One-Way Hash Functions

- Take a variable-length input *M* and produce fixed-length output (*hash value* or *message digest*)

$$h = H(M)$$

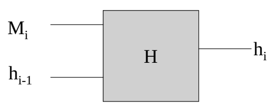
- The idea is to fingerprint *M*
  - Given *M* easy to compute *h*
  - Given *h* very hard to compute *M*
  - One-bit change in *M* changes many bits in *h*
  - Good one-way hash function is collision-free: given *M* it is very hard to find M′ such that H(M)=H(M′)
  - One-way hash function is public

Birthday Attacks

- Sometimes what we also need is **collision resistance**: it is hard to find two random messages *M* and M_1 such that H(*M*)=H(M_1)
- This is called *birthday attack* and is based on a *birthday paradox*
  - How many people must be in a room until the probability is greater than 0.5 that two of them have the same birthday? 23
  - For a hash function that produces *m* bit hash, it takes $2^{m/2}$ trials to find two messages that hash to the same value
  - We need large *m*, currently 128-160

One-Way Hash Functions

- Divide M into blocks, generate hash value iteratively



- Hash value of the whole message is obtained in the last step

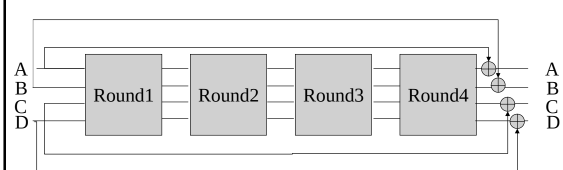
MD5

- Divide M into 512-bit blocks
  - Pad M with string of 1 and many zeros so that it is 64 bit short of multiple of 512
  - Concatenate original length as 64-bit number
- Blocks are processed sequentially
- Last result is hash value for the whole message and is 128-bit long

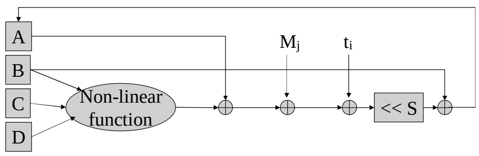
MD5

- For each message block:
  - Variables A, B, C, D are initialized (for first block they are constant values)
  - Go through 4 rounds; each round repeats the following operation 16 times:
    - Performs non-linear function on three variables
    - Sums the result, the fourth variable, message subblock (32-bits) and a constant
    - Rotates the result to the left, adds it to one of the variables and replaces this variable

MD5 One Block Processing



MD5 One Round



SHA

- Similar to MD5 but:
  - Has 20 operations per round
  - Operations are different than in MD5 but along similar lines
  - Each message block (16*32-bits) is expanded (80*32 bits)
  - Produces 160-bit hash value

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 (noone 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 $\rightarrow$ add timestamps
  - Signing the whole document with public key is slow $\rightarrow$ sign a hash of the document produced by one-way hash function

Digital Signature Algorithm (DSA)

- US standard by NIST
- Choose public values p, q, and g
  - p is a prime number, L bits long ($64 \le L \le 128$, $L=k*64$)
  - q is a 16-bit prime factor of $p-1$
  - $G=h^{(p-1)/q}$ where h is any number such that $h^{(p-1)/q} \bmod p > 1$
- Choose private key x and public key y so that
  - x is a number less than q
  - $y = g^x \bmod p$

13

CIS 659 – Introduction to Network Security – Fall 2003 – Class 5 – 9/23/03

Digital Signature Algorithm (DSA)

- To sign a message M, Alice:
  - Generates a random number k, $k<q$
  - Generates signatures r and s

$$r = (g^k \bmod p) \bmod q$$
$$s = (k^{-1}(H(M) + x * r)) \bmod q$$

- To verify signatures r and s, Bob computes

$$w = s^{-1} \bmod q$$
$$u_1 = (H(M) * w) \bmod q$$
$$u_2 = (r * w) \bmod q$$
$$v = ((g^{u_1} * y^{u_2}) \bmod p) \bmod q$$

- If v = r then signature is verified

14

CIS 659 – Introduction to Network Security – Fall 2003 – Class 5 – 9/23/03

Digital Signature Algorithm (DSA)

- To sign a message M, Alice:
  - Generates a random number k, $k<q$
  - Generates signatures r and s

$$r = (g^k \bmod p) \bmod q$$
$$s = (k^{-1}(H(M) + x * r)) \bmod q$$

- To verify signatures r and s, Bob computes

$$w = s^{-1} \bmod q$$
$$u_1 = (H(M) * w) \bmod q$$
$$u_2 = (r * w) \bmod q$$
$$v = ((g^{u_1} * y^{u_2}) \bmod p) \bmod q$$

- If v = r then signature is verified

15

CIS 659 – Introduction to Network Security – Fall 2003 – Class 5 – 9/23/03

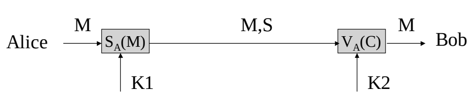
DSA vs. RSA

- Values r and k^{-1} can be precomputed so DSA signatures can be made very fast
- However verification is slower than RSA
- RSA can be done through DSA
  - With modulus n, message M and public key e for RSA, just do $p=q=n$, $g=M$, $k=e$, $x=h=0$
  - Returned value r will be ciphertext
- DSA is public, RSA used to be patented

16

CIS 659 – Introduction to Network Security – Fall 2003 – Class 5 – 9/23/03

Terminology



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

17

CIS 659 – Introduction to Network Security – Fall 2003 – Class 5 – 9/23/03

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 $V_A(S_A(M)) = M$

18

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

Revisiting Cryptography Goals

- Protect private communication in the public world (Symmetric and public key cryptography)
  - Alice and Bob are shouting messages in a crowded room
  - No guest can understand what they are saying
- Authentication (Digital signatures)
  - Bob can verify that Alice has created the message
- Integrity (Message digests)
  - Bob can verify that message has not been modified
- Non-repudiation (Digital signatures + timestamps)
  - Alice cannot deny that she indeed sent the message

Revisiting Cryptography Threats

- Ciphertext-only attack
- Known plaintext attack
- Chosen plaintext attack
- Adaptive chosen plaintext attack
- Man-in-the-middle attack
  - Substitute, modify, drop, replay messages

Revisiting Common Practices

- Alice and Bob exchange symmetric key using:
  - Public-key encryption
    - If they first send each other public keys, Mallory can do man-in-the-middle attack
    - If they obtain public keys from a database, Mallory can poison public-key database with bad keys
  - Diffie-Hellman key exchange
    - Mallory can do man-in-the-middle attack

Man-in-the-Middle Attack on Key Exchange

- 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)
- 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

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

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 and Key Exchange

- Alice wants to exchange keys with Bob
  - How can she be sure that she is 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

Key Exchange with Digital Signatures

- Trent signs both Alice's and Bob's public keys – he generates *public-key certificate*
- When they receive keys they verify the signature
  - Everyone has Trent's public key
- Mallory cannot impersonate Alice or Bob because his 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)*

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 $E_{TB}(K, A)$, he then encrypts this message, K, B and R_A with key he shares with Alice $E_{TA}(K, B, R_A, E_{TB}(K, A))$
- 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 $E_K(R_B)$
- Alice decrypts the message, sends to Bob $E_K(R_B\text{-}1)$

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

Kerberos Authentication Service

- A ticket is used to pass securely to the server the identity of the client
  - It is good for a single client and single server for some period of time
  - It contains client's name and network address, server's name, timestamp and a session key, all encrypted with a key server shares with Kerberos
- An authenticator is generated whenever a client requests some service from the server
  - It is good only for one request
  - It contains client's name, a timestamp and an optional additional key, all encrypted with session key

Kerberos Authentication Service

- To get 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_{ATGS} to be used between her and TGS and also generates Ticket Granting Ticket (TGT)
  - Kerberos encrypts K_{ATGS} with Alice's secret key and encrypts TGT with TGS's secret key, sends both to Alice
  - Alice retrieves K_{ATGS} and saves it and TGT

Kerberos Authentication Service

- To get ticket for specific server
  - Alice sends a request with her name and server's name to TGS, encrypted with K_{ATGS}, accompanied with TGT and authenticator
  - TGS decrypts TGT with his secret key and retrieves K_{ATGS}
  - TGS uses K_{ATGS} 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_{AS} to be used between her and server and a valid ticket T_{AS}
  - TGS encrypts K_{AS} with K_{ATGS} and encrypts T_{AS} with server's secret key, sends both to Alice

Kerberos Authentication Service

- To request service
  - Alice sends a valid ticket T_{AS} and authenticator
  - Server decrypts T_{AS} with his secret key and retrieves K_{AS}
  - Server uses K_{AS} to decrypt authenticator and compare Alice's information in authenticator with information in T_{AS}, 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_{AS}