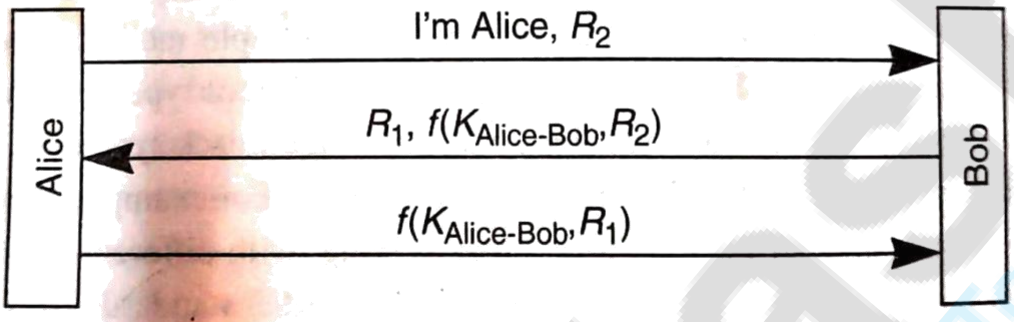
11.2.1 Reflection Attack

The first thing we might notice is that the protocol is inefficient. We can reduce the protocol down to three messages (instead of five used above) by putting more than one item of information into each message:



Protocol 11-8. Optimized mutual authentication based on a shared secret $K_{Alice\text{-}Bob}$

This version of the protocol has a security pitfall known as the **reflection attack**. Suppose Trudy wants to impersonate Alice to Bob. First Trudy starts Protocol 11-8, but when she receives the challenge from Bob, she cannot proceed further, because she can't encrypt R_1.

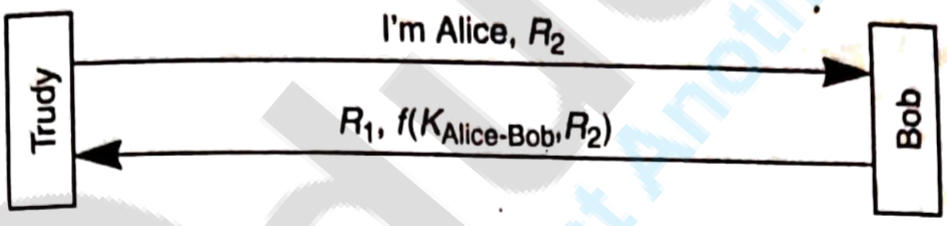


Figure 11-9. Beginning of reflection attack

"I can't explain myself, *I'm afraid sir," said Alice, "because I'm not myself, you see."*

Alice in Wonderland

However, note that Trudy has managed to get Bob to encrypt R_2. So at this point Trudy opens a second session to Bob. This time she uses R_1 as the challenge to Bob:

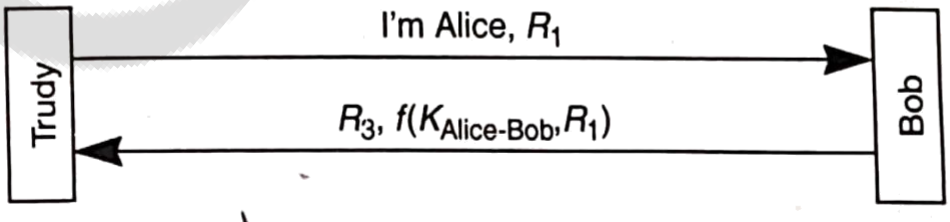


Figure 11-10. Second session in reflection attack

Trudy can't go any further with this session, because she can't encrypt R_3. But now she knows $K_{\text{Alice-Bob}}\{R_1\}$, so she can complete the first session.

This is a serious security flaw, and there are deployed protocols that contain this flaw. In many environments it is easy to exploit this, since it might be possible to open multiple simultaneous connections to the same server, or there might be multiple servers with the same secret for Alice (so Trudy can get a different server to compute $f(K_{\text{Alice-Bob}}, R_1)$ so that she can impersonate Alice to Bob).

We can foil the reflection attack if we are careful and understand the pitfalls. Here are two methods of fixing the protocol, both of which are derived from the general principle *don't have Alice and Bob do exactly the same thing*:

- different keys—Have the key used to authenticate Alice be different from the key used to authenticate Bob. We could use two totally different keys shared by Alice and Bob at the cost of additional configuration and storage. Alternatively we could derive the key used for authenticating Bob from the key used to authenticate Alice. For instance, Bob's key might be $-K_{\text{Alice-Bob}}$, or $K_{\text{Alice-Bob}}+1$, or $K_{\text{Alice-Bob}} \oplus \text{F0F0F0F0F0F0F0F0}_{16}$. Any of these would foil Trudy in her attempt to impersonate Alice to Bob since she would not be able to get Bob to encrypt anything using Alice's key.
- different challenges—Insist that the challenge from the initiator (Alice) look different from the challenge from the responder. For instance, we might require that the initiator challenge be an odd number and the responder challenge be an even number. Or the name of the party that created the challenge might be concatenated with the challenge before encryption, so that if the challenge from Alice to Bob was R, Bob would encrypt Bob$|R$ (the string Bob concatenated with R). This would foil Trudy, since in order to impersonate Alice to Bob, Trudy would need to get Bob to encrypt the string Alice concatenated with some number.

Notice that Protocol 11-7 did not suffer from the reflection attack. The reason is that it follows another good general principle of security protocol design: *the initiator should be the first to prove its identity.* Ideally, you shouldn't prove your identity until the other side does, but since that wouldn't work, the assumption is that the initiator is more likely to be the bad guy.

> *...if you only spoke when you were spoken to, and the other person always waited for you to begin, you see nobody would ever say anything...*
>
> —Alice (in *Through the Looking Glass*)

11.2 MUTUAL AUTHENTICATION

It is often necessary for both communicating parties to authenticate themselves to each other. For example, in Internet banking, it is imperative that a customer interacts with his/her bank and not some entity posing as the bank. Likewise, it is important that a bank be able to verify the identity of the customer. We next discuss mutual authentication using a secret key shared by both parties. We then use public key/private key pairs for mutual authentication. Finally, we design protocols that combine authentication and session key exchange.

11.2.1 Shared Secret-based Authentication

Figure 11.4(a) is a straightforward extension of the protocol for one-way authentication shown in Fig. 11.2(c). In Message 1, A communicates its identity and its challenge in the form of a nonce

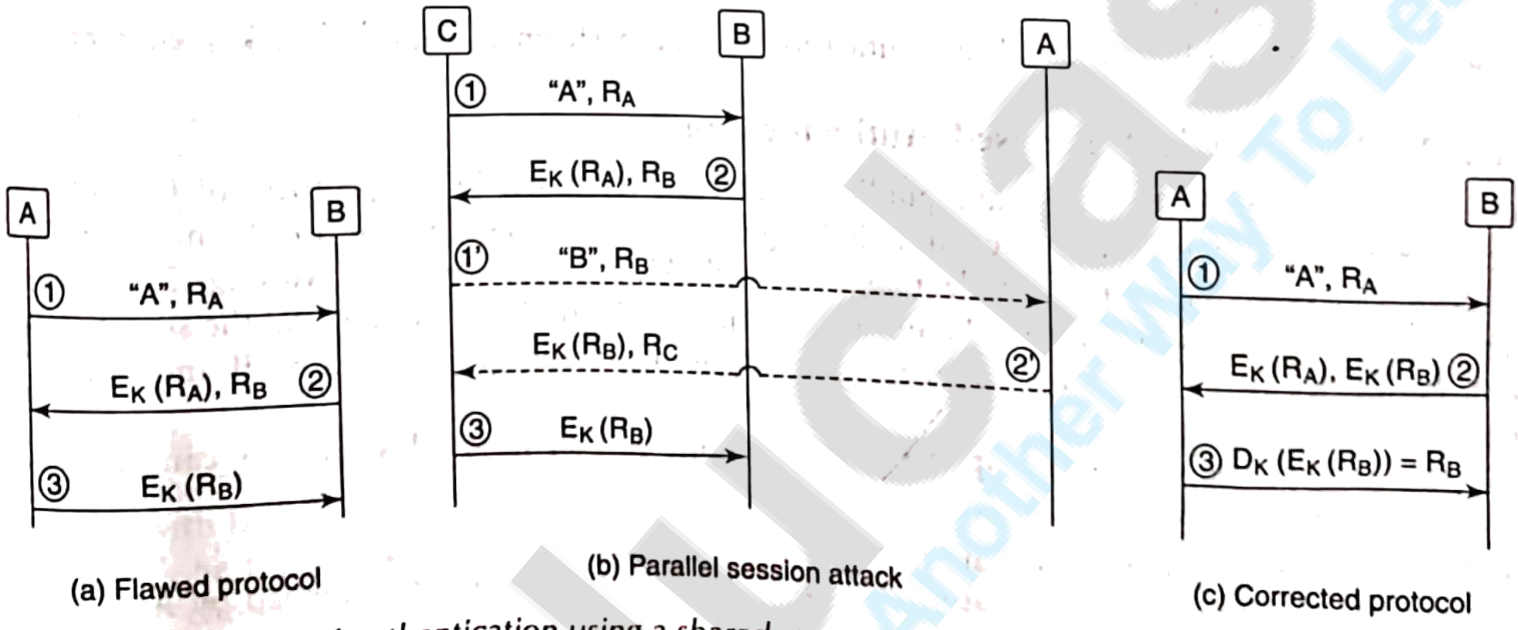


Figure 11.4 *Mutual authentication using a shared secret*

R_A. In Message 2, B responds to the challenge by encrypting R_A with the common secret, K, that A and B share. B also sends its own challenge, R_B, to A. A's response to B's challenge in the third message appears to complete the protocol for mutual authentication. While the protocol may appear sound, there are some serious flaws in it.

One attack scenario [see Fig. 11.4(b)] is as follows:

- Message 1: An attacker, C, sends a message to B containing a nonce R_A and claiming to be A.
- Message 2: B responds to the challenge with $E_K(R_A)$ and its own challenge R_B as required by the above protocol of Fig. 11.4(a).
- Message 1': Now C attempts to connect to A claiming it is B with a challenge R_B. Note that this is the *same* challenge offered to it by B in Message 2.
- Message 2': A responds to the challenge with $E_K(R_B)$ and a nonce of its own.
- Message 3: C uses A's response $E_K(R_B)$ to complete the three-message authentication protocol with B.

What has the attacker C accomplished? C has successfully impersonated A to B. Message 3 was required to complete the authentication of C (posing as A) to B. However, C could not compute the response to B's challenge since that required a computation involving the secret key, K, shared between A and B. So, C initiated the authentication protocol with A, presenting to A the *same* challenge it had received from B. A's response to the challenge in Message 2' was used by C to convince B that it was A that was trying to establish communication with him.

This attack is termed a *Reflection Attack* since a part of the message received by an attacker is reflected back to the victim. In this case, the reflected message fragment is $E_K(R_B)$. This attack is also called a *Parallel Session Attack* since the attacker, in the midst of a protocol run with one entity, opens another protocol run or session with the same or another entity.

One way of thwarting reflection attacks is for the initiator and responder to draw challenges from different disjoint sets. So, in the protocol of Fig. 11.4(a), for example, A could use nonces, which are odd numbers, while B could use nonces that are even numbers. With this modification, the R_B used in Message 2 of Fig. 11.4(b) cannot be reused in Message 1'. Another possibility is to have the initiator and responder handle challenges differently. For example, the protocol might require the responder to encrypt his challenge, while the initiator would be required to decrypt her challenge (see Fig. 11.4 (c)).

We now examine how mutual authentication can be performed using public key encryption.

11.2.2 Asymmetric Key-based Authentication

We assume that both A and B have public key/private key pairs. In the protocol of Fig. 11.5(a), each party transmits its own nonce and challenges the other to sign it. We use the notation $[m]_A$ to mean a message, m, sent in the clear together with A's signature on m. In Message 2, the string obtained by concatenating nonces R_A and R_B is signed by B. Both the nonces and the signature are sent. Nonce R_A is the challenge provided by A. R_B is the challenge provided by B and signed by A in response (Message 3).

Is this protocol flawed? There appears to be a subtle way in which this protocol can be abused as demonstrated by the following attack scenario depicted in Fig. 11.5(b).

Message 1: A initiates communication with C, sending her challenge R_A.
Message 1': C initiates communication with B using the same nonce R_A supplied by A.

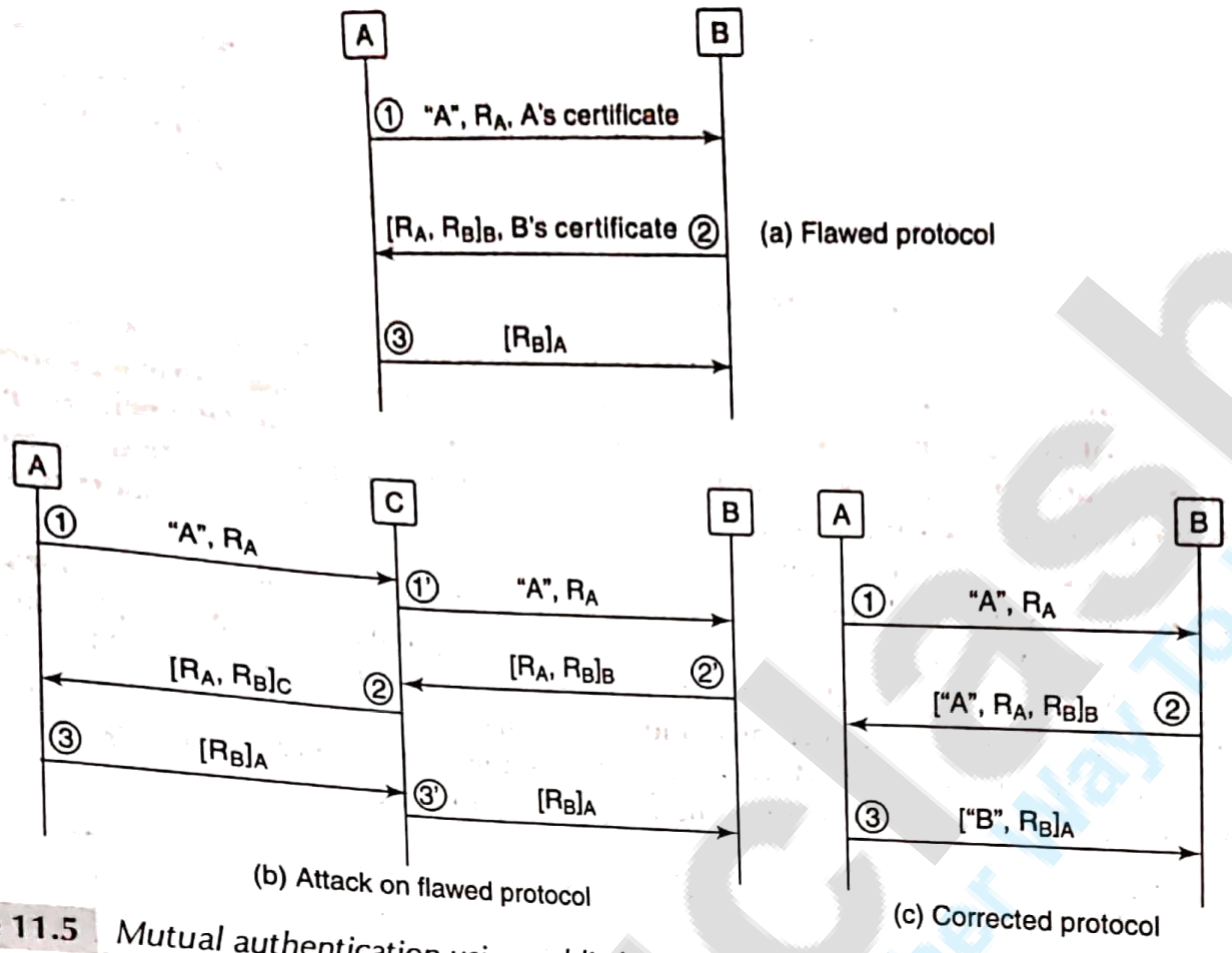


Figure 11.5 *Mutual authentication using public key cryptography*

Message 2': B responds to "A's challenge" and includes a challenge of his own, R_B.
Message 2: C responds to A's challenge and uses B's nonce, R_B, as his challenge to A.
Message 3: A responds to C's challenge (which was actually generated by B). A thus completes the mutual authentication protocol with C.
Message 4: C forwards A's response to B.

We next analyze the above message sequence by attempting to determine the intentions of the three parties.

It is clear from Fig. 11.5(b) that

- A does not intend to communicate with C. Otherwise A would not have responded, in Message 3, to C's challenge that was transmitted in Message 2.
- B wishes to communicate with A. Otherwise, B would not have responded in Message 2' to the nonce presented in Message 1'.

Note that Message 1' is sent by C but it includes A's identity. Who is C and what sort of game is he up to? C is probably known to A. After all, A intends to talk to C. But C is also the attacker here. When A initiates communication with C, the latter seizes the opportunity (after Message 1) and attempts to convince B that A intends to talk to him. B responds to what appears to be A's intention to communicate with him. Note that, in the current scenario, A may not wish to communicate with B and is not aware that C is attempting to do so on her behalf. Yet, after B receives Message 3', he feels A intends to communicate with him since Message 3' contains her signature on a nonce chosen by him.

One solution to the above problem is for the sender to include the *identity of the recipient* in all messages signed by him. This is shown in Fig. 11.5(c). Note that with this modification, Message 3 in Fig. 11.5(b) would be ["C", R_B]$_A$. If C tries to forward this message to B, the latter will smell a rat since it is C's identity that is included in the message. So B will realize that the message was intended for C, not for him.

11.2.3 Authentication and Key Agreement

In previous sections, authentication was performed using operations involving a long-term, shared secret or a private key. It is good security practice that these keys be used sparingly to minimize the probability of compromise. Also, private key operations are notoriously expensive. If the rest of the communication needs to be integrity-protected and/or encrypted (as it often is), then short-term keys for these purposes must be agreed upon. It is expedient to derive the short-term keys or *session keys* during the authentication phase itself.

Figure 11.6 shows protocols providing both mutual authentication and key agreement. Figure 11.6(a) uses secret key cryptography, while Fig. 11.6(b) uses public key cryptography. In both the figures, S_A and S_B are the contributions to the secret key by A and B, respectively. They are freshly chosen random numbers that are encrypted and sent so that they cannot be eavesdropped upon. In Fig. 11.6(a), they are encrypted in Messages 2 and 3 by the shared secret, K. In Fig. 11.6(b), they are encrypted in Messages 2 and 3 using the recipient's public key.

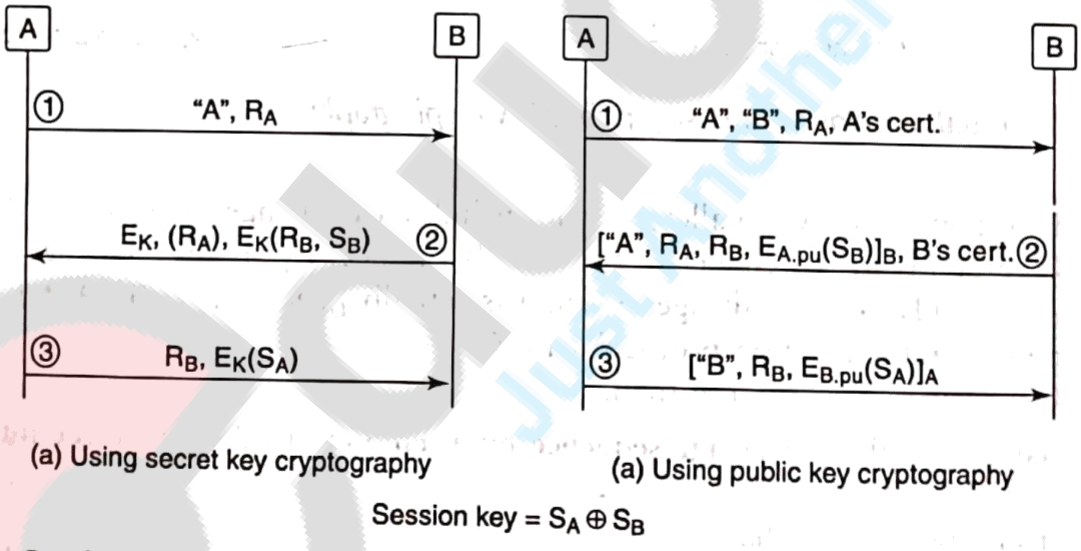


Figure 11.6 *Combined mutual authentication and key exchange*

It is possible that the session key could have been contributed exclusively by one of the communicating partners. Again however, it is good security practice that both parties contribute to the key. The key finally chosen could be a simple function of S_A and S_B, for example, $S_A \oplus S_B$.

11.2.4 Use of Timestamps

The use of nonces was introduced in Section 11.1 as a means to prevent replay attacks. Basically, each party generates a nonce which is used as a *fresh* challenge to the other party. The recipient is often expected to sign or encrypt the challenge using a secret known to only the recipient (and the sender). The key idea here is the *freshness of the nonce* – if nonces were re-used, the response to the challenge could be replayed from a previous session.

An alternative to nonces are *timestamps*. Ideally, by securely "stamping" a message with the current time, you convince the receiving party of its freshness. Figure 11.7 shows the use of timestamps in conjunction with public key cryptography for authentication.

- In Message 1, A inserts a timestamp, T_A, in her message and signs it.
- B, on receiving the message, checks whether the timestamp is sufficiently recent and then verifies the signature. He increments the received timestamp, inserts it into his response message to A, and signs the message.

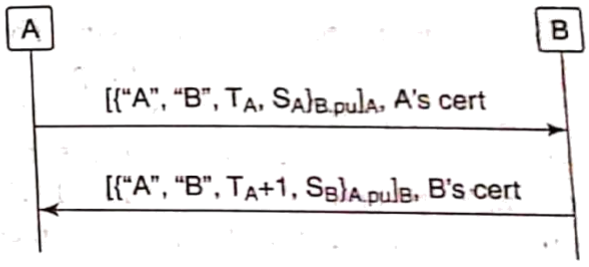


Figure 11.7 *Mutual authentication with timestamps*

The notation $\{m\}_{X.pu}$ denotes a message, m encrypted using the public key of X. If the clocks maintained by A and B are synchronized, the timestamp in Message 1 signed by A convinces B that the message was freshly created by A. The timestamp implicitly serves as A's challenge to B. By signing the incremented timestamp, B hopes to satisfy A that he is indeed responding to her message.

Q. What are pros and cons of symmetric and asymmetric key encryption? Explain a method that adapts the advantage of both techniques.

Methods for encrypting messages include the: Symmetric key encryption and Asymmetric key encryption methods. Each system has its own pros and cons which are outlined below:

Symmetric Key Encryption

Symmetric key encryption is also known as shared-key, single-key, secret-key, and private-key or one-key encryption. In this type of message encryption, both sender and receiver share the same key which is used to both encrypt and decrypt messages. Sender and receiver only have to specify the shared key in the beginning and then they can begin to encrypt and decrypt messages between them using that key. Examples include AES (Advanced Encryption Standard) and TripleDES (Data Encryption Standard).

Pros

- **Simple:** This type of encryption is easy to carry out. All users have to do is specify and share the secret key and then begin to encrypt and decrypt messages.
- **Encrypt and decrypt your own files:** If you use encryption for messages or files which you alone intend to access, there is no need to create different keys. Single-key encryption is best for this.
- **Fast:** Symmetric key encryption is much faster than asymmetric key encryption.
- **Uses less computer resources:** Single-key encryption does not require a lot of computer resources when compared to public key encryption.
- **Prevents widespread message security compromise:** A different secret key is used for communication with every different party. If a key is compromised, only the messages between a particular pair of sender and receiver are affected. Communications with other people are still secure.

Cons

- **Need for secure channel for secret key exchange:** Sharing the secret key in the beginning is a problem in symmetric key encryption. It has to be exchanged in a way that ensures it remains secret.
- **Too many keys:** A new shared key has to be generated for communication with every different party. This creates a problem with managing and ensuring the security of all these keys.
- **Origin and authenticity of message cannot be guaranteed:** Since both sender and receiver use the same key, messages cannot be verified to have come from a particular user. This may be a problem if there is a dispute.

Asymmetric/Public Key Encryption

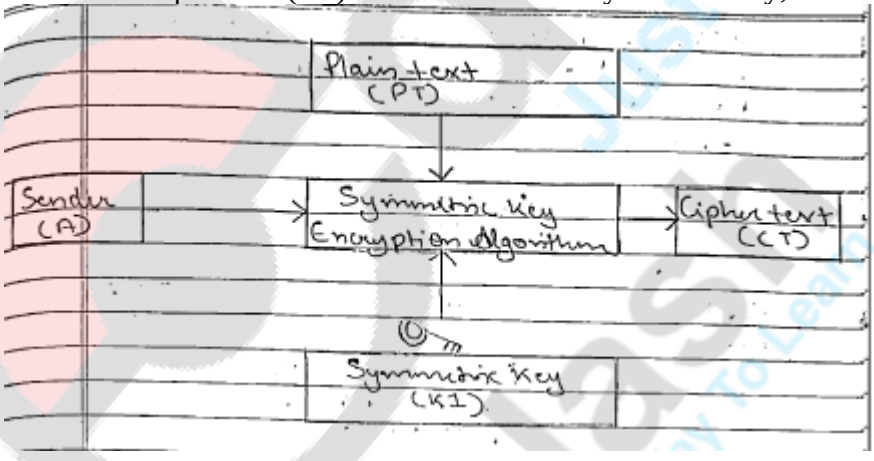
Also known as public key encryption, this method of encrypting messages makes use of two keys: a public key and a private key.The public key is made publicly available and is used to encrypt messages by anyone who wishes to send a message to the person that the key belongs to. The private

key is kept secret and is used to decrypt received messages. An example of asymmetric key encryption system is RSA.

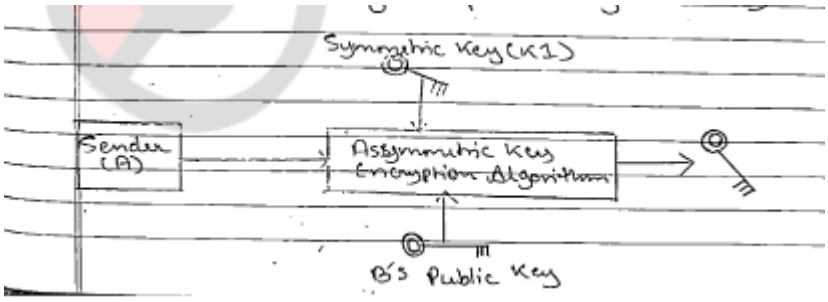
Pros
- **Convenience:** It solves the problem of distributing the key for encryption.Everyone publishes their public keys and private keys are kept secret.
- **Provides for message authentication:** Public key encryption allows the use of digital signatures which enables the recipient of a message to verify that the message is truly from a particular sender.
- **Detection of tampering:** The use of digital signatures in public key encryption allows the receiver to detect if the message was altered in transit. A digitally signed message cannot be modified without invalidating the signature.
- **Provide for non-repudiation:** Digitally signing a message is akin to physically signing a document. It is an acknowledgement of the message and thus, the sender cannot deny it.

Cons
- **Public keys should/must be authenticated:** No one can be absolutely sure that a public key belongs to the person it specifies and so everyone must verify that their public keys belong to them.
- **Slow:** Public key encryption is slow compared to symmetric encryption. Not feasible for use in decrypting bulk messages.
- **Uses up more computer resources:** It requires a lot more computer supplies compared to single-key encryption.
- **Widespread security compromise is possible:** If an attacker determines a person's private key, his or her entire messages can be read.
- **Loss of private key may be irreparable:** The loss of a private key means that all received messages cannot be decrypted.

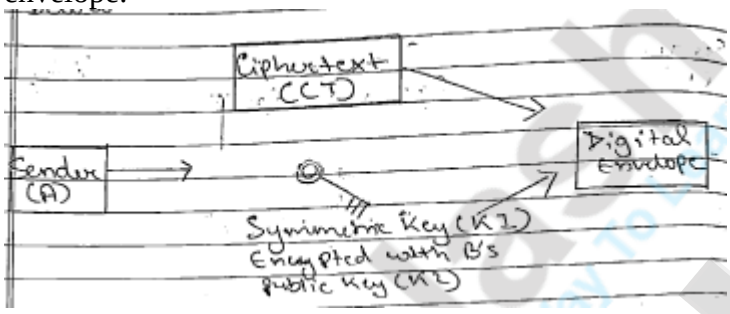
In practice, symmetric key cryptography and asymmetric key cryptography are combined to have a very efficient security solution. The way it works is as follows, assuming a is sender of message and b is its receiver.
1) A's computer encrypts the original plain-text (PT) with the help of a standard symmetric key cryptography algorithm, such as DES, IDEA, etc. This produces a cipher text message (CT). The key used in this operation (K1) is called one-time symmetric key, as it is used once and then discarded.



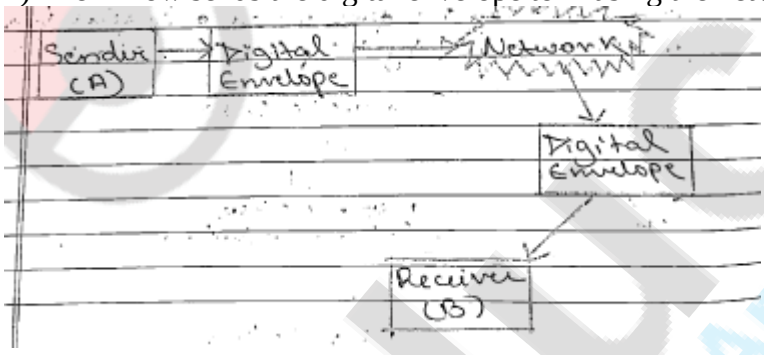
2) We have encrypted the plain-text (PT) with a symmetric key. We must now transport this one-time symmetric key (K1) to the server so that this server can decrypt the cipher text (CT) to get back the original plain-text message (PT).
It now takes K1 and encrypts it with B's public key (K2). This process is called key wrapping of symmetric key.



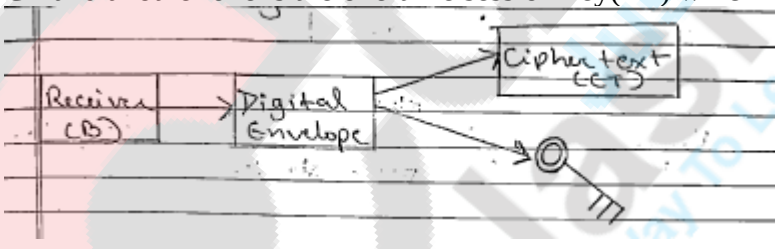
3) Now, A puts the cipher text (CT) and the encrypted symmetric key together inside a digital envelope.



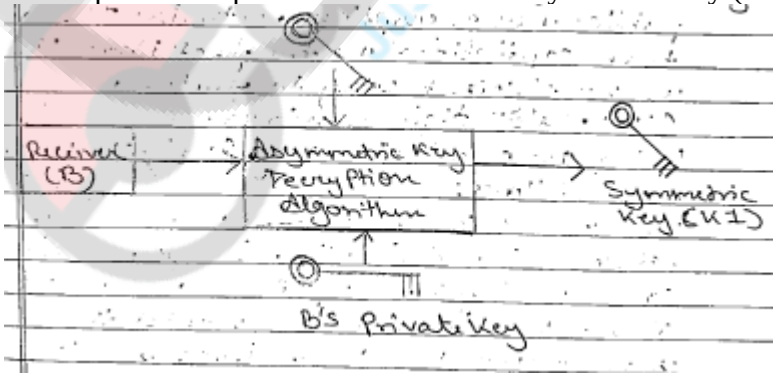
4) The A now sends the digital envelope to B using the network.



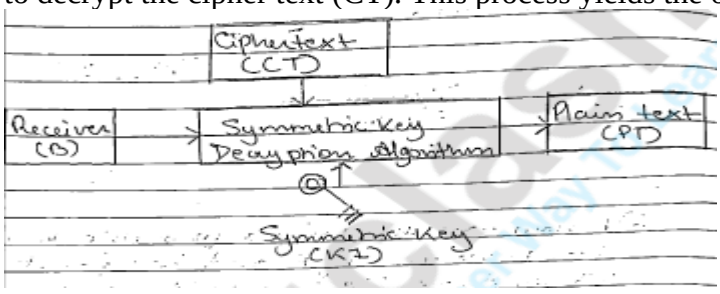
5) B receives digital envelope and opens it. After B opens this digital envelope, it gets 2 things, first is CT and another one is the one-time session key(K1) which is encrypted using B's public key (K2).



6) B now uses same antisymmetric-key algorithm as used by A and her private key (K3) to decrypt the logical box that contains the symmetric key (K1), which was encrypted with B's public key (K2). The output of this process is the one-time symmetric key (K1).



7) Finally, B applies the same symmetric-key algorithm as used by A, and uses the symmetric key K1 to decrypt the cipher text (CT). This process yields the original plain text (PT)

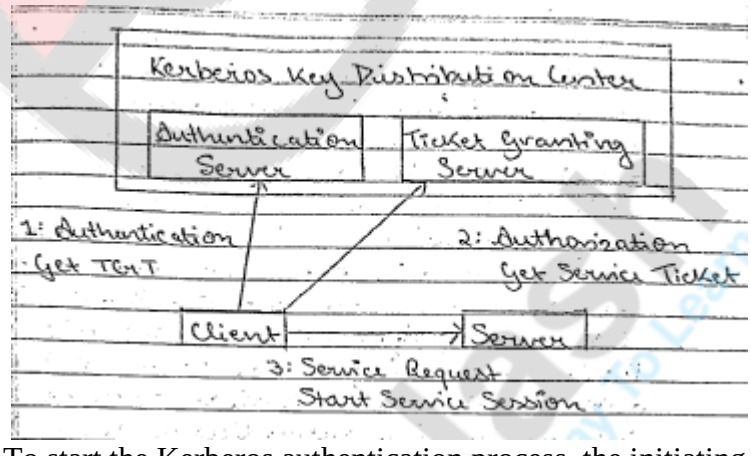


Q. Explain Kerberos as third party authentication service

Kerberos is an authentication protocol and a software suite implementing this protocol. Kerberos uses symmetric cryptography to authenticate clients to services and vice versa.

USES:
Possible uses of Kerberos include allowing users to log into other machines in a local area network, authentication for web services, authenticating email client and servers and authenticating use of devices such as printers.
Kerberos is protocol for authenticating service requests between trusted hosts across an untrusted network.

WORKING:
The Kerberos protocol uses strong cryptography so that client can prove its identity to a server and (vice versa) across an insecure network connection.
After client and server has used Kerberos to prove their identity, they can also encrypt all of their communications to assure privacy and data integrity.
Kerberos uses the concept of a ticket as a token that proves the identity of a user.
Tickets are digital documents that store session keys. They are typically issued during a login session and then can be used instead of passwords for any services.
During the course of authentication, a client receives two tickets: a ticket-granting ticket (TGT), which acts as a global identifier for a user and a session key – a service ticket, which authenticates a user to a particular service.
These tickets include time stamps that indicate an expiration time after which they become invalid. This expiration time can be set by Kerberos administrators depending on the service.
To accomplish secure authentication, Kerberos uses a trusted third party known as a Key Distribution Center (KDC), which is composed of two components, typically integrated into a single server.
1) An authentication server (AS), which performs user authentication
2) A ticket-granting server (TGS), which grants tickets to users.



To start the Kerberos authentication process, the initiating client sends a request to an authentication server for access to a service.

The initial request is sent as plaintext because no sensitive information is included in the request. The AS retrieves the initiating clients private key, assuming the initiating clients username is in the KDC database.
If the initiating clients username cannot be found in the KDC database, the client cannot be authenticated and authentication process stops.
If the clients username can be found in the KDC database, the authentication server generates a session key and a ticket-granting ticket. The ticket-granting ticket is timestamped and encrypted by the authentication server with the initiating clients password.
The initiating client is then prompted for a password, if what is entered matches the password in the KDC database, the encrypted ticket granting ticket sent from the authentication server is decrypted and used to request a credential from the ticket granting server for the desired service.
The client sends the ticket granting ticket to the ticket granting server.
The ticket granting service carries out an authentication check similar to that performed by the authentication server, but this time sends credentials and a ticket to access the requested service.
This transmission is encrypted with a session key specific to the user and service being accessed. This proof of identity can be used to access the requested kerberized service.
The timestamped ticket sent by the ticket granting service allows the requesting system to access the service using a single ticket for a specific time period without having to be re-authenticated.
Making the ticket valid for a limited time period makes it less likely that someone else will be able to use it later.

1)Discuss authentication. Explain how authentication done by the token?

What is authentication? How can be achieved with the help of token

Authentication:-

One of the key aspects of cryptography and network/Internet security is authentication. Authentication helps trust by identifying who a particular user / system is. Authentication can be defined as determining an identity to the required level of assurance. It is the first step in any cryptographic solution.

Authentication tokens:

An authentication token is an extremely useful alternative to a password. An authentication token is a small device that generates a new random value every time it is used. This random value becomes the basis for authentication. The small devices are typically of the size of small key chains, calculators or credit cards. Usually an authentication token has the following features:

- Processer
- Liquid Crystal Display(LCD) for displaying outputs
- Battery
- (Optionally) a small keypad for entering information
- (Optionally) a real-time clock

Each authentication token (i.e. each device) is pre-programmed with a unique number, called as a random seed, or just seed. The seed forms the basis for ensuring the uniqueness of the output produced by the token.

Step 1:-Creation of a token:

Whenever an authentication token is created, the corresponding random seed is generated for the token by the authentication server. This seed is stored or pre-programmed inside the token, as well as its entry is made against that user's record in the user database. Conceptually, think about this seed as the user's password (although this is technically completely different from a password). Also the user does not know about the value of the seed, unlike a password. This is because the seed is used automatically by the authentication token.

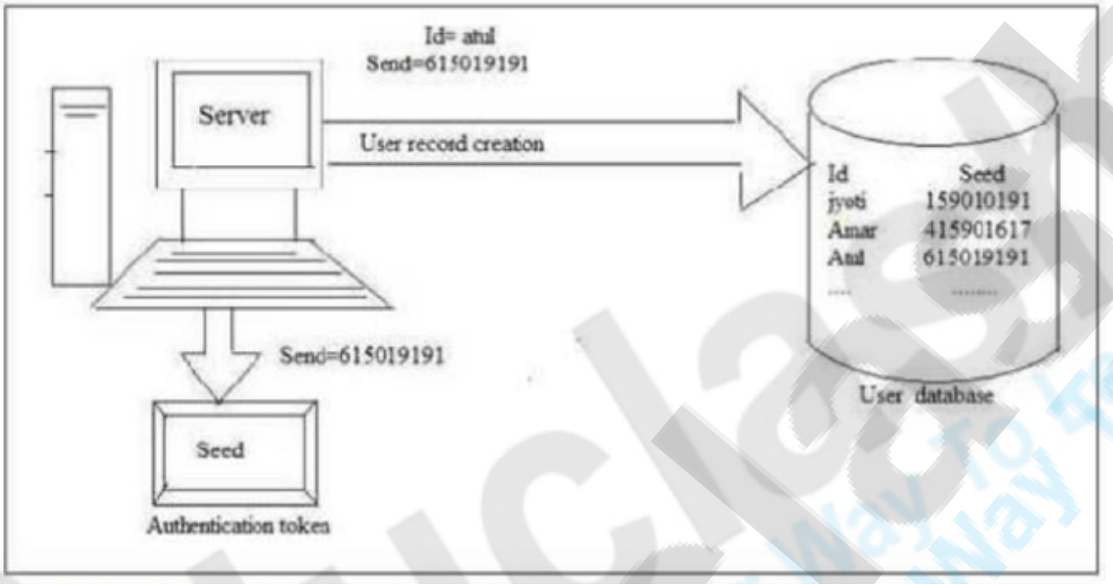
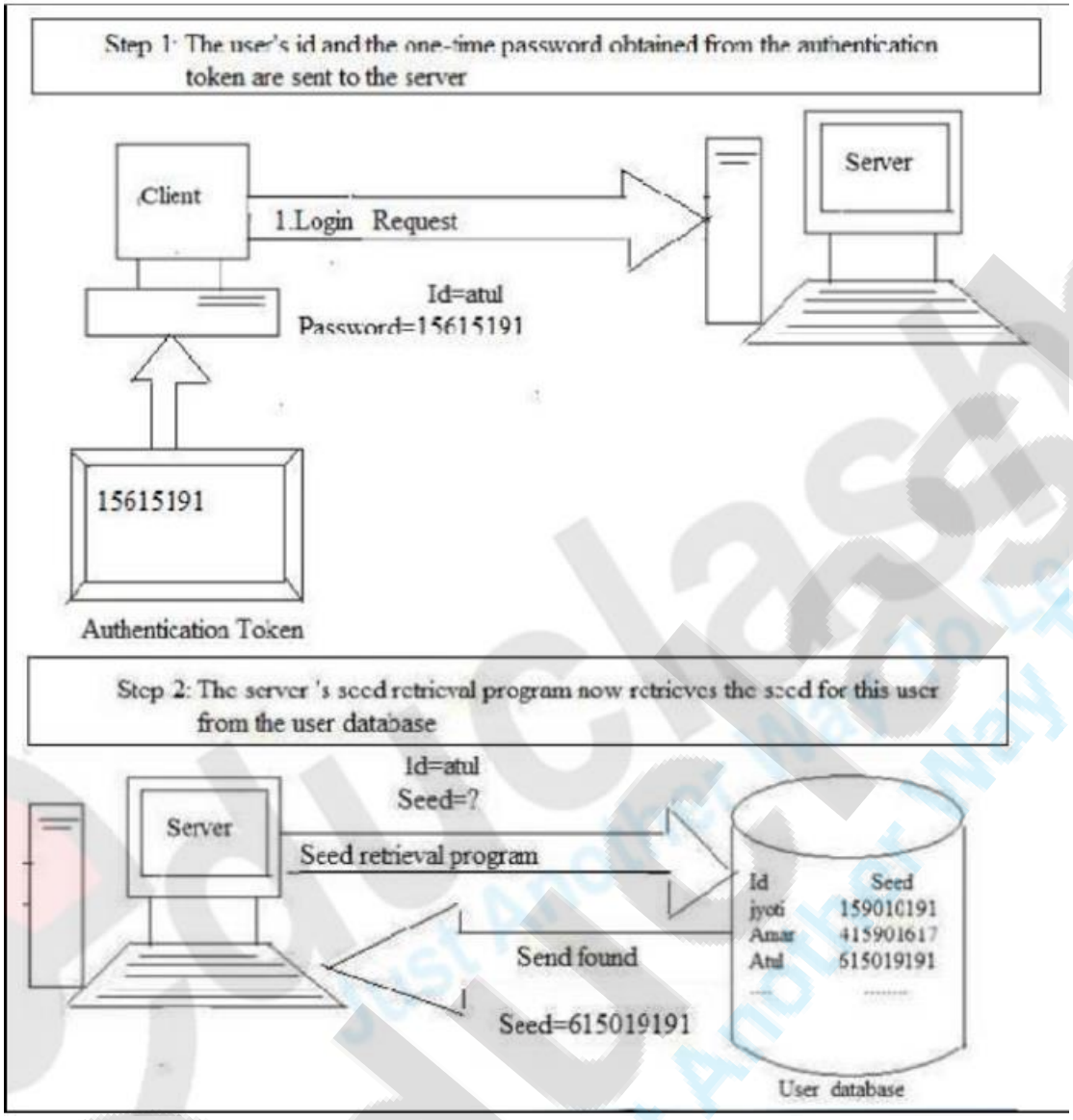
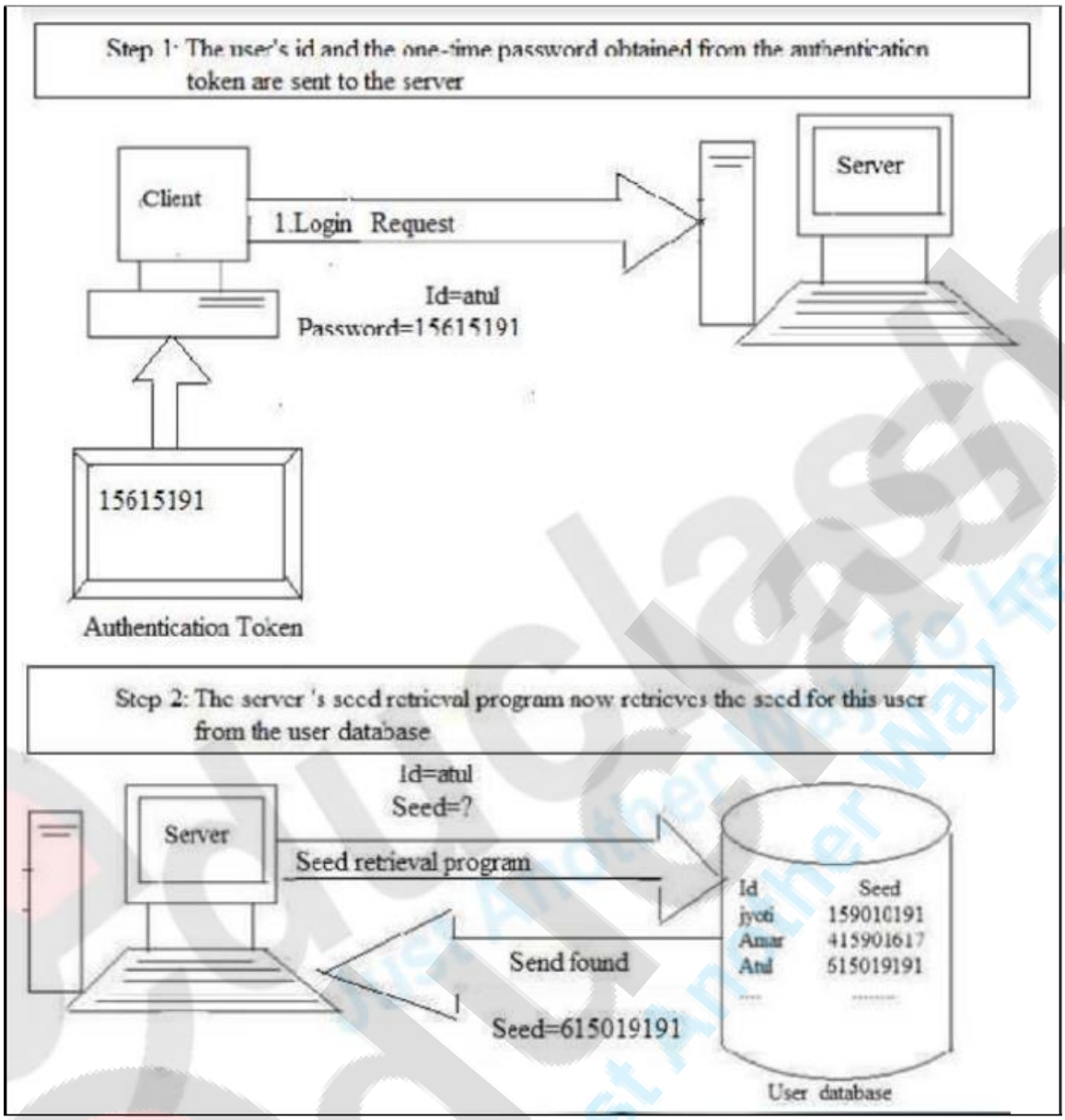


Fig:- Random seed storage in the database and the authentication token

Step 2:-Use of token:-

An authentication token automatically generates pseudorandom numbers, called as one time passwords or one-time passcodes. One-time passwords are generated randomly by an authentication token, based on the seed value that they are pre-programmed with. They are one-time because they are generated, used once , and discarded for ever. When a user wants to be authenticated, the user will be get a screen to enter the user id and the latest one-time password. For this, the user will enter the user id and the one-time password obtained from the authentication token. The user id and password travel to the server as a part of the login request. The server obtains the seed corresponding to the user id from the user database, using a Seed retrieval program. It then calls another program called as Password validation program, to which the server gives the seed and the one-time password. This program knows how to establish the relationship between the seed and the one-time password. How this is done beyond the scope of the current text. , but to explain it in simple terms, the program use synchronization techniques, to generates the same one-time password as was done by the authentication token. However, the main point to be noted here is that the authentication server can use this program to determine if a particular seed value relates to a particular one-time password or not.


Step 1: The user's id and the one-time password obtained from the authentication token are sent to the server
Client
1.Login Request
Server
Id=atul
Password=15615191
15615191
Authentication Token
Step 2: The server 's seed retrieval program now retrieves the seed for this user from the user database
Id=atul
Seed=?
Server
Seed retrieval program
Id
Seed
jyoti
159010191
Amar
415901617
Atul
615019191
Send found
Seed=615019191
User database


Step 1: The user's id and the one-time password obtained from the authentication token are sent to the server
Client
1.Login Request
Server
Id=atul
Password=15615191
15615191
Authentication Token
Step 2: The server 's seed retrieval program now retrieves the seed for this user from the user database
Id=atul
Seed=?
Server
Seed retrieval program
Id
Seed
jyoti
159010191
Amar
415901617
Atul
615019191
Send found
Seed=615019191
User database

Step 3: The server's password validation program calculate the one time password and checks the seed aganst the one-time password.

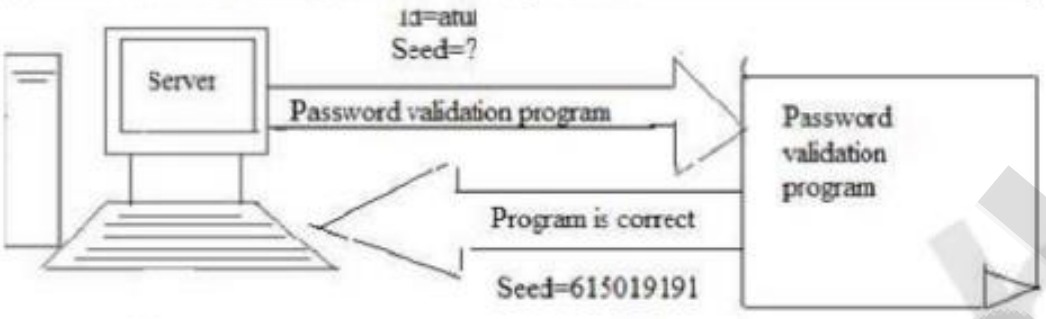


Fig:- Server validations the one-time password

A question at this stage could be, what would happen if a user loses an authentication token? Can another user simply grab it and use it? To deal with such situations, the authentication token is

generally protected by a password or a 4-digit pin. Only when this PIN is entered can the one-time password be generated. This is also the basis for what is called as multi-factor

authentication. What are this factors? There are three most common factors:

- Something that you know, e.g. a password or PIN
- Something you have , e.g. a credit card or an identity card
- Something you are ,e.g. your voice or finger print

Based on these principles, we can see that a password is a 1-factor authentication, because it is only something that you know. In contrast, authentication tokens are examples of 2-factor authentication, because here you must have something (the authentication token itself) and you must also know something (the PIN used to protect it). Someone only knowing the PIN or only having the cannot use it – both the factors are required for the authentication token to be used.

Step 3: server returns an appropriate message back to the user :- Finally, the server sends an appropriate message back to the user, depending on whether the previous operations yielded success or failure. This is shown in figure.

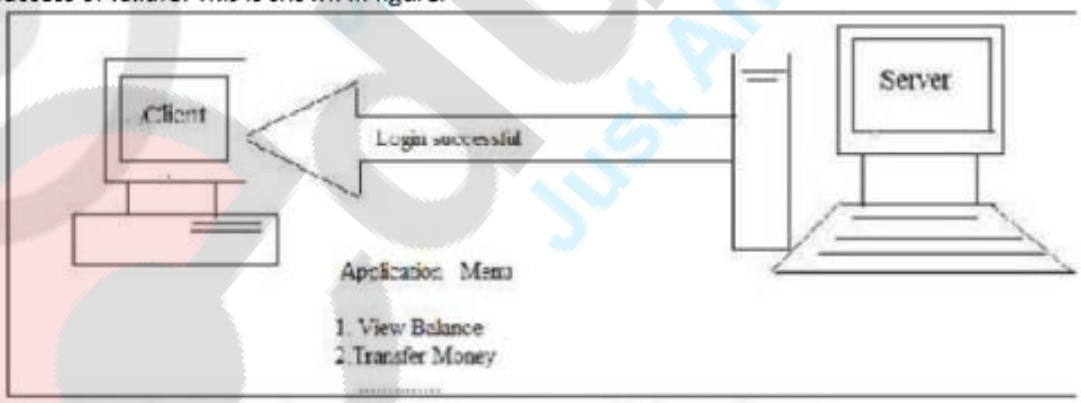


Fig:- Server sends an appropriate message back to the user

Authentication token type

There are two main types of authentication token:-

1. Challenge/Response Tokens
2. Time-based Tokens

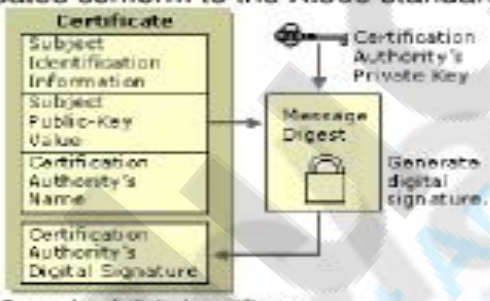
2)What are the digital certificate? Explain the stepwise process of certificate generation. How is digital certificate issued & by whom?

A digital certificate is an electronic "passport" that allows a person, computer or organization to exchange information securely over the Internet using the public keyinfrastructure (PKI). A digital certificate may also be referred to as a public key certificate.

Just like a passport, a digital certificate provides identifying information, is forgery resistant and can be verified because it was issued by an official, trusted agency. The certificate contains the name of the certificate holder, a serial number, expiration dates, a copy of the certificate holder's public key (used for encrypting messages and and the digital signature of the certificate-issuing authority (CA) so that a recipient can verify that the certificate is real.

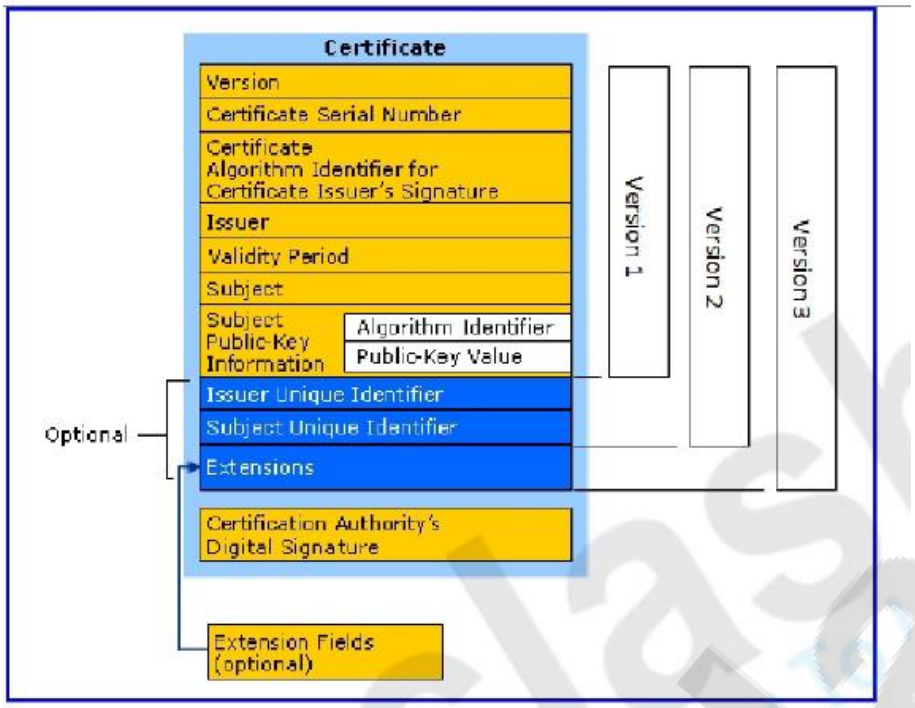
To provide evidence that a certificate is genuine and valid, it is digitally signed by a root certificate belonging to a trusted certificate authority. Operating systems and browsers maintain lists of trusted CA root certificates so they can easily verify certificates that the CAs have issued and signed. When PKI is deployed internally, digital certificates can be self-signed.

Many digital certificates conform to the X.509 standard



Example of digital certificate

Technical detail of digital certificate :
PKIX-compliant public key infrastructures, including the public key infrastructure in Windows 2000, support X.509 version 3 certificates, the contents of X.509 version 3 certificates



Content of digital certificate **Table 14.1 Description of X.509**

Certificate creation step

Asymmetric key cryptography can be a very good solution , but the exchange of public key between two parties is also a problem & this problem was solved by an idea of "Digital Certificate"

Certificate Creation Steps

Step -1 Key Generation

a) The end user can create his own private key & public key pair using some s/w . He then send the public key along with other information & proof about himself to R.A

b) Or alternatively RA generates the private as well as public key pair for user. (This happens in case when the user is not aware of generation of keys)

Step- 2 Registration

When user creates his own keys ,then he send the public key along with other information & proof about himself to R.A in a provided s/w wizard .

Note that he keeps his private key as private only.

After this he gets a requested identifier for tracking the progress of certificate request

Step 3- Verification

After registration process is complete RA has to verify user's identification

a) RA verifies the proof like address proof, email-id, ph.no, Passport /driving license...etc

b) Then he verifies private key of user by

- RA can demand at the time of sending the proof the user must send those with digital

signature . If RA can verify the signature by using user's public key ,so that RA can believe on private key of user

- Alternatively RA can encrypt documents with User's public key & send to user , which will decrypt its own private key

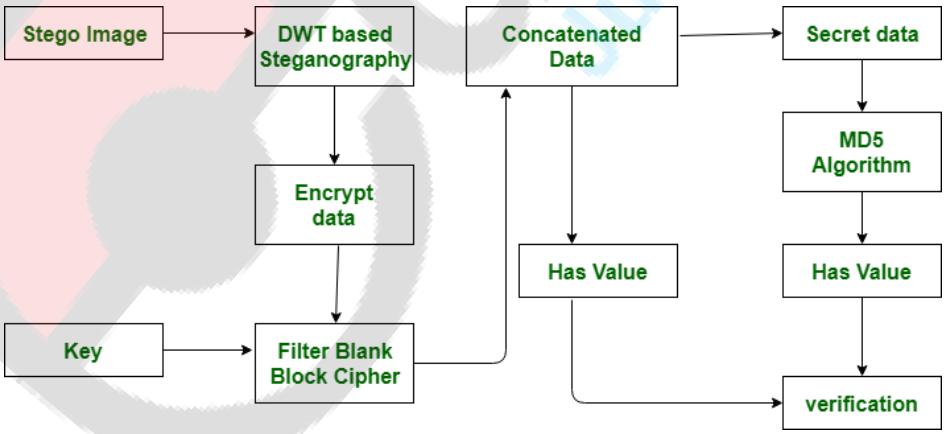
Step -4 Certificate creation
Assuming that all steps so far have been successfully ,
RA passes those documents to CA & CA will creates digital certificate in X.509 format.

CA send this certificate to user or user can download it from
CA's site

3)Define message digest. Explain MD5 & compare with SHA.
Message Digest in Information security. ***Message Digest*** is used to ensure the integrity of a ***message*** transmitted over an insecure channel (where the content of the ***message*** can be changed). The ***message*** is passed through a Cryptographic ***hash*** function. This function creates a compressed image of the ***message*** called ***Digest***.

MD5 & SHA

- Both **MD5** stands for **Message Digest** and **SHA1** stands for **Secure Hash Algorithm** square measure the hashing algorithms wherever The speed of MD5 is fast in comparison of SHA1's speed.
- However, SHA1 provides more security than MD5. The construct behind these hashing algorithms is that these square measure accustomed generate a novel digital fingerprint of knowledge or message that is understood as a hash or digest.



Some features of hash algorithms are given below:

1. The has functions can't be restrained.

2. The size of the hash (or digest) is often fastened and doesn't rely upon the scale of the info.
3. No 2 distinct information set square measure able to manufacture the same hash.

Let's see that the difference between MD5 and SHA1 which are given below:

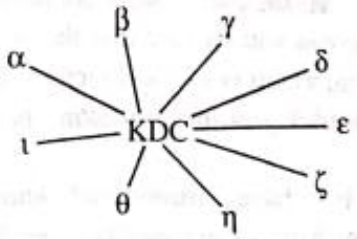
	MD5	SHA
1.	MD5 stands for Message Digest.	While SHA1 stands for Secure Hash Algorithm.
2.	MD5 can have 128 bits length of message digest.	Whereas SHA1 can have 160 bits length of message digest.
3.	The speed of MD5 is fast in comparison of SHA1's speed.	While the speed of SHA1 is slow in comparison of MD5's speed.
4.	To make out the initial message the aggressor would want 2^128 operations whereas exploitation the MD5 algorithmic program.	On the opposite hand, in SHA1 it'll be 2^160 that makes it quite troublesome to seek out.
5.	MD5 is simple than SHA1.	While SHA1 is more complex than MD5.
6.	MD5 provides indigent or poor security.	While it provides balanced or tolerable security.
7.	In MD5, if the assailant needs to seek out the 2 messages having identical message digest then assailant would	Whereas in SHA1, assailant would need to perform 2^80 operations which is greater

need to perform 2^64 operations. than MD5.

5)KDC

9.7.1 KDCs

One way to make things manageable is to use a trusted node known as a **Key Distribution Center** (KDC). The KDC knows keys for all the nodes. If a new node is installed in the network, only that



new node and the KDC need to be configured with a key for that node. If node α wants to talk to node β, α talks to the KDC (securely, since α and the KDC share a key), and asks for a key with which to talk to β. The KDC authenticates α, chooses a random number $R_{\alpha\beta}$ to be used as a key to be shared by α and β for their conversation, encrypts $R_{\alpha\beta}$ with the key the KDC shares with α and gives that to α. The KDC also encrypts $R_{\alpha\beta}$ with the key the KDC shares with β and gives that to β, with the instruction that it is to be used for conversing with α. (Usually, the KDC will not bother to actually transmit the encrypted $R_{\alpha\beta}$ to β but rather will give it to α to forward to β.) The encrypted message to β that the KDC gives to α to forward is often referred to as a **ticket**. Besides containing $R_{\alpha\beta}$, the ticket generally contains other information such as an expiration time and α's name. We'll discuss protocols involving KDCs in §11.4 *Mediated Authentication (with KDC)*.

KDCs make key distribution much more convenient. When a new user is being installed into the network, or when a user's key is suspected of having been compromised, there's a single location (the KDC) that needs to be configured. The alternative to using a KDC is installing the user's information at every server to which the user might need access. There are some disadvantages to KDCs, though:

- The KDC has enough information to impersonate anyone to anyone. If it is compromised, all the network resources are vulnerable.
- The KDC is a single point of failure. If it goes down, nobody can use anything on the network (or rather, nobody can start using something on the network—keys previously distributed can continue to be used). It is possible to have multiple KDCs which share the same database of keys, but that means added complexity and cost for extra machines and replication protocols, and added vulnerability, since there are now more targets that need to be protected.
- The KDC might be a performance bottleneck, since everyone will need to frequently communicate with it. Having multiple KDCs can alleviate this problem.

9.7.2 Certification Authorities (CAs)

Key distribution is easier with public key cryptography. Each node is responsible for knowing its own private key, and all the public keys can be accessible in one place. But there are problems with public keys as well. If, for instance, all the public keys are published in *The New York Times*, or stored in the directory service, how can you be sure that the information is correct? An intruder, Trudy, might have overwritten the information in the directory service or taken out her own ad in *The New York Times*. If Trudy can trick you into mistaking her public key for Alice's, she can impersonate Alice to you.

The typical solution for this is to have a trusted node known as a **Certification Authority (CA)** that generates **certificates**, which are signed messages specifying a name (Alice) and the corresponding public key. All nodes will need to be preconfigured with the CA's public key so that they can verify its signature on certificates, but that is the only public key they'll need to know *a priori*. Certificates can be stored in any convenient location, such as the directory service, or each node can store its own certificate and furnish it as part of the authentication exchange. CAs are the public key equivalent of KDCs. A CA or a KDC is the single trusted entity whose compromise can destroy the integrity of the entire network. The advantages of CAs over KDCs are:

- The CA does not need to be on-line. It might be in a locked room protected by a scary-looking guard. Perhaps only a single very trusted individual has access to the CA. That person types the relevant information at the CA, and the CA writes a floppy disk with the new user's certificate, and the floppy disk can be hand-carried to a machine that's on the network. If the CA is not on-line it cannot be probed by curious intruders.
- Since the CA does not have to be on-line or perform network protocols, it can be a vastly simpler device, and therefore it might be more secure.
- If the CA were to crash, the network would not be disabled (as would be the case with a KDC). The only operation that would be impacted is installing new users (until things start expiring, such as certificates or Certificate Revocation Lists—see §9.7.3 *Certificate Revocation*). So it's not as essential to have multiple CAs.
- Certificates are not security-sensitive. If they are stored in a convenient, but potentially insecure, location like the directory service, a saboteur might delete certificates, which might prevent network access by the owners of those certificates, but the saboteur cannot write bogus certificates or modify certificates in any way, since only the CA can generate signatures.
- A compromised CA cannot decrypt conversations, whereas a compromised KDC that sees a conversation between any two parties it serves can decrypt the conversation. A compromised CA can fool Alice into accepting an incorrect public key for Bob, and then the CA can impersonate Bob to Alice, but it will not be able to decrypt a conversation between the real Alice and the real Bob. (It's still really bad for a CA to be compromised, but we're just saying it's not quite as bad as compromise of a KDC.)

> *Why do you security people always speak of compromise as if it's a bad thing? Good engineering is all about compromise.*
>
> —overheard at a project review

5)MD5 algoithm

To preserve the integrity (i.e. to ensure that a message has not been tampered with after it leaves the sender but before it reaches the receiver) of a message, the message is passed through a message digest algorithm also called as hash function.

Thus we perform a hashing function (or message digest algorithm) over block of data to produce its hash or message digest, which is smaller in size than the original message. This concept is shown in figure:

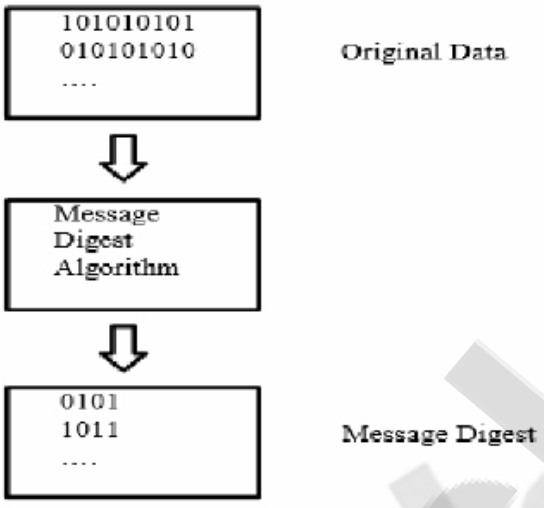


Figure: Message Digest Concept.

MD5

- MD5 is a message digest algorithm developed by Ron Rivest.
- MD5 actually has its roots in a series of message digest algorithm, which were predecessors of MD5, all developed by Ron Rivest.
- The original message digest algorithm was called as MD. He soon came up with next version MD2, MD3 & MD4 but those were quite weak, failure etc. consequently Ron Rivest released MD5.
- MD5 is quite fast & produces 128 bit message digest. MD5 has been able to successfully defend itself against collisions.

Working of MD5:

Step 1: Padding:

- The first step in MD5 is to add padding bits to the original message. The aim of this step is to make the length of the original message equal to a value, which is 64 bits less than an exact multiple of 512.
- Note that padding is always added, even if the message length is already 64 bits less than a multiple of 512. Thus, if the message were already of length say 448 bits, we will add a padding of 512 bits to make its length 960 bits. Thus, the padding length is any value between 1 and 512.

The padding process is shown in below:

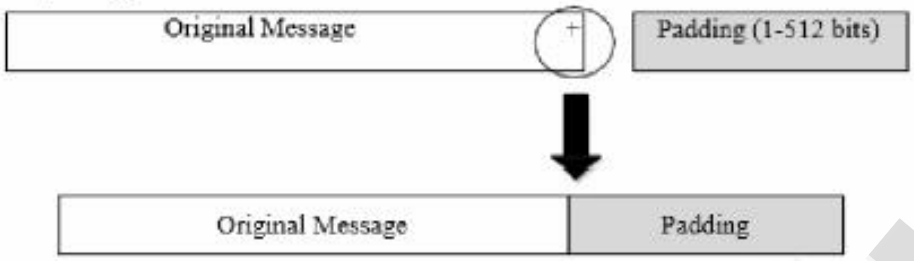


Figure: Padding Process

Step 2: Append length:

- After padding bits are added, the next step is to calculate the original length of the message and add it to the end of message, after padding.
- This length of the original message is now expressed as a 64-bit value and these 64 bits are appended to the end of the original message + padding. This is shown in Figure below.
- Note that if the length of the message exceeds 2 bits (i.e. 64 bits are not enough to represent the length, which is possible in the case of a really long message), we use only the low-order 64 bits of the length. That is, in effect, we calculate the length mod 2 in that case.
- We will realize that the length of the message is now an exact multiple of 512. This now becomes the message whose digest will be calculated.

Figure: Append Length

Step 3: Divide the input into 512-bit blocks

- Now, we divide the input message into blocks, each of length 512 bits. This shown below:

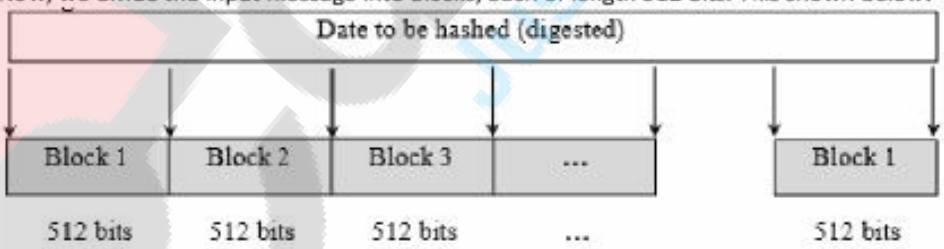


Figure: Data is divided into 512-bits blocks

Step 4: Initialize chaining variables:

- In this step, four variables (called as chaining variables) are initialize. They are called as A, B, C and D. Each of these is a 32-bit number.
- The initial hexadecimal values of these chaining variables are shown below:

A	Hex	01	23	45	67
B	Hex	89	AB	CD	EF
C	Hex	FE	DC	BA	98
D	Hex	76	54	32	10

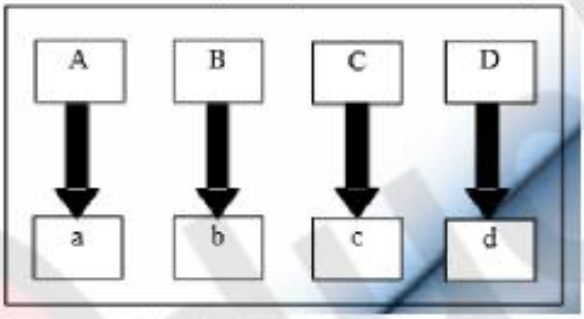
Figure: Chaining Variables

Step 5: Process blocks:

- After all the initializations, the real algorithm begins .There is a loop that runs for as many 512-bit block as are in the message.

Step 5.1:

- Copy the four chaining variables into four corresponding variables, a, b, c and d (note the smaller case). Thus, we now have a = A, b = B, c = C and d = D. This is shown below:



- Actually, the algorithm considers the combination of a, b, c and d as a 128-bit single register (which we shall call as abcd). This register (abcd) is useful in the actual algorithm operation for holding intermediate as well as final results. This is shown below:

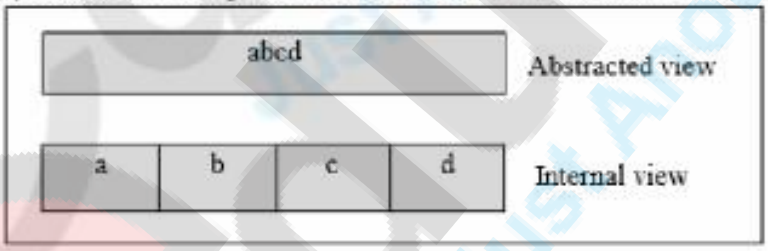


Figure: Abstracted view of the chaining variables

Step 5.2:

- Divide the current 512-bit block into 16 sub-blocks. Thus, each sub-block contains 32 bits, as shown below:

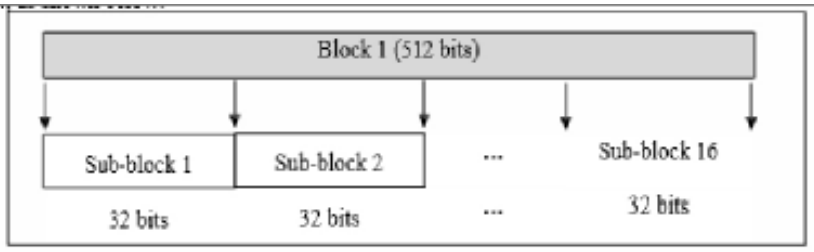


Figure: Sub-blocks within a block

Step 5.3:

- Now, we have four rounds. In each round, we process all the 16 sub-blocksSte p 5.3: belonging to a block. The inputs to each round are: (a) all the 16 sub-blocks, (b) the variables a, b, c, d and (c) some constants, designated as t. This is shown as below:

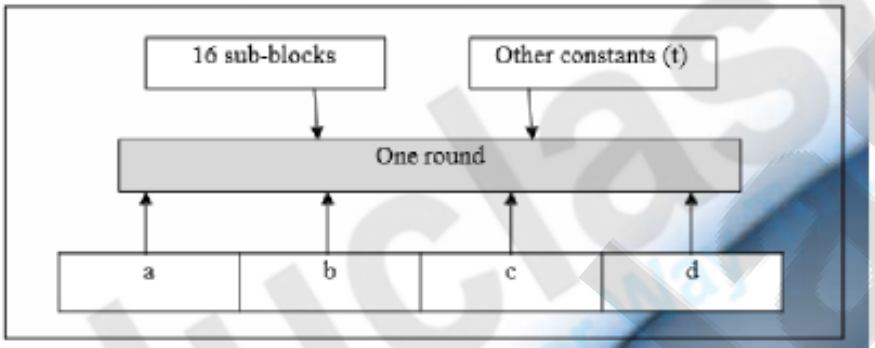


Figure: Conceptual process within a round

All the four rounds vary in one major way: Step 1 of the four round has different processing. The other steps in all the four rounds are the same.

1. A process P is first performed on b, c and d. This process P is different in all the four rounds.
2. The variable a is added to the output of the process P (i.e. to the register abcd).
3. The message sub-block M[i] is added to the output of Step 2 (i.e. to the register abcd).
4. The constant t[k] is added to the output of Step 3 (i.e. to the register abcd).
5. The output of Step 4 (i.e. the contents of register abcd) is circular-left shifted by s bits. (The value of s keeps changing, as we shall study).
6. The variable b is added to the output of Step 5 (i.e. to the register abcd).
7. The output of Step 6 becomes the new abcd for the next step.

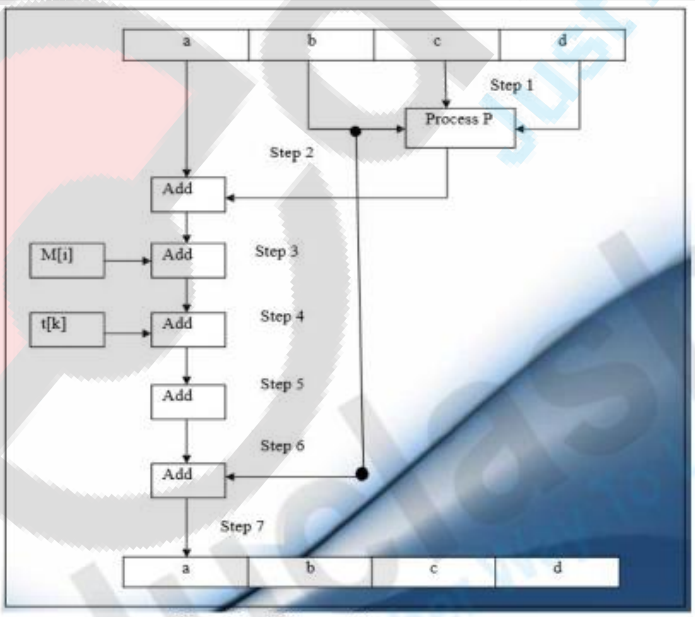
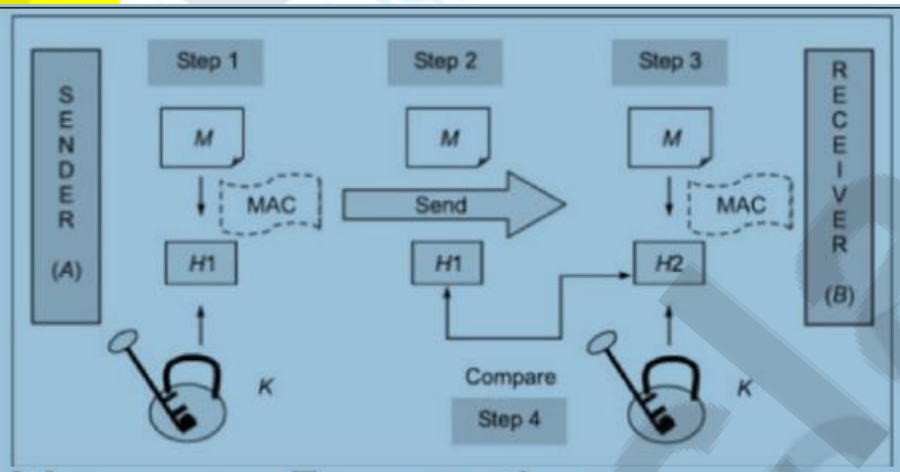


Figure: One MD5 operation

6)MAC

- Similar to Message Digest
- Shared Symmetric (Secret) key is used for encryption
- Message authentication is concerned with:
  - protecting the integrity of a message
  - validating identity of originator
  - non-repudiation of origin (dispute resolution)
- consider the security requirements



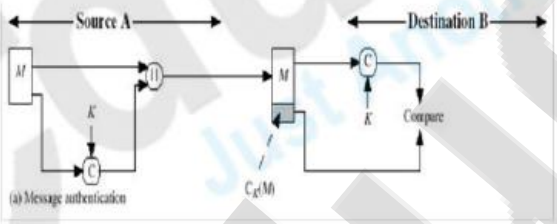
Message Encryption

- MAC generation of message using shared symmetric (secret) key.
- Sends original message and MAC(H1)
- At receiver end, it receives original message and MAC
- Receiver calculate MAC(H2) using key and original message.
- Compare H1 & H2
  - If **H1!=H2** then, Message altered
  - If **H1==H2** then, Message not changed

Message Authentication Code (MAC)

- Generated by an algorithm that creates a small fixed-sized block
  - depending on both message and some key
  - like encryption though need not be reversible
- appended to message as a **signature**
- receiver performs same computation on message and checks it matches the MAC
- provides assurance that message is unaltered and comes from sender

Message Authentication Code

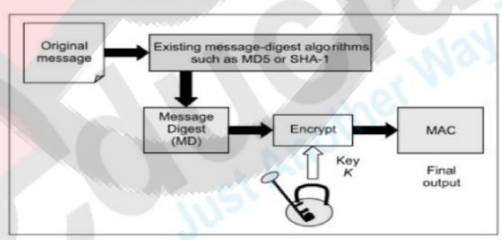


7)HMAC

HMAC

- HMAC stands for -**Hash Message Authentication Code**
- Mandatory for security implementation for Internet Protocol security.
- Idea of HMAC is to reuse existing Message-Digest algorithms(such as MD5,SHA-1..)
- Uses shared symmetric key to encrypt message digest.

HMAC CONCEPT



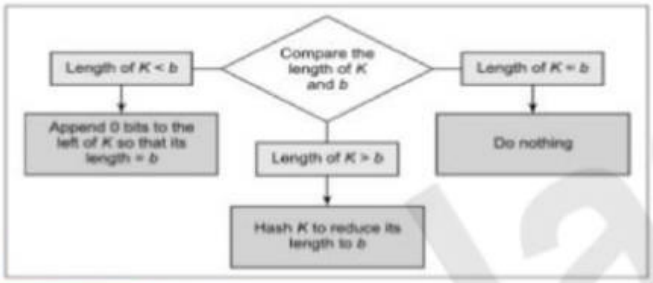
WORKING OF HMAC

- Variables used in HMAC
  - MD = the message digest/hash function used(e.g. MD5,SHA-1,etc.)
  - M = the input message whose MAC is to be calculated.
  - L = the number of blocks in the message M.
  - b = the numbers of bits in each block.
  - K = the shared symmetric key to be used in HMAC.
  - ipad = A string 00110110 repeated b/8 times.
  - opad = A string 01011010 repeated b/8 times.

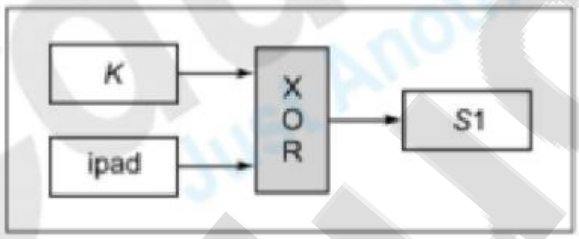
WORKING OF HMAC

- STEP-1 Make the length of K equal to b.
- STEP-2 XOR K with Ipad to produce S1.
- STEP-3 Append M to S1.
- STEP-4 Message-digest algorithm.
- STEP-5 XOR K with opad to produce S2.
- STEP-6 Append H to S2.
- STEP-7 Message-digest algorithm.

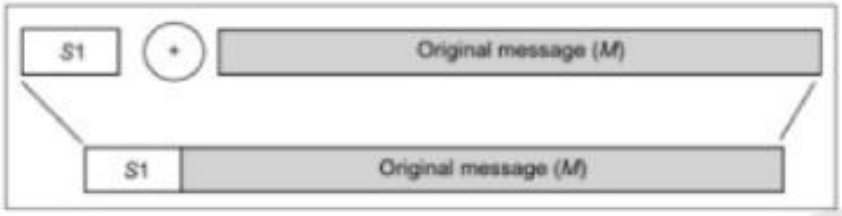
- STEP-1 Make the length of K equal to b.
  - **If length of K<b :** add 0 bit as required to the left of k
  - **If length of K=b :** In this case, we do not take any actic proceed to step 2.
  - **If length of K>b :** we need to trim k, for this, we pass K the message-digest algorithm(H) selected for this particul instance of HMAC



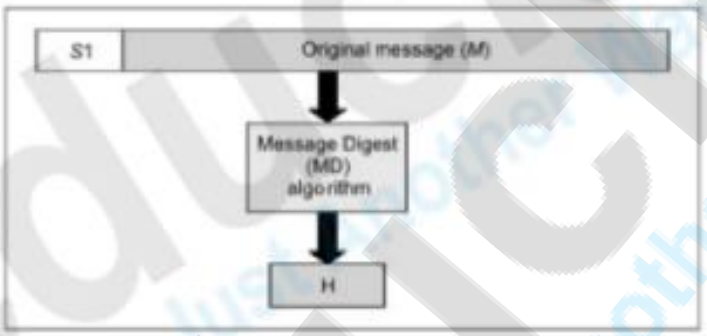
- STEP-2 XOR K with Ipad to produce S1
  - XOR K (the output of step 1) and ipad to produce a variable called S1.



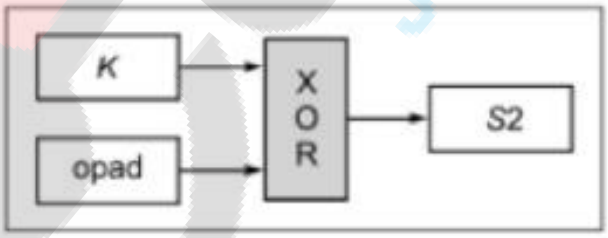
- STEP-3 Append M to S1
  - Take the original message (M) and simply append it to the end of S1.



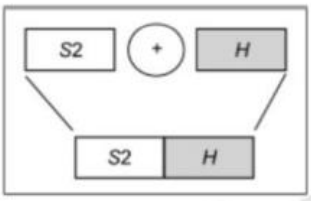
- STEP-4 Message-digest algorithm
  - The selected message-digest algorithm (e.g. MD5,SHA-I, etc.) is applied to the output of step 3.



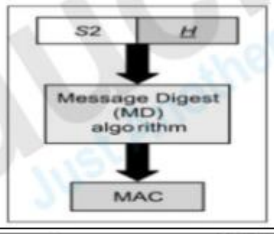
- STEP-5 XOR K with opad to produce S2
  - XOR K (the output of step 1) with opad to produce a variable called as S2.



- STEP-6 Append H to S2
  - Append the message digest calculated in step 4 to the end of S2.



- STEP-7 Message-digest algorithm
  - the selected message-digest algorithm (e.g. MD5, SHA-I, etc.) is applied to the output of step 6 (i.e. to the concatenation of S2 and H). This is the **Final MAC** that we want



Disadvantages of HMAC

1. Key exchange is main issue
2. Somehow the key-exchange problem is resolved, HMAC cannot be used if the number of receivers is greater than one.
3. If multiple parties share the same symmetric key. How does a receiver know that the message was prepared and sent by the sender
4. Replay of Message

Types of authentication.

1) Password based Authentication
2) Address based Authentication
3) Cryptographic Authentication
4) Smart Cards
5) Biometrics
6) Mutual authentication

i) To secure a network the first step is to avoid unauthorized access to networks
ii) This can be achieved with any type of authentication mechanisms.
iii) When one node wants to communicate with other nodes in a network in a secure manner they use an authentication mechanism
iv) The node that wants to communicate has to ~~improve~~ it's identity in the network so that it's right to access the network resources can be determined
v) Knowledge based authentication is based on "what you know" i.e. what the user knows eg. username, password, keys etc.
vi) Artifact based Authentication is based on "what you possess" i.e. certificates, tokens, smart cards etc.

vii) Biometric Based Authentication is based on "what you are" i.e. what the user inherits for eg. biometric techniques.

(0) Password Based Authentication:-

Setup

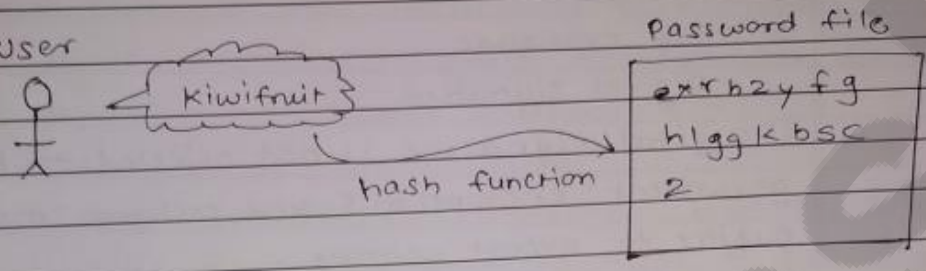
- User chooses password
- Hash of password stored in password file.

Authentication :-

- User logs into system, supplies password
- System computes hash, compares to file.

Basic Password scheme :-

- hash f^n $h: \text{Strings} \rightarrow \text{Strings}$
- User password stored as h(password) in password file.
- when user enters password
  - > System computes h(password)
  - > Compares with entry in password file
  - > If password entered by ~~file~~ user is equal to password from password file, then user is authenticated.
  - > else, it gives an error, saying that Username or password is incorrect



(b) Address based Authentication

1) Uses the address of the node in the network
2) MAC address or IP address
3) Allow only preconfigured set of MAC or IP address to access the network
4) Usually implemented in the switch / Router
5) Loop holes.
   > simple one-to-one mapping between a node and a user
   > So doesn't really authenticate a user
   > weak to MAC spoofing & IP spoofing attack

(c) Cryptographic Authentication

1) A provider can authenticate to a verifier by providing knowledge of a private key.
2) The private key may

- encryption
- key exchange
- Digital signature

3) Digital signature is not objected to by any government; encryption & key exchange may be subject to export controls.

4) Cryptographic Authentication by digital signal
   i) verifier generates random nonce
   ii) A challenge is constructed from material including verifier's nonce
   iii) The provider signs challenge with its private key.

5) Digital signature cryptosystem:
   i) RSA
      - Dual purpose: encryption, signature
   ii) DSA
      - Designed by NSA to provide signature but no encryption
   iii) ECDSA
      - elliptic curve version of DSA

(D) Smart cards

i) smart cards are hardware device that provide much secure authentication for storing & transferring the important information
ii) they are the size of credit cards containing a small chip which stores the private key & a copy of a certificate

> iii) A PIN (personal identification number) is used in association with smard cord to provide more secure authentication

> iv) Working of smard cords.

(I) The authentication method used in smart card is challenge Response Type of authentication

(II) When user inserts his smart card in a card reader, The program that is stored in client system asks the user for his unique PIN.

(III) The user enters the PIN & if the PIN is correct the communication between the client application & smart card starts.

(IV) The challenge response procedure takes place bet^n the client & the server.

(V) Private key on the card is used to encrypt the data & encrypted data is transferred to the server

(VI) The public key stored on server is used to decrypt the data.

(VII) If the data gets successfully decrypted, the user is authenticated.

> Advantages of smart card

This is very secure authentication mechanism because

(a) The process works on two-factor authentication - "what you know" (PIN) & "what you have" (smart card or private key).

(b) Brute force attack & dictionary attacks don't work here as only limited no. of PIN entries are allowed for smart card holder.

> Applications of smart cards.
- electronic toll machine
- financial services
- Healthcare
- cellular phone
- set-top boxes
- secure network access.

(E) Biometrics :-

> a biometric authentication system attempts to authenticate an individual based on his/her unique physical characteristics

> Individual characteristics can be static characteristics i.e. fingerprints, hand geometry, facial characteristics, & retinal & iris patterns.

> dynamic characteristics i.e. voice print and signatures

> Advantage → can't be disclosed, lost, forgotten

> Disadvantage :- cost, installation, maintainance

> Physiological:-

1) Iris, 2) Fingerprint (including nail)
3) Hand (including knuckle, palm, vascular)
4) face 5) voice 6) Retina
7) DNA 8) even odor, sweat pore, lips, earlobe

> Behavioral pattern
- signature
- keystroke
- voice

→ Physiological Biometrics

1) Fingerprint :- A fingerprint is a pattern of ridges on surface of fingerprint

> Two premises for fingerprint identification
- fingerprint details are permanent
- fingerprints are unique

2) Face :-

> facial characteristics are most common means of human-to-human identification

> we can define characteristics based on relative location & shape of key facial features such as eyes, eyebrow, nose, lips, chin shape

3) Retinal Pattern - The pattern formed by veins beneath the retinal surface is unique & ∴ suitable for identification

4) Hand - Hand geometry ~~features~~ systems identify features of hand, including shape & length & width of fingers

5) Iris - The detailed structure of iris also a ~~ung~~ unique physical characteristic in authentication

→ Behavioral Biometric

1) Signature - each individual has unique style of handwriting & ~~als~~ this reflected especially in the signature which is typically frequently written sequence. However multiple signature samples from a single individual will not be identical. This complications the task of developing a computer representation of signature that can be matched to future samples

2) Voice -

- Voice patterns are more closely tied to the physical & anatomical characteristic of the speaker
- There is still variation from sample to sample over time from the same speaker complicating biometric ~~reg~~ recognition task.