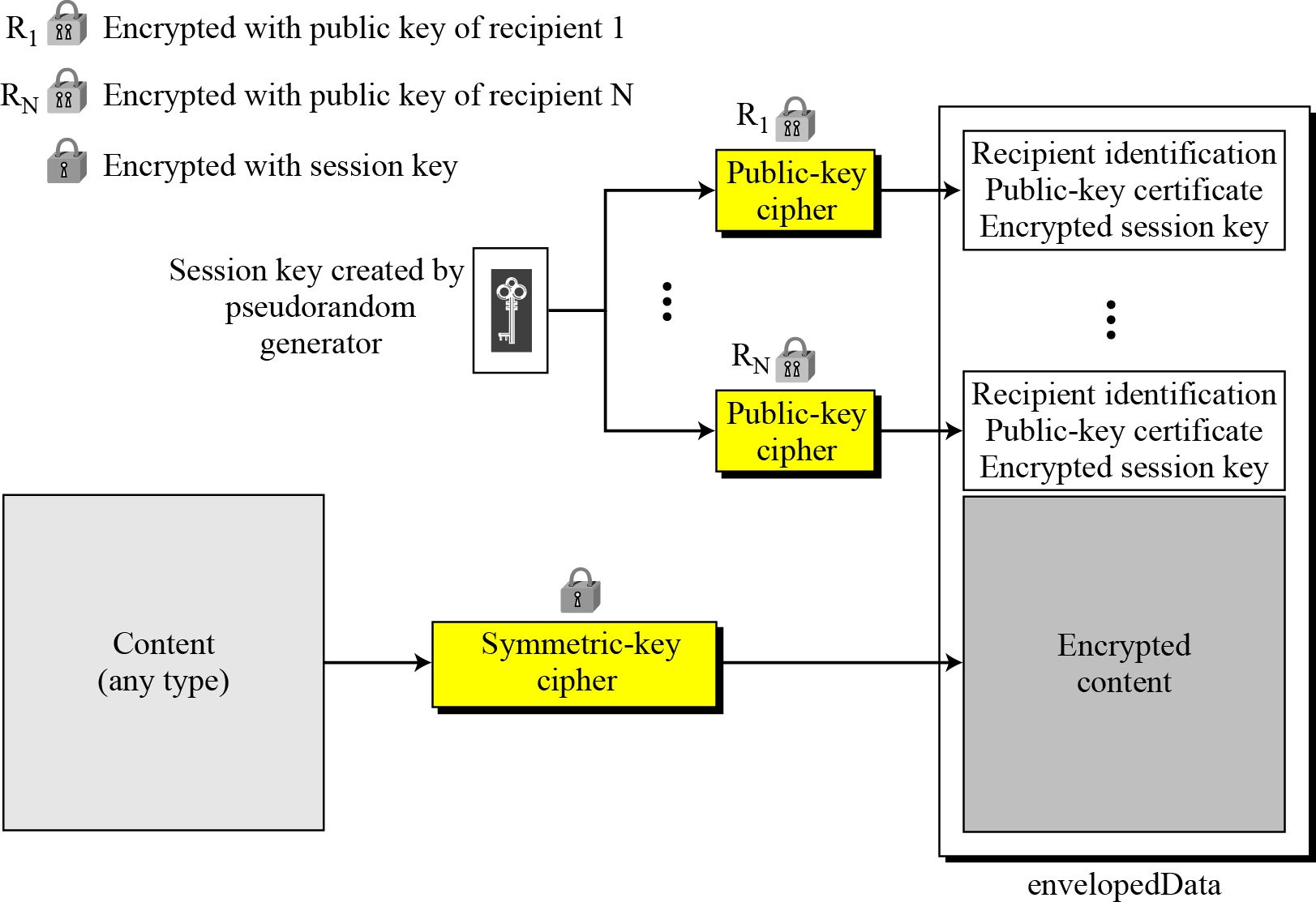
**Step 1:** For each signer, a message digest is created from the content using the specific hash algorithm chosen by that signer.

**Step 2:** Each message digest is signed with the private key of the signer.

**Step 3.** The content, signature values, certificates, and algorithms are then collected to create the Signed Data Object.

#### Enveloped-Data Content Type:-

* + This type is used to provide privacy for the message.
  + It contains any type and zero or more encrypted keys and certificates. The encoded result is an object called enveloped Data.
  + The process Of creating an object of this type:
    1. A pseudorandom session key is created for the symmetric-key algorithms to used
    2. For each recipient , a copy of the session key is encrypted with the public key of each recipient
    3. The content is encrypted using the defined algorithm and created session key.
    4. The encrypted contents, encrypted session keys, algorithm used, and certificates are encoded using Radix-64.

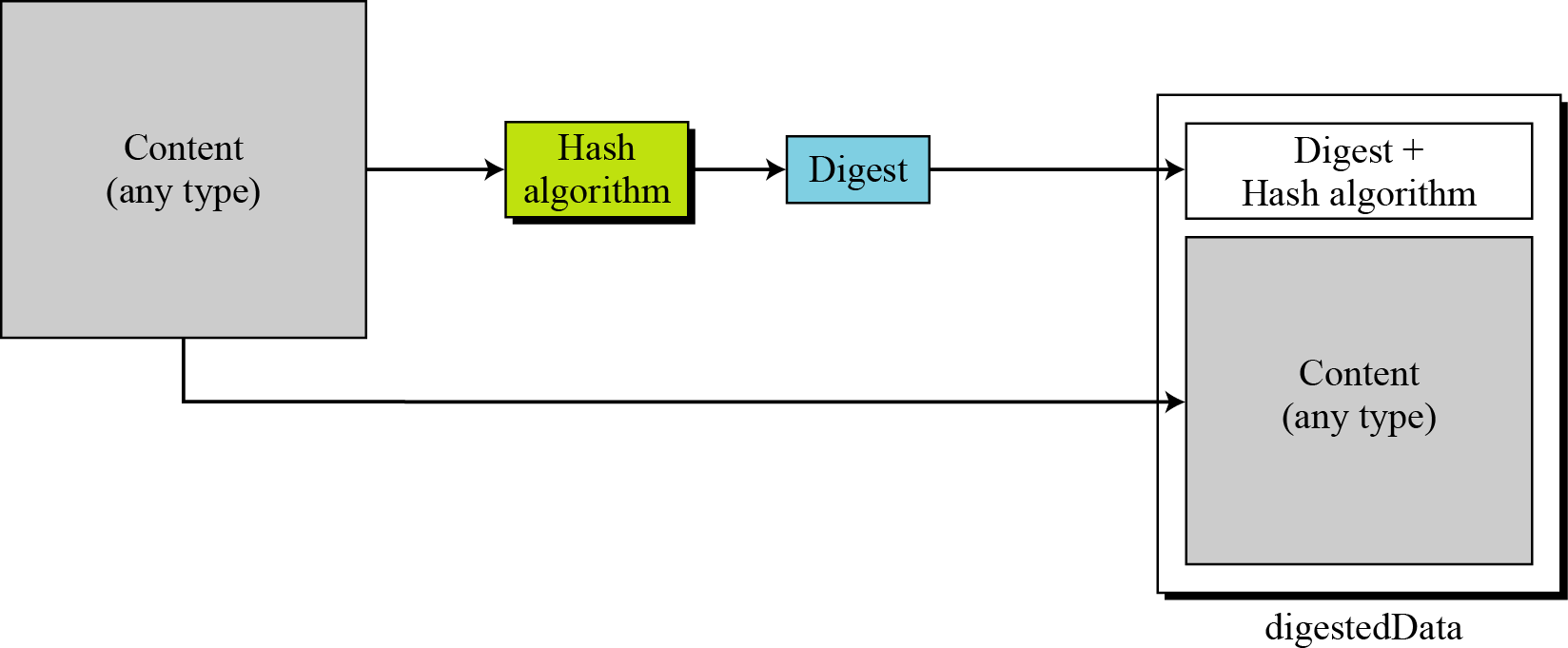


#### Digested data content type:-

It is used to provide Integrity for the message. The result is used as the enveloped-data content type. The encoded result is d**igesteData.**

#### Process for creating this object:

* 1. A message digest is calculated from the content
  2. The message digest the algorithm and the content are added together to create the d**igesteData** object**.**



#### Encrypted-Data content type:-

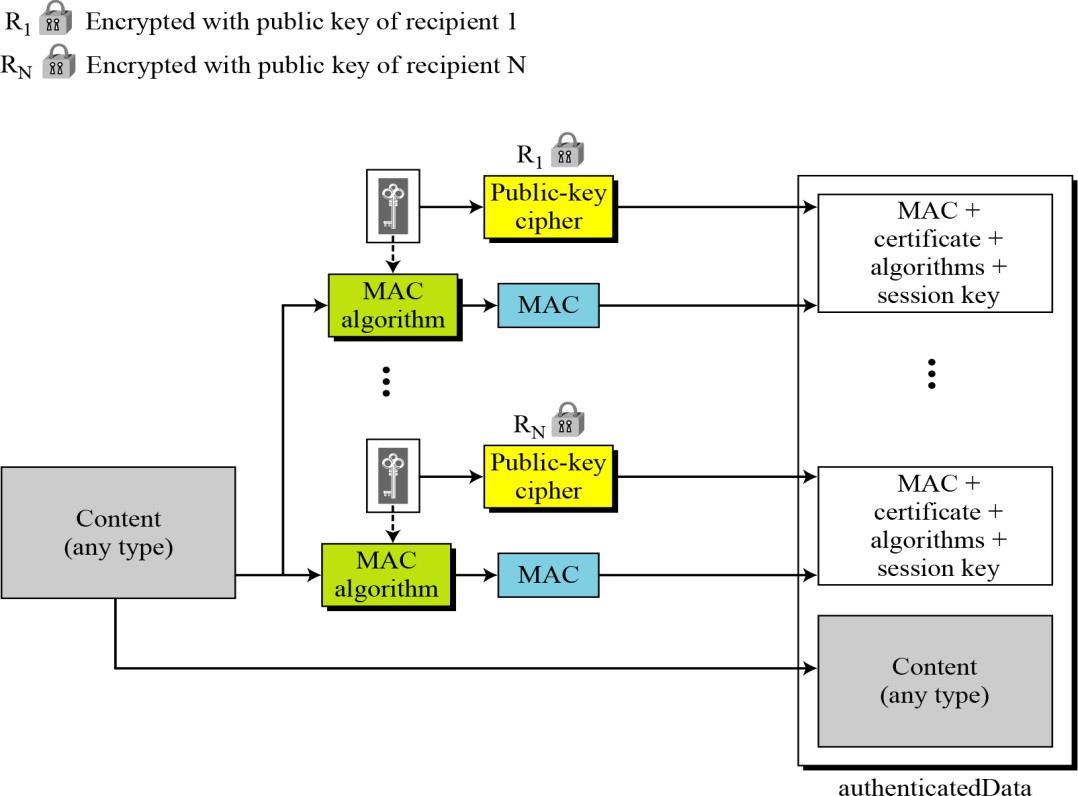
It used to create an encrypted version of any content type. It has no recipient. It is used to store the encrypted data instead of transmitting it.

##### Authenticated-data content type:-

It used to provide ***Authentication of the data.***

The process:

1. Using pseudorandom generator, a MAC key is generated for each recipient.
2. The MAC key is encrypted with public key of the recipient
3. A MAC is created for the content
4. The content, MAC, algorithms and other information are collected together to form the autheticatedData object.



x

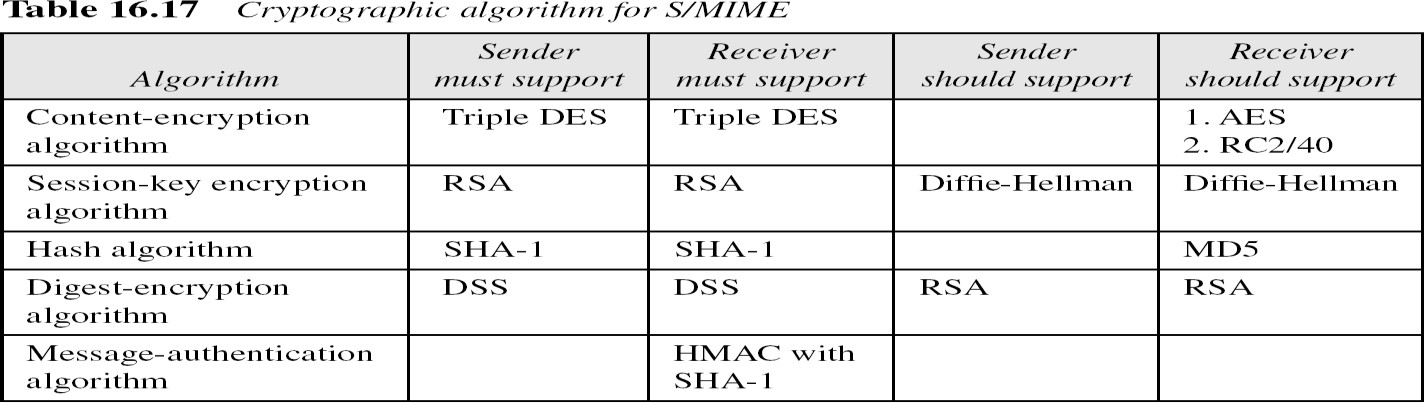
### Key Management

The Key Management in S/MIME is combination of Key Management used by X.509 and PGP. S/MIME uses public-key certificates signed by the Certificate Authorities defined by X.509. Each user in S/MIME maintains PGP Web of trust model to verify the signatures.

x

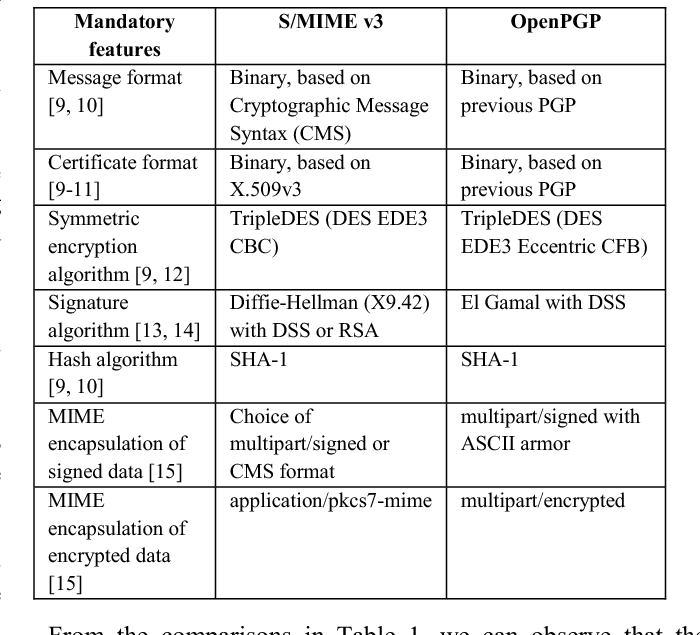
### Cryptographic Algorithms

S/MIME defines several cryptographic algorithms. The term “must” means an absolute requirement; the term “should” means recommendation.



x

* 1. **Difference between S/MIME and PGP**

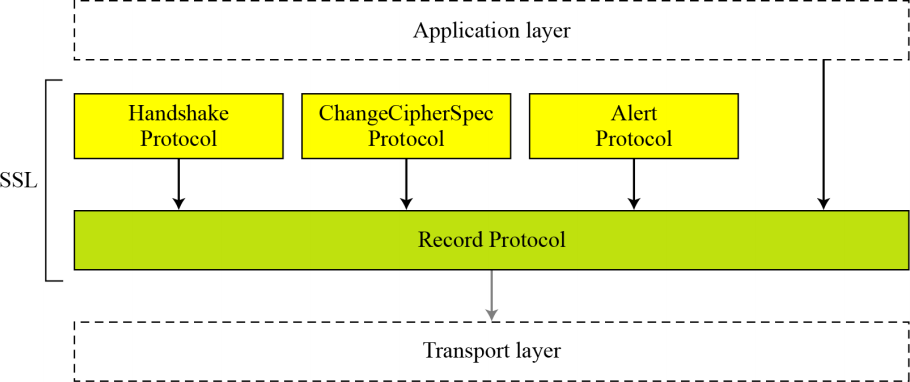


# PART2: Security at transport layer

* 1. **SSL - Secure Socket Layer**
     + SSL was developed by Netscape.
     + It provides server authentication, client authentication and encrypted communication b/w two machines.
     + TSL is derived from SSLv3. Therefore TSL is SSLv3.1
     + SSL and TSL cryptographic protocols are referred to as ”SSL”.
     + SSL uses a combination of public-key and symmetric-key encryption to secure a connection between two machines.
     + These are used in applications such as web browser, e-mail, voice-over-IP(VoIP), File transfer etc

**SSL Architecture**

* + - SSL uses TCP for reliable end-to-end secure communication b/w two machines.
    - SSL is a two layers of protocols.
    - Two layers of protocols
      * Higher layer
        + Handshake protocol
        + change cipher spec protocol
        + Alert protocol
      * Lower layer
        + Record protocol



#### SSL Record Protocol:

Handles data security and integrity; encapsulates data sent by higher level protocols, such as HTTP.

#### Handshake, Cipher change, Alert:

Establish a connection; session management, crypto management, SSL message transfer

**-------------------------------------------------------------------------------------------------------------------------------**

* 1. **Handshake Protocol**

This protocol allows the server and client to

* + - authenticate each other
    - negotiate an encryption and MAC algorithm
    - agree cryptographic keys to be used to protect payload data

The handshake protocol is used before any application data is transmitted. It establishes a **session**.

* + - A session defines the set of cryptographic security parameters to be used
    - Multiple secure connections between client and server can share the same session
      * reduces computation cost

The Handshake Protocol consists of a series of messages exchanged by client and server, in **four phases.**

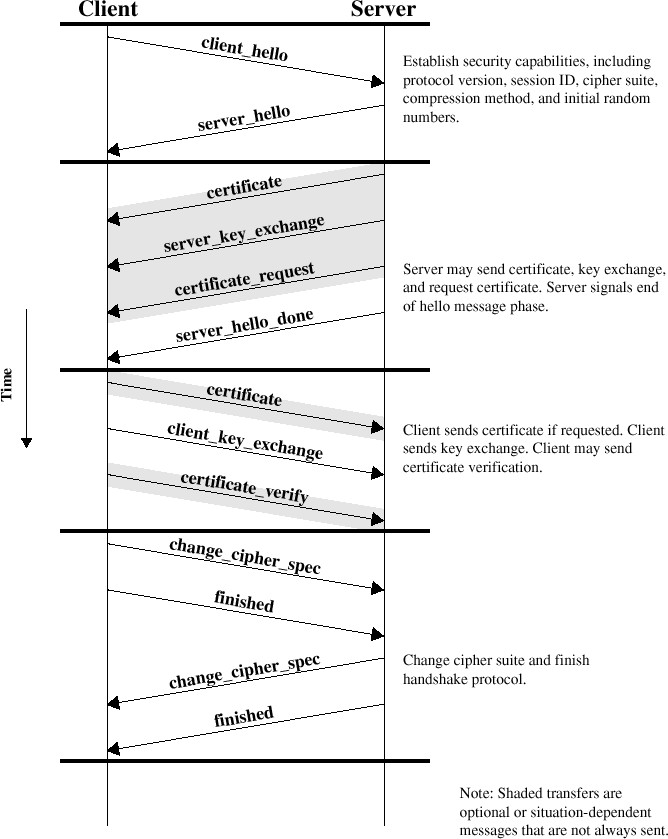
#### Phase 1. Establish Security Capabilities

**Phase 2. Server authentication and key exchange Phase 3. Client authentication and key exchange Phase 4. Finalizing the Handshake protocol.**

|  |  |
| --- | --- |
| **Handshake protocol Message Format** | **Type (1 byte):** Indicates one of 10 messages.(ex: Hello, Certificate, key exchange)  **Length (3 bytes):** The length of the message.  **Content ( bytes):** The parameters associated with this message. |

|  |  |
| --- | --- |
|  | **After Phase I, the client and server know the following:**  ❏ The version of SSL  ❏ The algorithms for key exchange, message authentication, and encryption  ❏ The compression method  ❏ The two random numbers for key generation |

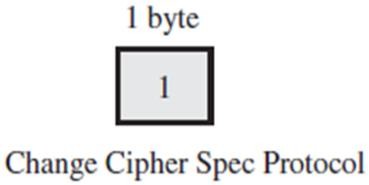
|  |  |
| --- | --- |
|  |  |
|  | **After Phase II,**  ❏ The server is authenticated to the client.  ❏ The client knows the public key of the server if required. |
|  | **After Phase III,**  ❏ The client is authenticated for the server.  ❏ Both the client and the server know the pre-master secret. |
|  | **After Phase IV**, the client and server are ready to exchange data |



**-------------------------------------------------------------------------------------------------------------**

#### ChangeCipherSpec Protocol

* It uses the SSL Record Protocol.
* This protocol consists of a single message, which consists of a single byte with the value 1.
* It causes the pending state to be copied into the current state, which updates the cipher suite to be used on this connection.

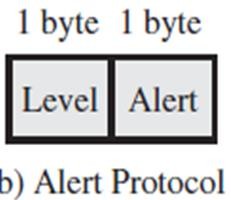


#### Movement of parameters from pending state to active state

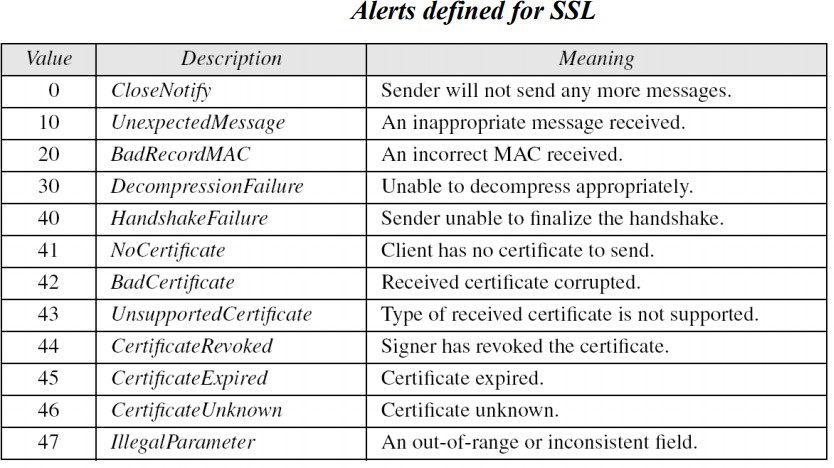
**----------------------------------------------------------------------------------------------------------------------**

#### Alert protocol

* It conveys SSL-related alerts to peer.
* Alert messages are compressed and encrypted.
* Each message consists of two bytes



* 1st byte indicates alert level
  + Warning or fatal
  + SSL will immediately terminates the connection
  + Ex: UnexpectedMessage, BadRecordMAC, DecompressionFailure etc.
* 2nd byte indicates specific alerts
  + Warning alerts
  + Ex: CloseNotify, NoCertificate, BadCertificate etc.



**-------------------------------------------------------------------------------------------------------------**

* 1. **Record Protocol**
     + The SSL Record Protocol provides two services for SSL connections:

#### Confidentiality:

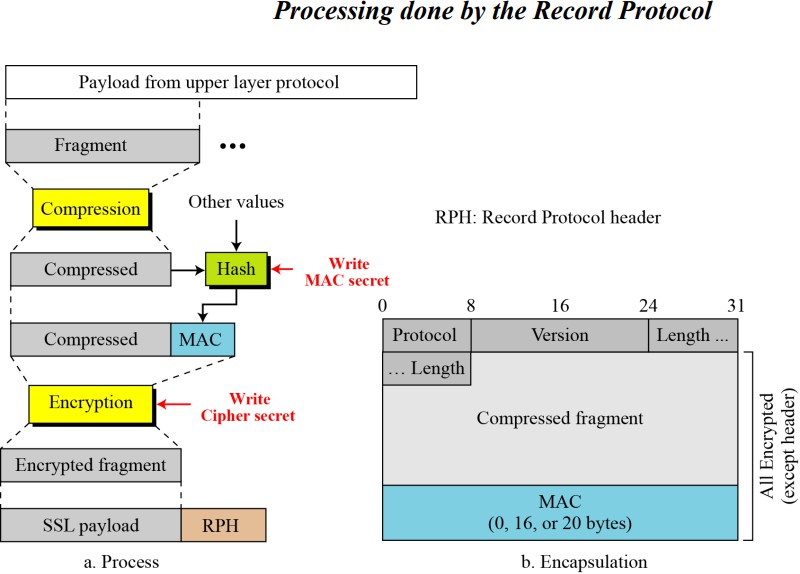
* + - * The Handshake Protocol defines a shared secret key that is used for conventional encryption.

#### Message Integrity:

* + - * The Handshake Protocol also defines a shared secret key that is used to form a message authentication code (MAC).

#### Record Protocol operation:

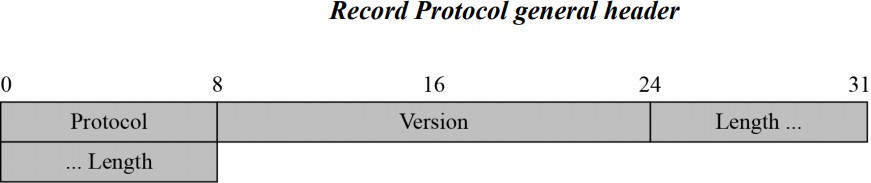
1. **Fragmentation**: Each upper-layer message is fragmented into blocks of 214 bytes or less.
2. **Compression**: Optional step and may not increase the content length by more than 1024 bytes
3. **Message Authentication Code**: A shared secret key is used to compute MAC.
4. **Encryption**: compressed message and MAC are encrypted using symmetric encryption.
5. **RPH :** Record Protocol Header preparation: Header consists of the following:
   * Protocol (1 byte): The higher-layer protocol used to process the enclosed fragment.
   * Version (2 bytes): holds Major and Minor version of SSL.
   * Length (2 bytes): The length in bytes of the plaintext fragment (or compressed fragment).



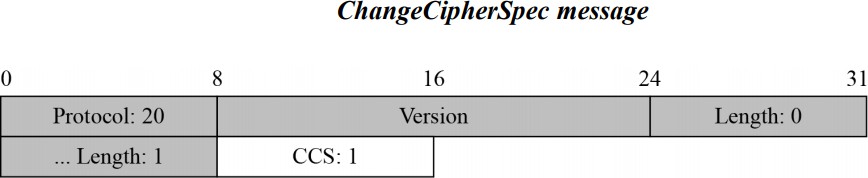
**------------------------------------------------------------------------------------------------------------------------**

## SSL MESSAGE FORMATS

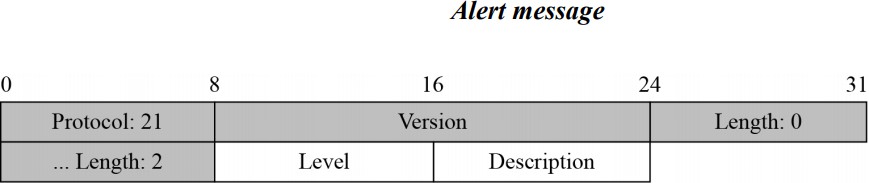
* + - **Record Protocol general header**



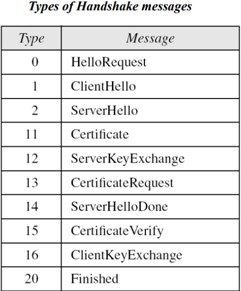
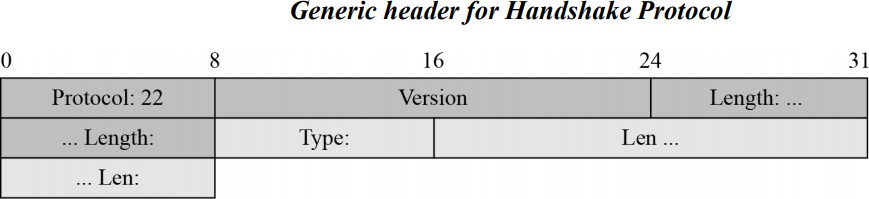
* + - **ChangeCipherSpec Protocol**

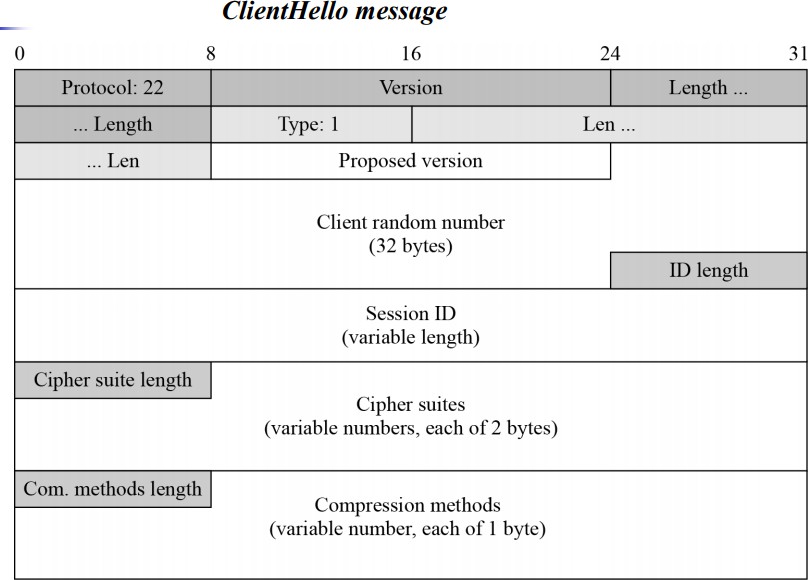


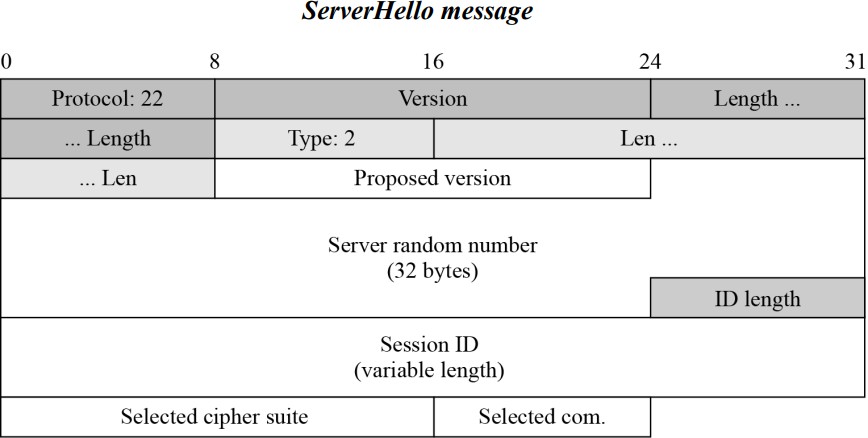
* + - **Alert Protocol**

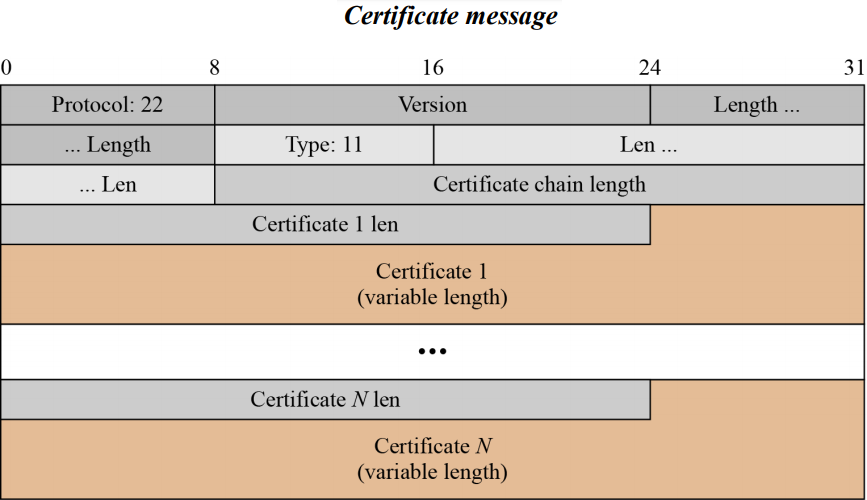


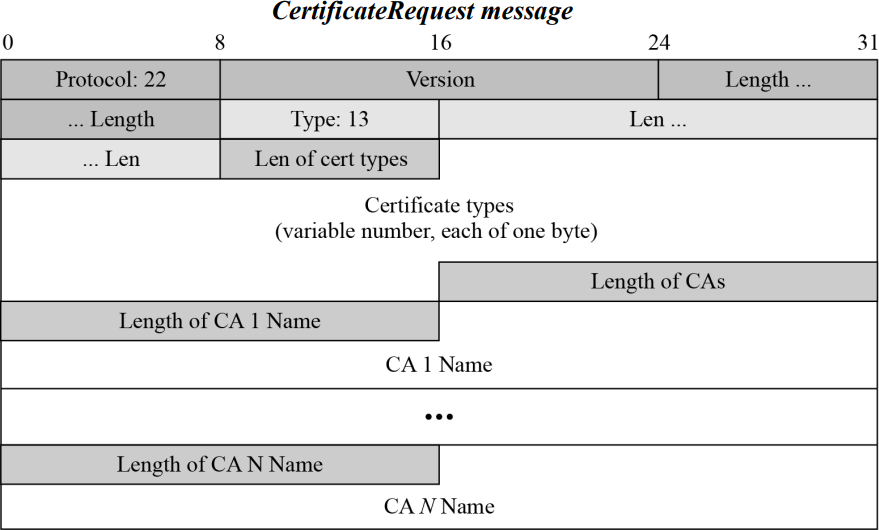
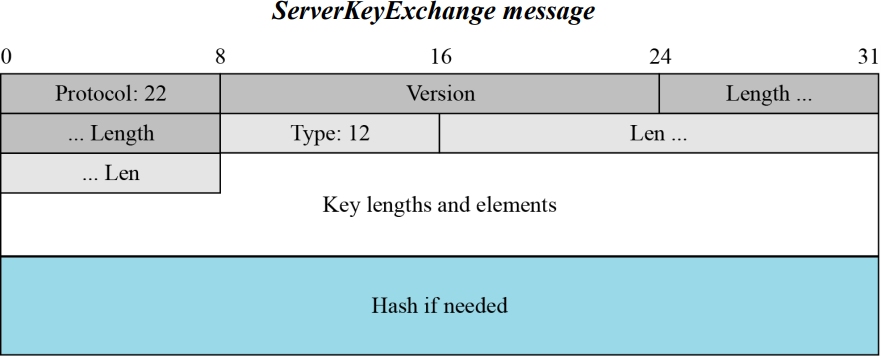
* + - **Handshake Protocol**

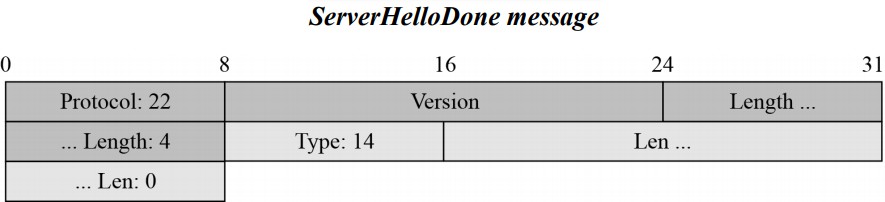


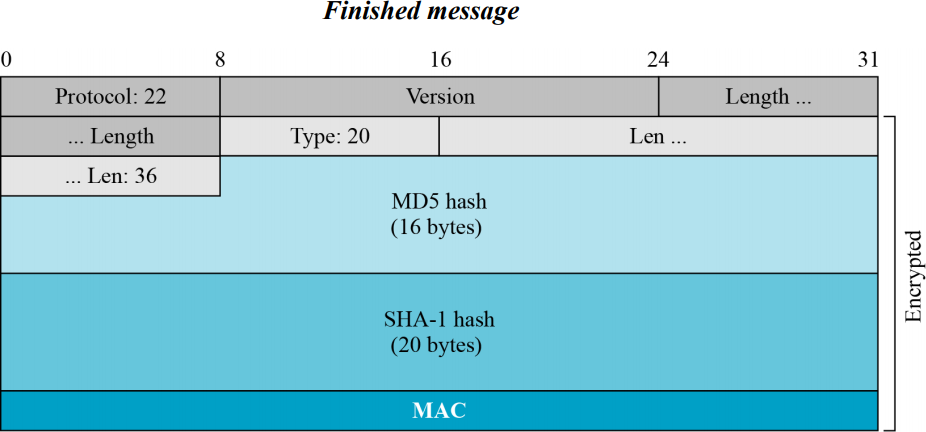
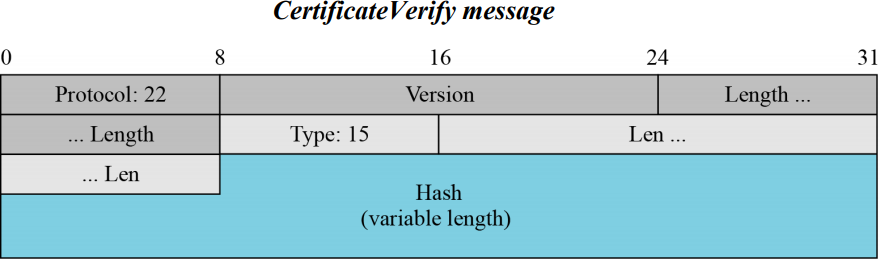




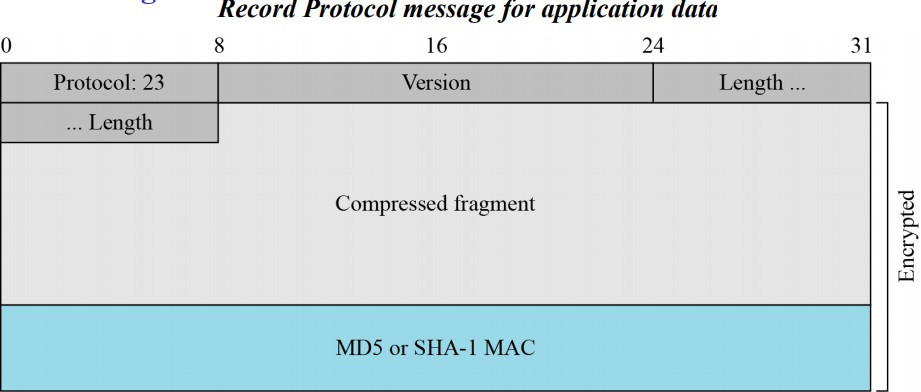








* + - **Application Data**

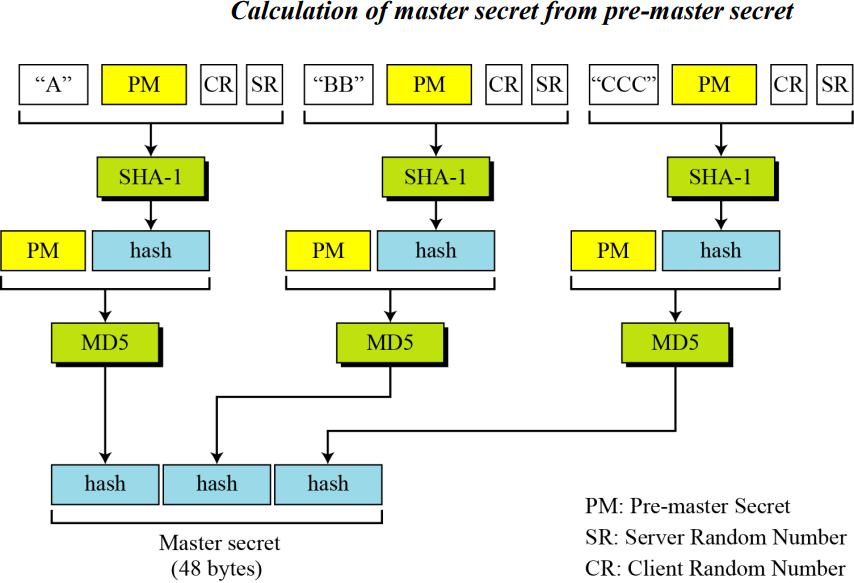


**------------------------------------------------------------------------------------------------------------------------**

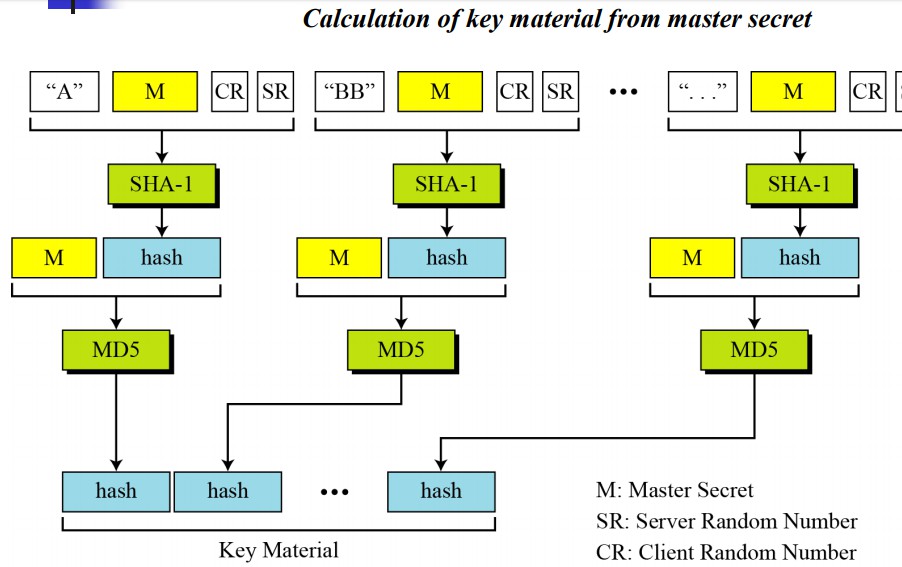
* 1. **Cryptographic Parameter Generation**

The parameters are generated using the following procedure:

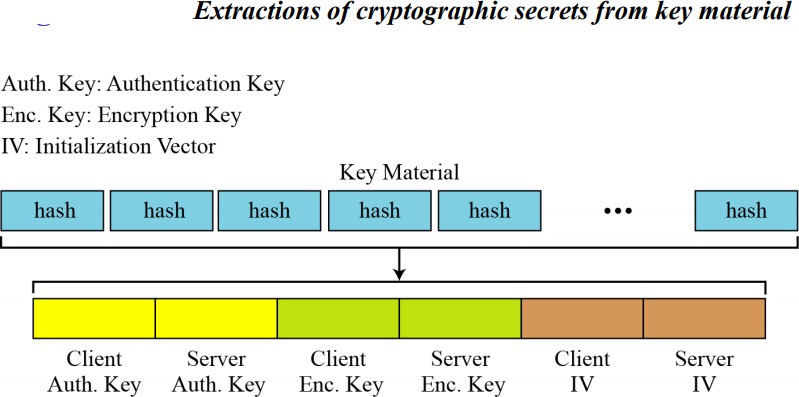
1. The client and server exchange two random numbers—one is created by client and the other by the server.
2. The client and server exchange one pre-master secret using one of the key-exchange algorithms.
3. A 48-byte master- secret created by applying two hash functions SHA-1 and MD5



1. The master- secret is used to create variable-length **key material** by applying set of hash functions and prepending with different constants.

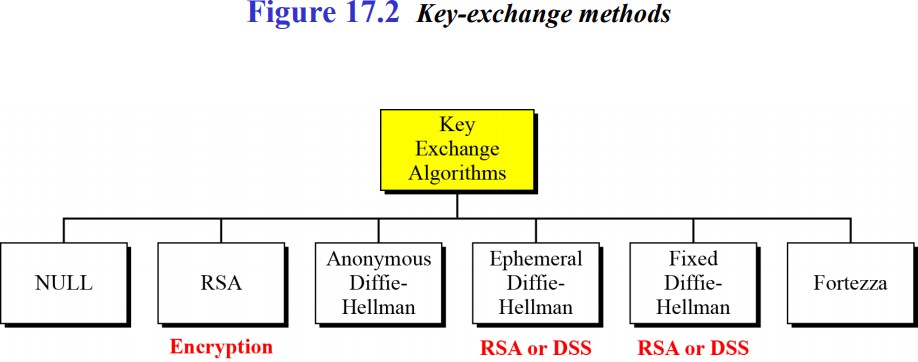


1. Six different keys are extracted from the **key material**



**------------------------------------------------------------------------------------------------------------------------**

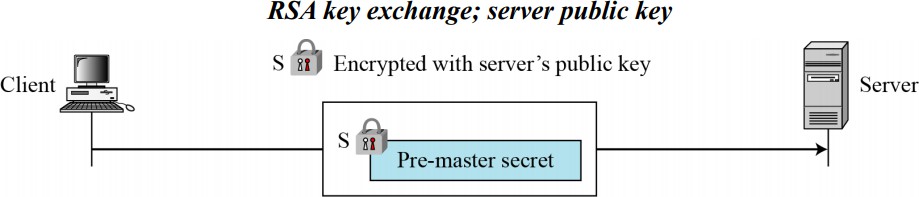
**Key Exchange Algorithms**



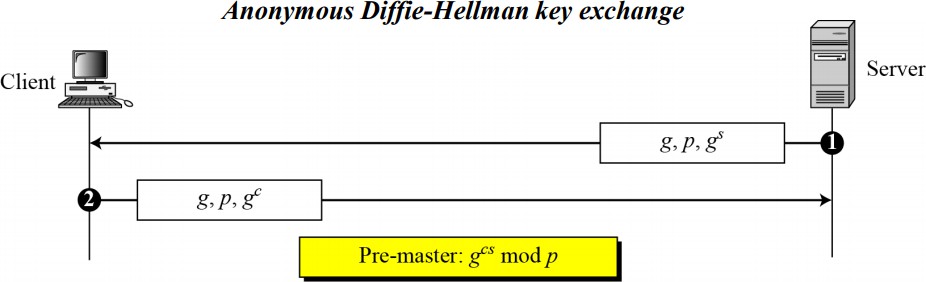
**Null**

There is no key exchange in this method. No premaster secret is established between the client and the server. Both client and server need to know the value of the pre-master secret.

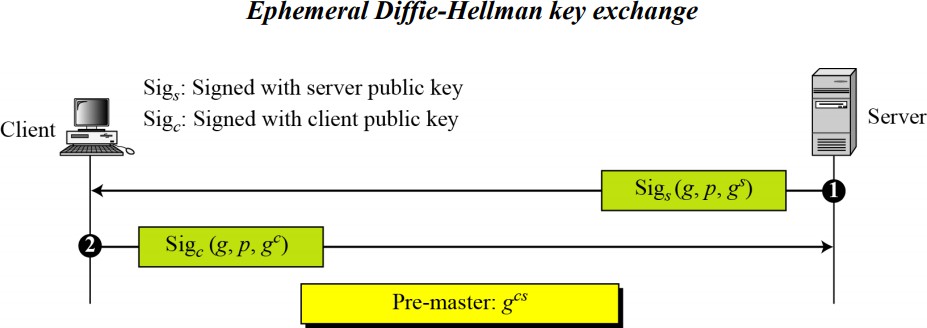
## RSA



**Anonymous Diffie-Hellman**



**Ephemeral Diffie-Hellman key exchange**



**Fixed Diffie-Hellman**

Another solution is the fixed Diffie-Hellman method. All entities in a group can prepare fixed DiffieHellman parameters (g and p).

**Fortezza**

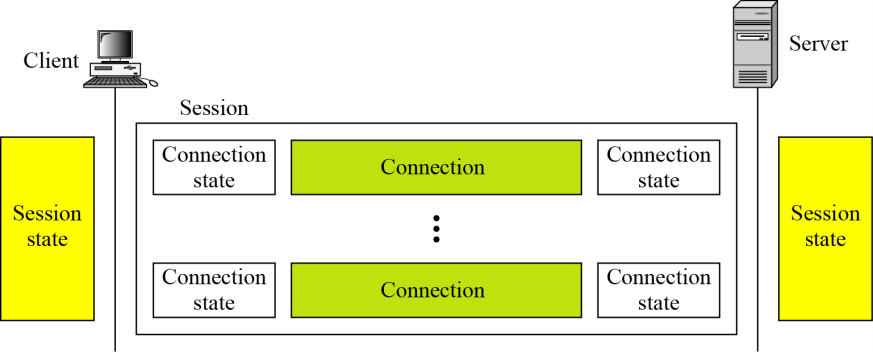
Fortezza is a registered trademark of the U.S. National Security Agency (NSA). It is a family of security protocols developed for the Defense Department.

**------------------------------------------------------------------------------------------------------------------------**

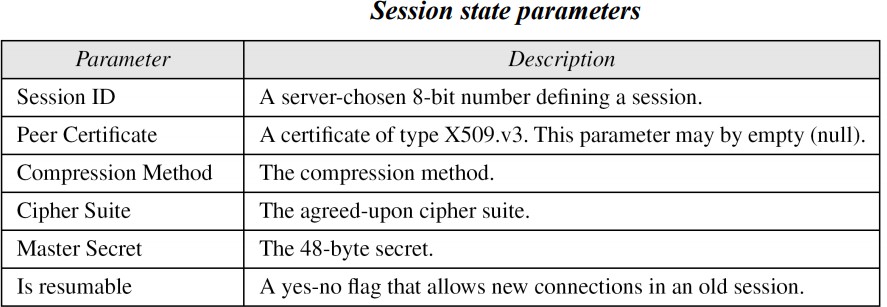
* 1. **A session and connections**

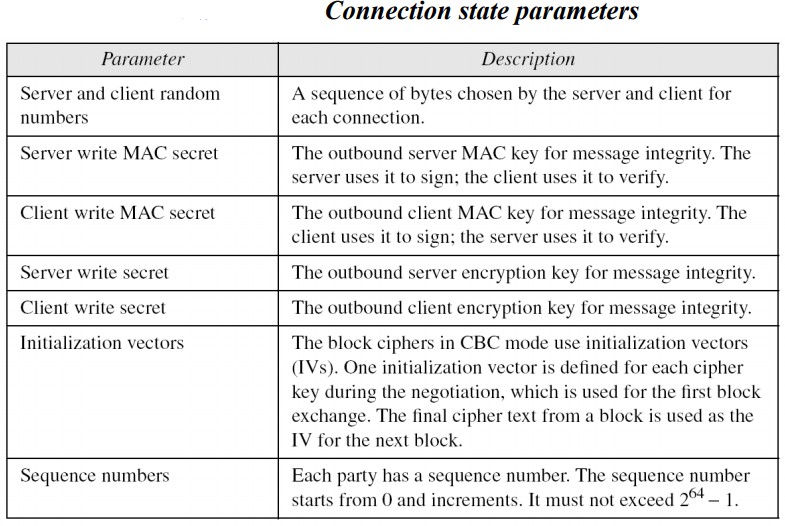
SSL Connection:

* A transient peer-to-peer communications link.
* Each connection is associated with one SSL session. SSL Session:
* A session is an association between client and server.
* It is created by the Handshake Protocol.
* It defines a set of security parameters.
* It may be multiple SSL sessions shared b/w parties.
* It is useful to avoid expensive negotiations of security parameters for each connection.
* Single session has many connections. Every connection has a different key



Session states:

* Current state: once a session is established, a current state is created for receive and send messaged.
* Pending state: during the Handshake protocol a pending state is created for receive and send. If the Handshake is successful, pending state will become current state.



x