Message Authentication Code (MAC)
- Key-dependent one-way hash function
- Only someone with a correct key can verify the hash value
- Easy way to turn one-way hash function into MAC is to encrypt hash value with symmetric algorithm

Digital Signatures
- Proof of authorship or agreement with contents of a document
- Signature is authentic (no one but Alice could have signed a document with her signature)
- Signature is unforgeable
- Signature is not reusable
- Signed document in unalterable
- Signature cannot be repudiated

Arbitrated Signatures
- Alice wants to sign a message and send it to Bob
- Alice and Trent share $K_A$, Bob and Trent share $K_B$
- Alice encrypts the message with $K_A$ and sends it to Trent
- He decrypts it, adds a statement that he has received this from Alice, encrypts it with $K_B$ and sends it to Bob
- Bob can also prove to Carol that he received the message from Alice but he needs to involve Trent again

Public-Key Signatures
- Alice encrypts the document with her private key
- Sends the signed document to Bob who decrypts it with her public key
- This signature is reusable, Bob can take the same message and claim he received it multiple times → add timestamps
- Signing the whole document with public key is slow → sign a hash of the document produced by one-way hash function

Terminology

\[
\begin{align*}
\text{Alice} & \quad \text{M} \quad S_A(\text{M}) \quad \text{MLS} \quad \text{Bob} \\
& \quad \text{K1} \quad \text{K2}
\end{align*}
\]

M – message
K1 – Alice’s private key
S_A(M) – message M is signed by Alice
K2 – Alice’s public key
V_A(M) – signature of the message M (generated by Alice) is verified

Digital Signatures With Encryption
- Combining digital signatures with public-key cryptography we gain security and authenticity
- Alice first signs the message (or message digest) with her private key: $S_A(M)$
- Alice encrypts the signed message with Bob’s public key: $E_B(S_A(M))$
- Bob decrypts the message with his private key: $D_B(E_B(S_A(M))) = S_A(M)$
- Bob verifies Alice’s signature $V_A(S_A(M)) = M$
Digital Signatures With Encryption

- Only Bob can decrypt the message (security) and he knows that Alice has sent the message (authenticity)
- If Alice encrypted message digest he can also verify that the message has not been changed
- If Alice added timestamps he can also verify that the message has not been replayed

Man-in-the-Middle Attack
On Key Exchange

- Alice sends to Bob her public key Pub(A)
- Mallory captures this and sends to Bob Pub(M)
- Bob sends to Alice his public key Pub(B)
- Mallory captures this and sends to Alice Pub(M)
- Now Alice and Bob correspond through Mallory who can read all their messages

Key Exchange With Interlock Protocol

- First four steps are the same
  - Alice to Bob her public key Pub(A)
  - Mallory captures this and sends to Bob Pub(M)
  - Bob sends to Alice his public key Pub(B)
  - Mallory captures this and sends to Alice Pub(M)
  - Alice encrypts a message in Pub(M) but sends half to Bob – Mallory cannot recover this message and duplicate it
- This works if Mallory cannot mimic Alice’s and Bob’s messages

Delayed Key Exchange

- Alice and Bob need not exchange keys directly to communicate
  - Alice generates a random session key \( K \)
  - She obtains Bob’s public key from a database and encrypts \( K \) with that \( E_B(K) \)
  - She sends both the message encrypted with \( K \), \( E_K(M) \) and a key \( E_B(K) \) to Bob
- This is how most real-world protocols work

Authentication

- How does Alice prove her identity?
  - When she logs on
  - When she sends messages to Bob

Authentication On Log-on

- Alice inputs her password, computer verifies this against list of passwords
- If computer is broken into, hackers can learn everybody’s passwords
  - Use one-way functions, store the result for every valid password
  - Perform one-way function on input, compare result against the list
Authentication On Log-on

- Hackers can compile a list of frequently used passwords, apply one-way function to each and store them in a table – dictionary attack
- Host adds random salt to password, applies one-way function to that and stores result and salt value

Authentication On Log-on

- Someone sniffing on the network can learn the password
- Host keeps a file of every user’s public key
- Users keep their private keys
- When Alice attempts to log on, host sends her a random number \( R \)
- Alice encrypts \( R \) with her private key and sends to host
- Host can now verify her identity by decrypting the message and retrieving \( R \)

Authentication On Log-on

- Lamport hash – Alice will have different password each time she logs on
- To set-up the system, Alice enters random number \( R \)
- Host calculates
  \[ x_0 = h(R), x_1 = h(h(R)), x_2 = h(h(h(R))), \ldots, x_{100} \]
- Alice keeps this list, host sets her password to \( x_{101} \)
- Alice logs on with \( x_{100} \), host verifies \( h(x_{100}) = x_{101} \), resets password to \( x_{100} \)
- Next time Alice logs on with \( x_{99} \)

Authentication And Key Exchange

- Alice wants to exchange keys with Bob
  - How can she be sure that she is really talking to Bob?
  - How is this solved in the real world?
    - Bob gets his ID from a trusted authority – government, DMV
    - Bob shows his ID to Alice

Arbitrated Key Exchange

- Trent will play the role of trusted authority
- He will arbitrate key exchange and guarantee for Alice’s and Bob’s identity

Needham-Schroeder Key Exchange

- Alice sends message to Trent with her name, Bob’s name and a random number \( A, B, R_A \)
- Trent generates session key \( K \), encrypts \( K, A \) with key he shares with Bob \( E_{TA}(K, A) \), he then encrypts this message, \( K, B \) and \( R_A \) with key he shares with Alice \( E_{TB}(K, B, R_A) \)
- Alice decrypts the message, verifies \( R_A \) and sends \( E_{TB}(K, A) \) to Bob
- Bob decrypts the message, generates a random number \( R_B \) and sends to Alice \( E_{TA}(R_B) \)
- Alice decrypts the message, sends to Bob \( E_{TA}(R_B - 1) \)
Key Exchange With Digital Signatures

- Everyone has Trent’s public key
- Trent signs both Alice’s and Bob’s public keys – he generates **public-key certificate**
- When they receive keys they verify the signature
- Mallory cannot impersonate Alice or Bob because her key is signed as Mallory’s
- Certificate usually contains more than the public key
  - Name, network address, organization
  - Trent is known as **Certificate Authority (CA)**

Kerberos Authentication Service

- Kerberos is trusted authority with whom everyone shares keys
- When a client on a network wants to talk to a server, he issues a request for a **ticket** to Kerberos’ Ticket Granting Server (TGS)
- Client uses this ticket always when he talks to the server, sometimes he also sends **authenticators**
- Clients and servers do not trust each other

Getting Initial Ticket

- Alice sends a message with her name and a name of a Ticket Granting Server (TGS) to Kerberos
- Kerberos generates a session key \( K_{A,TGS} \) to be used between her and TGS and also generates Ticket Granting Ticket (TGT)
- Kerberos encrypts \( K_{A,TGS} \) with Alice’s secret key and sends that and TGT to Alice
- Alice retrieves \( K_{A,TGS} \) and saves it and TGT
Getting Initial Ticket

Alice

Kerberos

TGS

E_{A,TGS}(K_{A-TGS}), TGT

Save K_{A-TGS}, TGT

Getting Ticket for Server S

Alice sends a request with her name and server’s name to TGS, encrypted with K_{A-TGS}, accompanied with TGT and authenticator

TGS decrypts TGT with his secret key, retrieves K_{A-TGS}

TGS uses K_{A-TGS} to decrypt authenticator and compare Alice’s information in authenticator with information in TGT, and compare timestamps

If everything matches he generates a session key K_{A-S} and a valid ticket T_{A-S}

TGS encrypts K_{A-S} with K_{A-TGS} and sends this and T_{A-S} to Alice

Getting Ticket for Server S

Alice

Kerberos

TGS

Generate K_{A-S}

T_{A-S}=E_{A-S}(A, netaddr, timestamp, K_{A-S})

E_{A,TGS}(K_{A-S}), T_{A-S}, TGT, auth=E_{A,TGS}(A, timestamp)

Save K_{A-S}, T_{A-S}

Requesting Service

Alice sends a valid ticket T_{A-S} and authenticator

Server decrypts T_{A-S} with his secret key and retrieves K_{A-S}

Server uses K_{A-S} to decrypt authenticator and compare Alice’s information in authenticator with information in T_{A-S} and compare timestamps

If everything matches he grants the request

For applications that require mutual authentication server will send to Alice a timestamp encrypted with K_{A-S}
SSH
- Protocol for secure remote login
- Protocol architecture:
  - Transport layer protocol provides server authentication, confidentiality, integrity, compression (optional)
  - User authentication protocol authenticates the client to the server
  - Connection protocol
    - Multiplexes the encrypted tunnel into several logical channels

SSH Transport Protocol
1. Client contacts the server, performs TCP 3-way handshake
2. Both sides send the protocol and software version numbers to negotiate which protocols to use
3. They negotiate key exchange
4. They perform key exchange

3-Way Handshake
1. TCP SYN
2. TCP SYNACK
3. TCP ACK

Negotiate Protocol
- SSH protocol and sw version

Negotiate Key Exchange
- SSH_KEXINIT
- Random value R
- Key exchange algorithms
- Server host key algorithm
- Encryption algorithm client-to-server
- Encryption algorithm server-to-client
- MAC algorithm client-to-server
- MAC algorithm server-to-client
- Compression algorithm client-to-server
- Compression algorithm server-to-client
Negotiate Key Exchange

Client

SSH_KEXINIT

Random value R
Key exchange algorithms
Server host key algorithm
Encryption algorithm client-to-server
Encryption algorithm server-to-client
MAC algorithm client-to-server
MAC algorithm server-to-client
Compression algorithm client-to-server
Compression algorithm server-to-client

Server

SSH_KEXINIT

SSH_KEXINIT

Random value R
Key exchange algorithms
Server host key algorithm
Encryption algorithm client-to-server
Encryption algorithm server-to-client
MAC algorithm client-to-server
MAC algorithm server-to-client
Compression algorithm client-to-server
Compression algorithm server-to-client

Key Exchange (Diffie-Hellman)

Client

Compute

Server

\[ e^{x^g} \mod p \]

\[ H = \text{hash}(V_C || V_S || I_C || I_S || K_S || e || f || K) \]

\[ \text{Sig} = \text{signature on } H \text{ with private key} \]

\[ V_C, V_S \text{ – version string for client and server} \]

\[ K_1, K_2 \text{ – random numbers sent in KEXINIT message} \]

\[ K_S \text{ – server’s public (host) key} \]

Host Keys

- Each host has a host key – private/public key pair used for authentication
- Two different trust models exist:
  - First time client contacts a new host he stores the public key and associates it with the server name. Next time client checks if public key matches the stored value. If you have specified strict checking mismatches will be rejected, otherwise not.
  - Host’s name-to-key mapping is certified by a certification authority, client only knows CA’s public key.

Host Keys

- Each host has a host key – private/public key pair used for authentication
- Two different trust models exist:
  - First time client contacts a new host he stores the public key and associates it with the server name. Next time client checks if public key matches the stored value. If you have specified strict checking mismatches will be rejected, otherwise not.
  - Host’s name-to-key mapping is certified by a certification authority, client only knows CA’s public key.

- Each host has a host key – private/public key pair used for authentication
- Two different trust models exist:
  - First time client contacts a new host he stores the public key and associates it with the server name. Next time client checks if public key matches the stored value. If you have specified strict checking mismatches will be rejected, otherwise not.
  - Host’s name-to-key mapping is certified by a certification authority, client only knows CA’s public key.
Key Exchange (Diffie-Hellman)

Client

Server

Verify $K_a$ using certificates or local database

Compute

$K = g^x \mod p = g^y \mod p$

$H = \text{hash}(V_C || V_S || I_C || I_S || K_S || e || f || K)$

Verify signature $\text{Sig}$ on $H$

Example

```
ssh -v copland
```

```
debag1: Connecting to copland [128.175.13.92] port 22.
debag1: Connection established.
debag1: Identity file /usa/sunshine/.ssh/identity type -1
debag1: Remote protocol version 1.99, remote software version OpenSSH_3.7.1p2
debag1: match: OpenSSH_3.7.1p2 pat OpenSSH*
debag1: Enabling compatibility mode for protocol 2.0
debag1: Local version string SSH-2.0-OpenSSH_3.7.1p1
debag1: SSH2_MSG_KEXINIT sent
debag1: SSH2_MSG_KEXINIT received
debag1: kex: client->server blowfish-cbc hmac-md5 none
debag1: kex: client->server blowfish-cbc hmac-md5 none
debag1: kex: client->server blowfish-cbc hmac-md5 none
debag1: kex: client->server blowfish-cbc hmac-md5 none
debag1: kex: client->server blowfish-cbc hmac-md5 none
debag1: kex: client->server blowfish-cbc hmac-md5 none
debag1: kex: client->server blowfish-cbc hmac-md5 none
```

User Authentication Protocol

- Runs on top of transport layer
- Authenticates the client’s user to the server

User Authentication

Alice

Client

List of authentications that can continue

Server

Client chooses one of the offered methods

publickey, password, hostbased

Authentication request:
Username: alice
Service name
Authentication method name
Authentication data

Server verifies that the username exists authentication data is correct

Authentication success
or

Authentication failure
List of authentications that can continue
Public Key Authentication

Client

Alice

Authentication request:
Username: alice
Service name
Authentication method name: publickey
Public key algorithm name
Alice’s public key
Signature on session identifier and X with Alice’s private key

Server

Password Authentication

Client

Alice

Authentication request:
Username: alice
Service name
Authentication method name: password
Password in plaintext

Server

Host-Based Authentication

Client

Alice

Authentication request:
Username: alice
Service name
Authentication method name: hostbased
Public key algorithm for host key
Public host key and certificates
Client host name: strauss
User name on server: alicebrown
Signature on session identifier and X with strauss’ private key

Server

Server sometimes does DNS lookup to verify that source address in packets and client host name match

Connection Protocol

- A channel can be open for terminal sessions, forwarded connections, etc.
- All channels are multiplexed onto a single connection
- Channels can be open from either side
  - Send channel request describing the type of channel
  - Request is granted if such channel is available