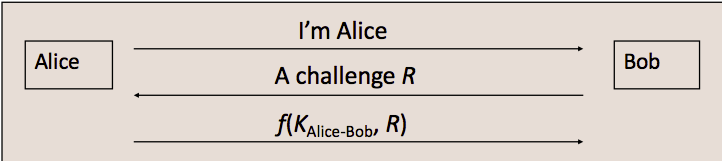
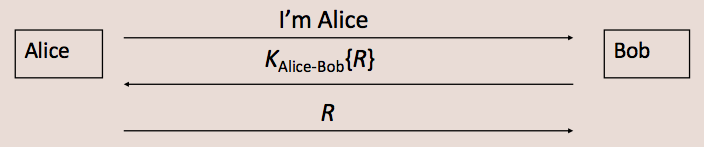
Authentication Protocol

**Weaknesses of Auth w/ Shared Secret:** auth is not mutual; Trudy can convince Alice that she is Bob; Trudy can hijack conversation after initial exchange; if the shared key is derived from a password, Trudy can mount an offline password guessing attack; Trudy may compromise Bob’s database and later impersonate Alice **|| Variation of Auth w/ Shared Secret:** requires reversible crypto; all previous weaknesses remain; Trudy doesn’t have to see R to mount offline password guessing if R has certain patterns

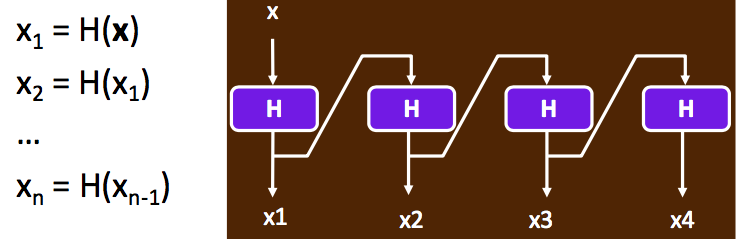
**Auth w/ Shared Secret: Variation of Auth w/ Shared Secret:**

** **

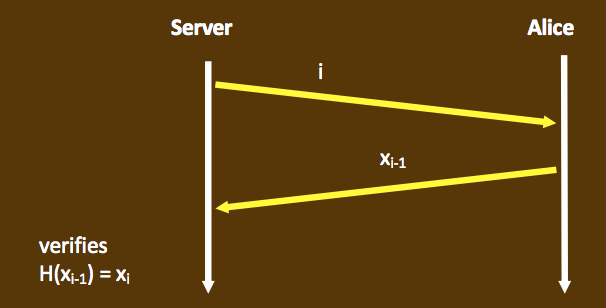
User Authentication

**Authentication:** the process of reliably verifying certain information; user and message authentication **|| Identification:** process through which one ascertains the identity of another person or entity **|| Auth vs Identification:** identification requires that verifier check the info presented against all the entities it knows about; Auth requires that info be checked for a single entity; Identification must identify a unique entity; *Auth does not need uniqueness* **|| Auth is Based on:** what the user *knows*; *Where* the user is or can be reached; *Physical Characteristics* of a user; *What* the user has in their possession **|| Types of Auth:** Password, Address, Crypto **|| Address-based User Auth:** associates identity with network or email address; Threats: spoofing of addresses **|| Crypto Auth:** user performs a requested crypt operation on a value (a challenge); usually reduced to the knowledge of a key; RSA **|| Password Storage:** best way is to use password to make a hash and store the hash; don’t store in unencrypted file **|| Common Passwords:** pet names, common names, common words, dates, variations of these **||Dictionary Attack v1 (online):** create a dictionary of common words and names and their simple transformations; use these to guess password **|| Dictionary Attack v2 (offline):** usually *F* is public and so is password file; compute *F(word)* for each word in the dictionary; a match gives the password **|| Dictionary Attack v2 (offline):** to speed up search, pre-compute *F*(dictionary); a simple lookup gives the password **|| Password Salt:** make dictionary attack more difficult; Salt is n-bit # between 0 & 2n; derived from system clock or PID; *Storing password*: Username, Salt, *F*(*Password* + *Salt*); *Verifying Password*: reverse process **|| Password Guidelines:** initial passwords are system generated & must be changed by user at first login; user must change password periodically; passwords vulnerable to dictionary attack are rejected; user should not use same password on multiple sites **|| Password Vulnerabilities:** eavesdropping, keystroke logging, Trojan horse password entry program, MITM attack, phishing **|| Defeat Eavesdropping:** generate list of passwords and use each *only once*; S/Key protocol which is based on hash function **|| S/Key Password Gen:** Alice selects password x, Alice specifies *n*, the number of passwords to generate; Alice’s computer then generates a sequence of passwords; Alice communicates (securely) to a server the last value in the sequence, x*n*; no one knowing x*i* can easily find an x*i*-1 such that H(x*i*-1) = x*i*; server needs possession of x*i* **|| Limitations of S/Key:** value of *n* limits # of passwords; need to regenerate new chain of passwords; does not authenticate server; real server sends *i* to fake server, which is masquerading as Alice, fake server sends *I* to Alice, who responds with x*i* – 1, fake server then presents x*i* – 1 to real server **|| Auth Hardware (Token):** a physical device that can be interfaced to the computer and carries identifying info; *Passive* tokens store info; *Active* tokens have processors and can perform cryptographic operations **|| Time Synchronized Tokens:** contains internal clock, display, secret key; computes one-way function of current time+key & shows that; user types it into server; clocks must be synchronized

**S/Key Process:**

****

**Server S/Key Auth:**

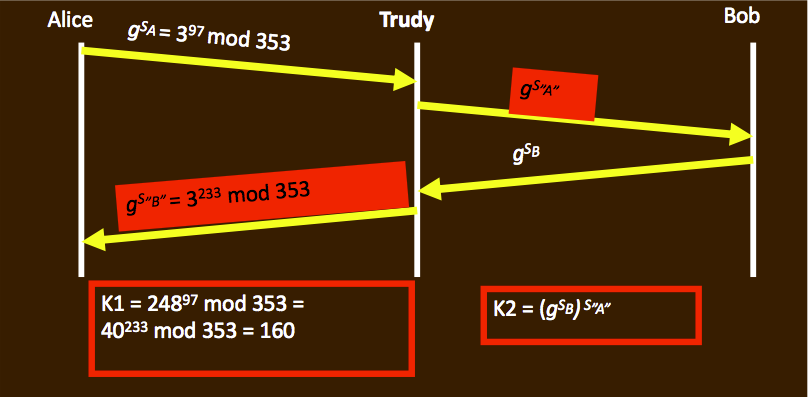
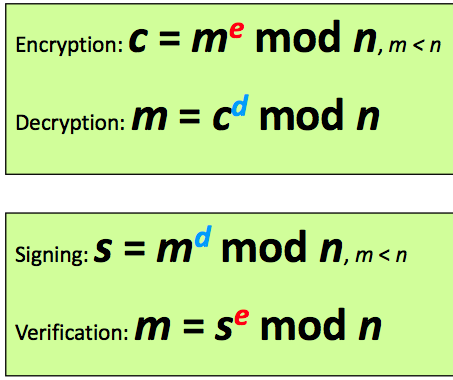
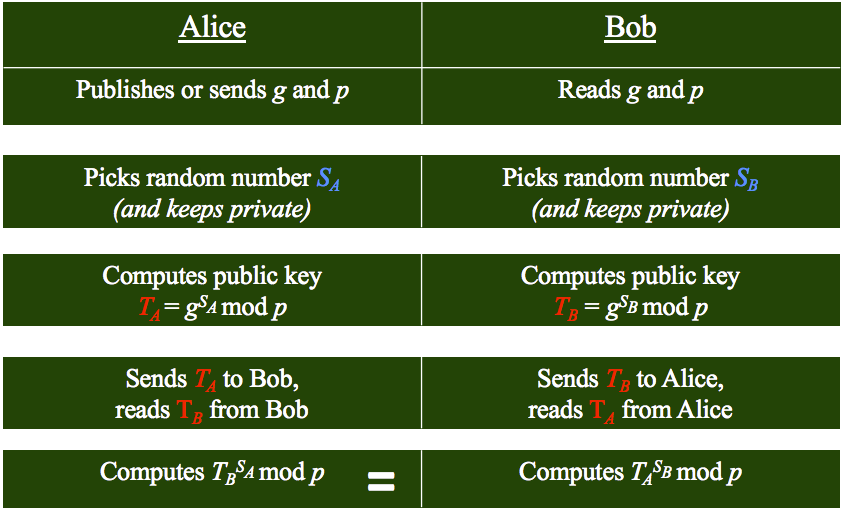
****

Public Key Cryptography

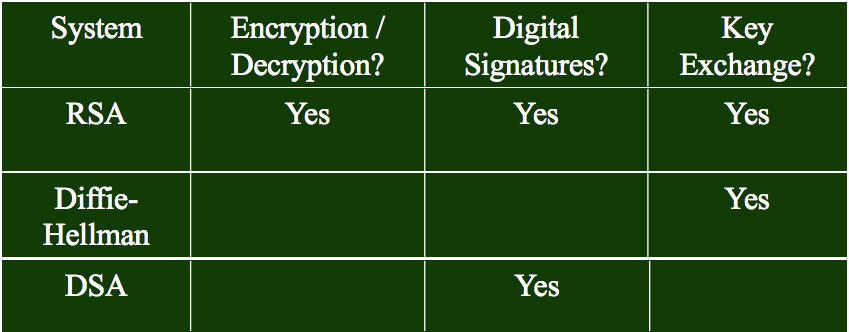
**Public Key Crypto:** public/private key pair used; asymmetric crypto; much slower than secret key crypto **||Applications of Public Key Crypto:** Message integrity with *digital signatures* (only one person can sign message) (sign with private key, verify with pub key); Communicating securely over insecure channel (encrypt with other persons public key & they decrypt with private key); Secure storage on insecure medium (encrypt with pub key, decrypt with private key); User Auth (do op with private key and others verify with pub key); Key exchange for secret key **|| Public Key Requirements:**  must be computationally easy to generate a key pair but hard to determine the private key, given the public key; must be computationally easy to encrypt using pub key, easy to decrypt using private key and hard to recover plaintext from just the cipher text and pub key **||RSA Algorithm:** most popular pub key method; based on large numbers being hard to factor; variable key length (1024 bits or greater); plaintext block size must be smaller than key size; ciphertext block size is same as key size **||RSA Operations:** Encryption, decryption, signing & verification **||RSA Key Negotiation Procedure:** A sends random number R1 to B, encrypted with B’s public key; B sends random number R2 to A, encrypted with A’s public key; A & B both decrypt received messages using their respective private keys; A & B both compute K = H(R1 XOR R2), and use that as the shared key **|| Make Breaking RSA Harder:** key sizes are 1024 bits but 2048 bits is better; Make n difficult to factor by: p &q lengths should be similar, both (*p*-1) & (*q*-1) should contain a “large prime factor”, gcd(*p*-1, *q*-1) should be “small”, and *d* should be larger than *n*1/4 **|| Probable-Message RSA Attacks:** encrypt all possible plaintext messages; try to find a match between the ciphertext and one of the encrypted messages; only works for small plaintext message sizes; can be defended against by padding plaintext with random text before encryption **|| Timing RSA Attack:** recovers the private key from running time of the decryption algorithm; computing m=c*d*mod *n*; defense is blinding **|| RSA Blinding Algorithm:** confounds timing attacks; generate a random number *r* between 0 & *n*-1 such that gcd(*r, n*) = 1; compute c’ = c \* *re* mod *n*; compute *m’* = (c’)*d* mod *n*; compute *m* = *m’* \* *r*-1 mod *n*; attacker does not know what the bits of c’ are **|| Diffie-Hellman Key Exchange:** For negotiating a shared secret key using only public communication; does not provide authentication of communicating parties; Involves *p* a large prime number (512 bits), *g* is a primitive root of *p* and *g* < *p*; *p* & *g* are publicly known; the shared key is the computed *gsASB* mod *p* **|| Diffie-Hellman Security:** secure because it is hard to compute *TA* = *gSA* mod *p*), *g*, and *p* **|| Limitations of Diffie-Hellman:** expensive exponential ops used; *only* used for key negotiation not pub key encryption; NOT for user auth; can negotiate key with a complete stranger **||Prevent DH MITM Attacks:** requires communicating parties already share a secret; then use encryption or MAC of DH messages **|| D-H in “Phone Book” mode:** Alice and Bob each choose a semi-permanent secret number & generate TA & TB; Alice and Bob publish these numbers where either person can get their respective number at any time; They both generate a semi-permanent shared key without communicating **||Make *g* & *p* Secure:** change *g* & *p* periodically; don’t use the same *g* & *p* for everyone; *g*(*p*-1)/2 should be -1 mod *p*;

(*p*-1)/2 should be prime **|| Certification Authorities (CA):** a trusted node that maintains the public keys for all nodes (each node maintains private key) **|| Certificates:** signed message vouching that a particular name goes with its pub key; involved in authenticating user’s public keys; knowing the CA’s pub key, users can verify the cert and authenticate Alice’s pub key; can hold expiration date & time; Alice keeps the cert as long as she has the same pub key and the cert does not expire; Alice can append the cert to her messages so others know her pub key for sure

**Diffie-Hellman Man-IT-Middle Attack: RSA Operation Equations: Diffie-Hellman Key Exchange Protocol:**

**  **

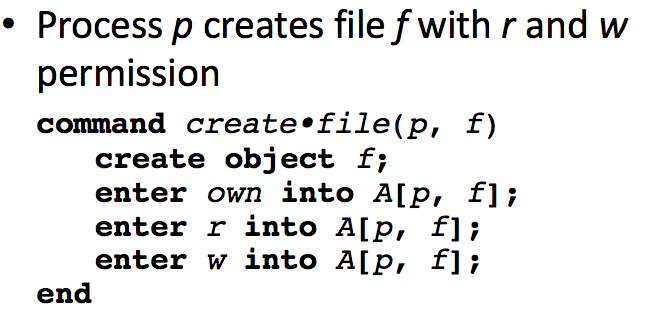
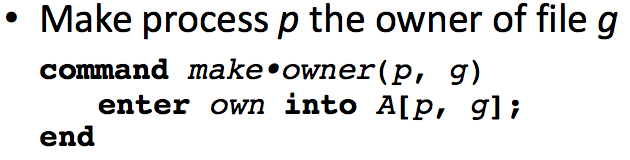
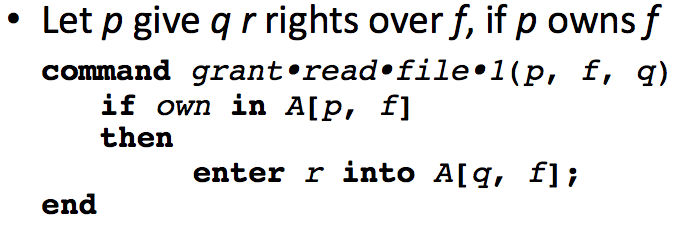
**Pub Key Algorithm Comparisons:**

****

Access Control Matrix

**Protection state of system:** current settings, values of system relevant to protection **|| State of a system:** collections of current values of all memory locations, storages, registers **|| Protection state:** subset of state of system; deals with protection **|| Access control matrix:** describes protection state precisely; matrix describe rights of subjects; state transitions change elements of matrix**|| State Transitions:** change the protections state of system; state transitions due to commands; “|-“ represents transition; command moves system to next state; a sequence of commands moves system to next state **|| Transformation procedures:** another name for commands; the result of transforming an authorized state with an operation allowed in that state is an authorized state**|| Protection State Transitions:** sequences of state transitions are represented by commands that update the access control matrix **|| Mono-operational commands:**  single primitive operation in this command **||Mono-conditional command:** single condition in this command **|| Special Rights:** the copy right (or grant right) allows the possessor to grant rights to another; the own right enables the possessor to add or delete privileges for themselves and others **|| Principle of attenuation of privilege:**  a subject may not give rights it does not possess to another **|| Key points:** access control matrix simplest abstraction mechanism for representing protection state: Transitions alter protection state; 6 primitive operations alter matrix; Transitions can be expressed as commands composed of these operations and conditions**|| Secure:**  use access control matrix to express the policy**|| Leaked:**  when a generic right is added to an element of the access control matrix not already containing **|| Safe:**  when a system can’t leak**|| Safety:** refers to the abstract model**|| Security:** refers to actual implementation; A secure system corresponds to a model safe with respect to all rights; BUT a model safe with respect to all rights does not ensure a secure system

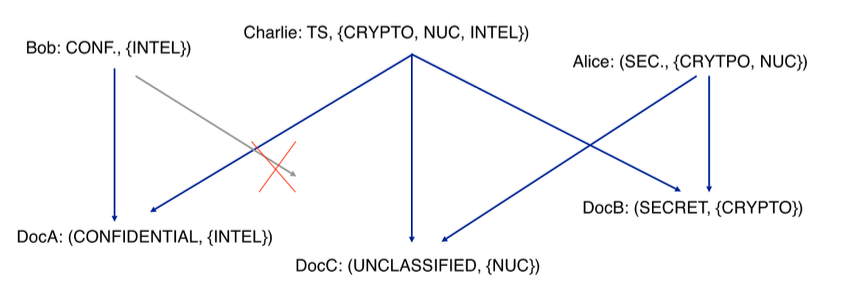
**Create File: Mono-Operational Command: Conditional Command:**

Security Policy

**Policy Partitions:** Authorized (secure) these are the states the system can enter; Unauthorized (nonsecure) if the system enters any of these states, its’s a security violation **|| Secure System:** starts in authorized state; never enters unauthorized state**|| Breach of Security:** occurs when a system enters an unauthorized state **|| Confidentiality:** information is confidential if target group cannot access it**|| Trust Integrity:**  trust information, its conveyance and protection (data integrity) **|| Origin Integrity:** information about origin of something or identity **|| Resource Integrity:**  means resource functions as it should (assurance) **|| Availability:** If the specified people can access the info **|| Traditional availability:** Person X gets access or not **|| Quality of Service Availability:** promised a level of access (for example, a specific level of bandwidth) and not meet it, even though some access is achieved **|| Policy:** the person who breaks policy is the one in the wrong no matter what **|| Mechanisms:** entity or procedure that enforces some part of the security policy; has access controls (like bits to prevent someone from reading a homework file) **||Access control:** determines what right a particular entity has for a set of objects; have read/write access?; can be considered as a function: P(S,O,R) -> {accept, deny} where S=subjects, O=objects, R=rights; policy is a lot these tuples; common way to represent policy**|| 2 Ways to specify a policy:** Discretionary access control (DAC), and Mandatory Access Control (MAC) **|| Discretionary Access Control:** Object “owners” define policy; individual user sets access control mechanism to allow or deny access to an object; ex-UNIX file system read/write access assigned by file owners **|| Mandatory Access Control:** Environment enforces static policy; environment (system mechanism controls access to object, and individual cannot alter that access **|| Process Labeling:** system assigns labels for processes, objects, and a dominance calculus to evaluate rights**|| Access Control Model:**  used to express policy; security model **|| Security Model:**  model that represents a particular policy or set of policies; abstracts details relevant for analysis; ***Policies***can focus on secrecy, integrity, conflict of interests and jobs **|| Types of Security Polices:** Military (confidentiality), commercial (integrity), Confidentiality, Integrity**|| Goals of Confidentiality policy:** prevent unauthorized disclosure of info; deals with info flow; Multi-level security (MLS) models are best **|| Bell-LaPadula Model:** Security levels arranged in linear ordering; subject have *security clearance*; objects have *security classification* **|| Lattice Model:** MLS; expand security to include categories **||Lattice Model Categories:** NUC(lear), INTEL(igence), CRYPTO(graphy) **|| Lattice Model Format:**  Security level is (*clearance, category set*) or (*L, C*); Example: Charlie: (TOP SECRET, {CRYPTO, NUC, INTEL}) **||Basic Security Theorem:**  if a system is initially secure and every transition satisfies the simple security condition and the \*- property, then every state of the system is secure **|| Confidentiality:** who can READ a document **|| Integrity:** who can WRITE to a document **|| Biba Model:** users can only view content at or above their own integrity level; a monk may read a book written by high priest, but may not read a pamphlet written by a lowly commoner; opposite of lattice model **|| Low-Water Mark Integrity:** Change integrity level based on actual dependencies; subject is initially at the highest integrity; subject has integrity of lowest object read **|| Chinese Wall model (Hybrid Policy Model):** model of security policy that refers equally to confidentiality and integrity; deals with conflict of interest situations **|| Role Based Access Control (RBAC):**  access depends on function, not identity **|| Role:** a collection of privileges/permissions associated with some function or affiliation; permissions are static; not direct MAC and DAC but can be either one

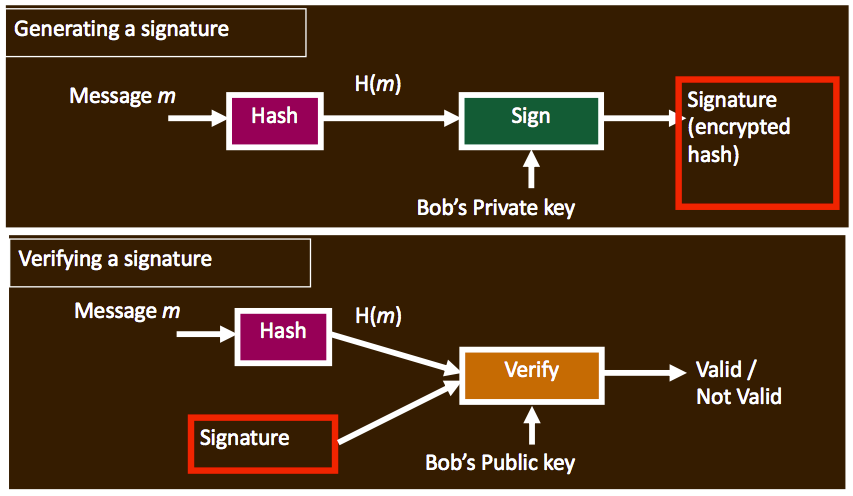
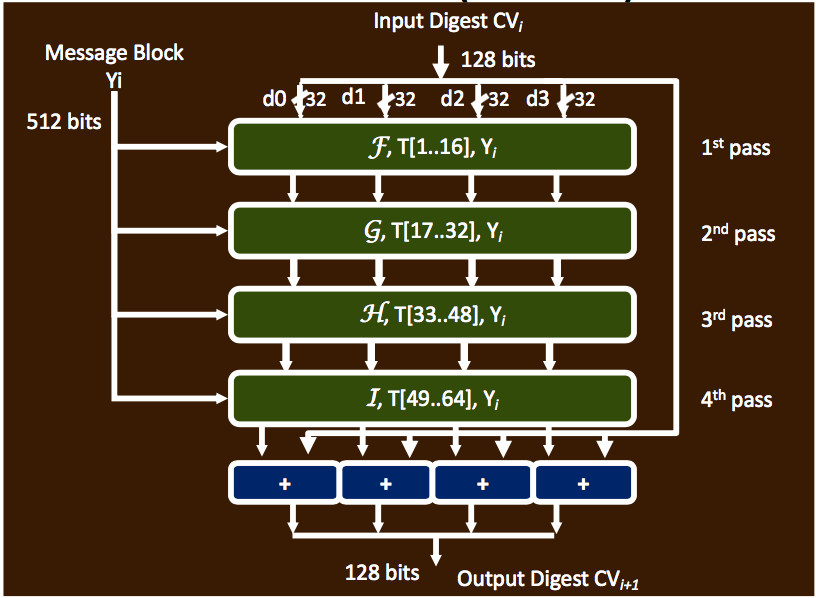
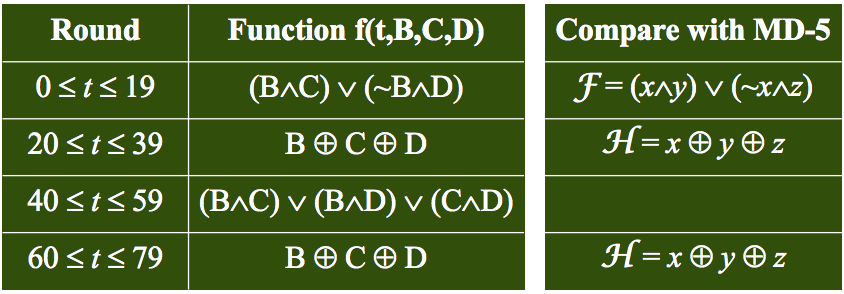
**Lattice Model Example:**

****

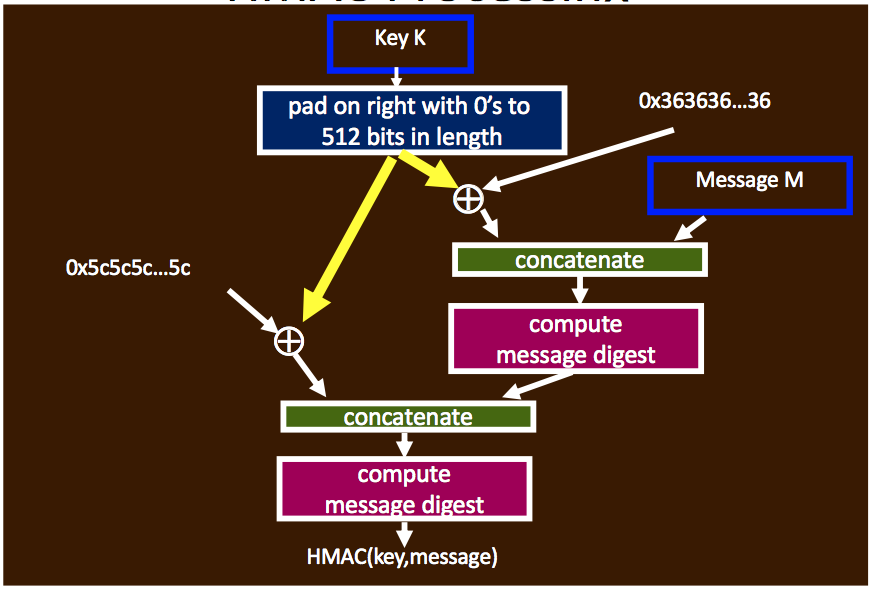
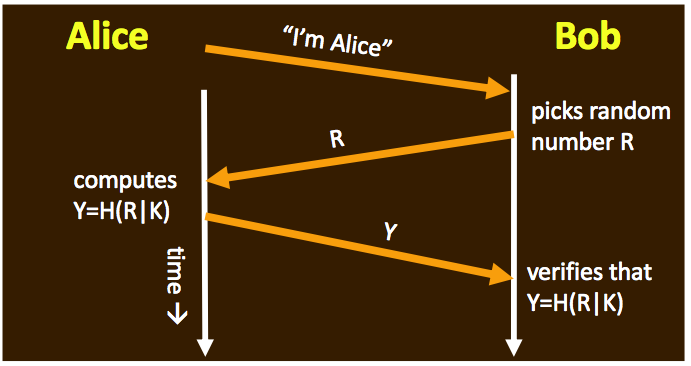
Hash Functions

**Hash Function:** aka message digest, one-way transfer function, one-way function, hash; length of *H(m)* much shorter than length of *m*; usually fixed lengths 128/160 bits **|| Properties of Hash Functions:** performance – easy to compute; One-way property – given *H(m)* but not *m*, it’s hard to find *m*; Collision resistance – given *H(m),* it’s computationally infeasible to find *m*’ such that *H(m’)=H(m)*; hard to find m1 & m2 such that *H(m1)=H(m2)* **|| Birthday Paradox:** What is the smallest group size *k* such that the probability that at least 2 people in the group have the same birthday is greater then0.5?; P(*k* people having *k* different birthdays): Q(365, *k*) = 365!/(365-*k*)!365k ; P(at least 2 people have the same birthday: P(365, *k*) = 1-Q(365, *k*) >= 0.5; *k* is about 23; basically there will always be 2 people that have the same birthday given a large enough sample **|| Hash Birthday Paradox:** with *H(m)*, probability of at least 0.5 with 2m/2 random inputs, 2 messages will have the same hash via the birthday attack; CHOOSE m >= 128 **|| File Auth:**  computer hash of file, store hash separate from file, check for tampering by computing hash again and checking against original hash **|| Modern Hash Functions:** Message Digest 5 (MD5), Secure Hash Algorithm (SHA), SHA1, SHA256, SHA384 **|| Notations:**  ~x-NOT; x^y-AND; xVy-OR; x<<y-left shift of x by y bits **|| MD5 Passes:** F(x,y,y)=(x^y)V(~x^z); G(x,y,z)=(x^z)V(y^~z); H(x,y,z)=(x XOR y XOR z); I(x,y,z)= y XOR (x V ~z); every pass has 16 processing steps (each step involves the previous functions and circular shift); NO LONGER SECURE **|| SHA1:** processed in 512bit blocks; message must be < 264 bits; message digest output is 160 bits long; block processing is 80 steps **|| SHA1 vs MD5:** SHA1 is stronger and twice as expensive to compute; both are faster than DES **|| Extension Attack:** given M1 and secret key K, can easily concatenate and compute the hash H(K |M1|padding); given M1, M2 and H(K|M1|padding) easy to compute H(K|M1|padding|M2|newpadding) for some new message M2; simply use H(K|M1|padding) as the IV for computing the hash of M2|newpadding which doesn’t require know the value of secret key K **|| HMAC:** protects against extension attack; digest-inside-a-digest with secret used at both levels; the hash function used determines the length of the message digest = length of HMAC output **||Hashing Summary:** Hashing is fast to compute; hash images must be at least 128 bits long

**Digital Signatures: MD5 Block Processing: SHA1 vs MD5:**

**  **

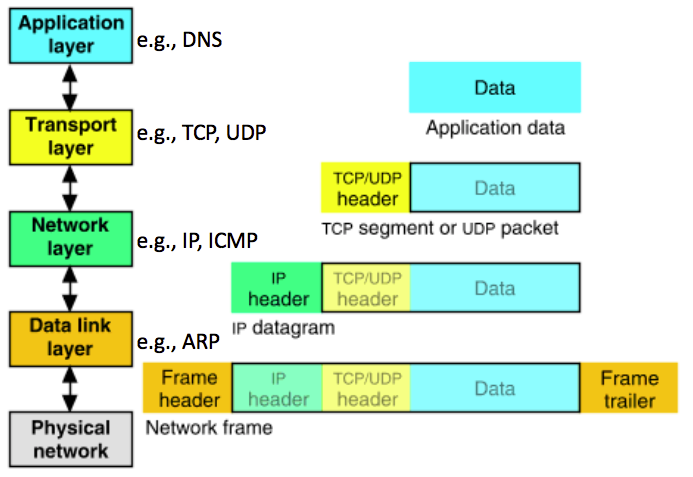
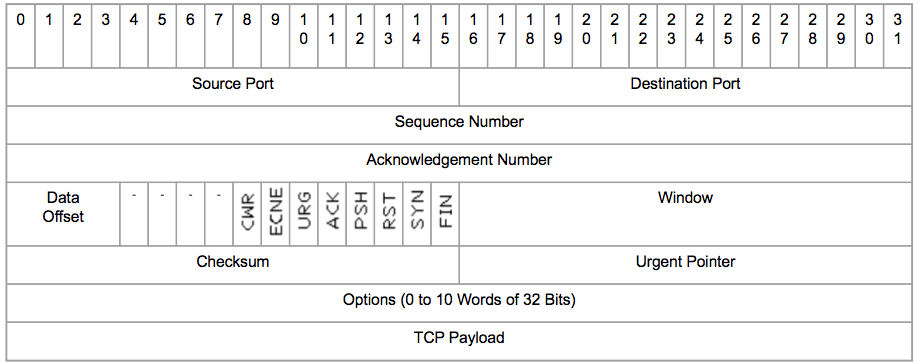
**HMAC Processing: User Authentication:**

** **

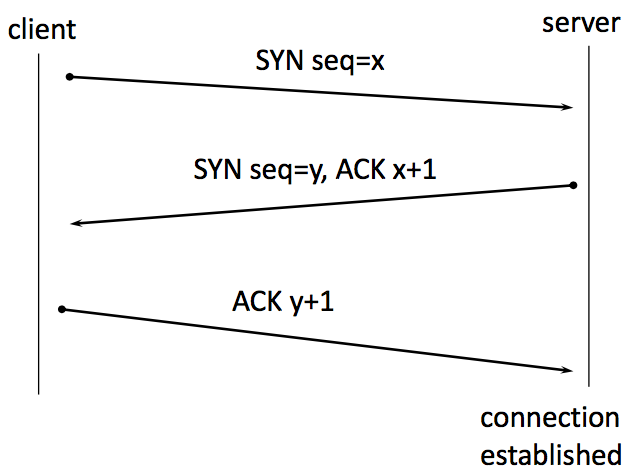
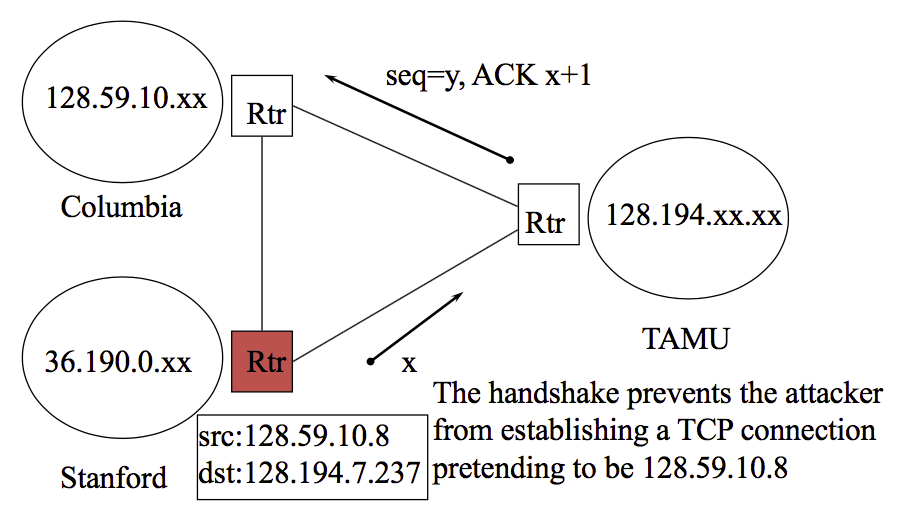
Vulnerability Analysis

**Threat:** potential violation of security **||Attack:** any action that violates security **||Vulnerability:** failure of security policies, procedures, and controls that allow a subject to commit an action that violates the security policy **|| Vulnerability analysis:** a process that defines, identifies, and classifies the security holes in a computer, network, or communications infrastructure; can forecast the effectiveness of proposed countermeasures and evaluate their actual effectiveness **||Typical Steps:** Defining & classifying network/system resources; assigning relative levels of importance to the resources; identifying potential threats to each resource; developing a strategy to deal with the worst problems first; defining and implementing ways to minimize consequence **|| Port Info:** ports dynamically “bind” IP packets/port to a process; ports range from 0-65535; port 0-1023 need root access **||NMAP:** does SYN scan; does TCP connect, FIN, Ping, UDP; FIN probe, IPID sampling, TCP Initial Window, TCP options**|| OS Fingerprinting:** method of detecting the remote hosts operating system using information leaked by that host’s TCP stack; figures it out by looked at OS responses to weird packets (active mode) or observing captured network traffic (passive mode) **|| FIN probe:**  send any packet without an ACK/SYN flag (FIN packet) to an open port and wait for response; correct RFC 793 behavior is to NOT respond but Windows, BSDI, CISCO & IRIX send RESET packet back **||IP Spoofing:** send harmful packet from fake trusted source so packet is received; attack packets hide the attacking source; IPv4 has no authentication for source **||Spoofing fixes:** router/firewall filtering; TCP handshake **|| Router Filtering:** decides whether packet should come from this side of network; very effective; hard to handle volatile or mobile networks; need to trust other routers **|| TCP “Window”:**  with every ACK, the receive indicates how many more bytes it is prepared to receive **|| TCP Sequence Numbers:** Initial Sequence Number (ISN) for both sides of connection; ISN’s are exchanged in SYN handshake; TCP packets with sequence numbers outside window are ignored **||TCP Handshake:**  effective for stopping TCP attacks **|| SYN Flooding attack:**  exploits the 3-way handshake of TCP; SYN packets are sent to the target node with incomplete source IP addresses; node under attack sends ACK packet and waits for response; request not processed therefore takes up memory; many SYN packets clog the system and take memory; eventually the attacked note runs out of memory and can’t answer requests; streaming spoofed TCP SYN’s **||SYN Flooding Synopsis:** server starts “half open” connections, they build up, queue gets full and all additional requests are blocked; DOS attacks use SYN floods **|| Prevent SYN Flood:** SYN cookies, server responds with SYN-ACK cookie; server does not save state; honest client responds with ACK(sqn+1); server checks response and validates connection**|| TCP RST Attack:**  send a reset packet with spoofed IP to either side of valid connection; need to guess a sequence number inside the window; sequence numbers-32 bits; window size-16 bits; # of guesses-32-16=16 bit addr space 65535 RST attempts **||Address Resolution Protocol (ARP):** used by routers to find the destination node; router broadcasts IP address of destination and gets the 48bit MAC address **||ARP Poisoning:** ARP communication is intercepted by redirection from router; hosts store the IP-to-MAC address mapping in the ARP table; after spoof, all packets meant for one computer go to another because router has incorrect MAC addr **|| Defenses Against ARP spoofing:** no big defense; static ARP entries, port Binding **|| Internet Control Message Board (ICMP):** error handling and debugging protocol; inside IP header; no authentication **|| Smurf Attack:** generate ping stream (ICMP echo request) to a network broadcast address with spoofed IP set to victim host; every host on network will generate ping stream toward victim; amplified ping reply stream will overwhelm victim’s network connection **|| ICMP Redirect Attack:** tell computer to change destination to attacker

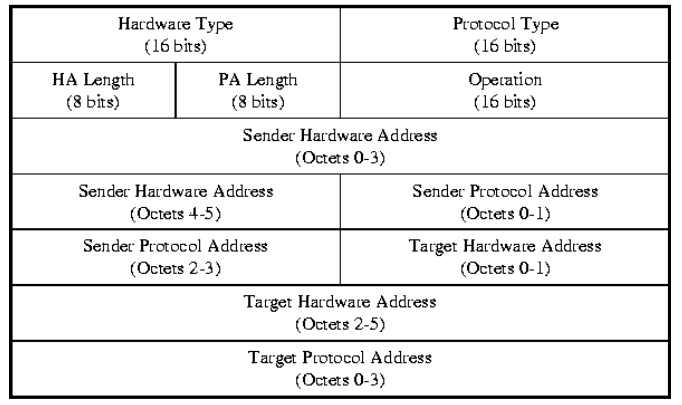
**TCP/IP Stack: TCP Protocol Header:**

** **

**TCP Handshake:**

** **

**ARP Protocol:**

****

Intrusion Detection System (IDS)

**Intrusion:** a set of actions aimed to compromise the security goals **||Intrusion detection:** the process of identifying and responding to intrusion activities **|| IDS Assumptions:** system activities are observable; normal and intrusive activities have distinct evidence **|| Intrusion Detection Approaches:** modeling evidences extracted from audit data; analysis approach: piecing the evidences together **|| Host-based IDS (HIDS):** using OS auditing mechanisms; BSM on Solaris logs all direct or indirect events generated by a user; *strace* for system calls made by a program; monitoring user activities and analyze shell commands; monitor executions of system program by analyzing system calls made by sendmail **|| Network IDS (NIDS):**  deploying sensors at strategic locations; inspecting network traffic; monitoring user activities; may be easily defeated by encryption **|| Firewall vs. Network IDS:**  firewall is active filtering and fail-close; NIDS is passive monitoring and fail open **|| Requirements of Network IDS:**  high-speed, large volume, real time notification; extensible; broad detection coverage; economy in resource usage; resilience to stress; resilience to attacks upon the IDS itself

Firewalls

**Firewall:** controlled access point; restrict incoming/outgoing traffic by IP address, ports or users; block invalid packets; cannot protect traffic that doesn’t go through it or when misconfigured **|| Filtering Firewall:**  packets checked then passed; inbound & outbound affect when policy is checked; packet filtering (access control lists); Session filtering (dynamic packet filtering, stateful inspection) **||Packet Filtering:**  decision made on per-packet basis; no state information saved; ports > 1024 left open; if dynamic protocols are in use, entire ranges of ports must be allowed **|| FTP Protocol:** client opens command channel to server; tells server second port number; server validates and opens data channel to the specified second port; client validates **|| Session Filtering:** packet decision made in context of connection; if packet is new connection, check against security policy; if packet is part of an existing connection, match it up in the state table & update table; usually all access denied unless allowed; Dynamic protocols allowed if supported **|| FTP Passive Mode:** client opens command channel to server & requests packet mode; server opens port for data channel and tell client; client opens channel on specified port; server validates **|| Proxy Firewalls:** Relay for connections; Client <> Proxy <> Server; application and circuit level **||Application level Gateway (ALG):** understands specific applications; limited proxies available; proxy impersonates both sides of connection; Resource intensive; HTTP proxies may cache web pages; Proxy Firewall; Telnet, FTP, SMTP, HTTP; more secure than packet filters; simplifies rules needed in packet filter; more appropriate to TCP, ICMP difficult; block all unless allowed **||Application level Gateway Operation:** client connects; gateway does in-depth inspection of the application level packet, if connection meets criteria on the gateway, rule base packet will be proxied to the server; proxy firewall is directly between the client and the server on an application by application basis **|| Circuit Level Gateways (CLG):**  stateful inspection firewall; Session layer of OSI; Shim between transport and application layer of TCP/IP; monitors handshake used to establish connections; hides information about internal network; breaks TCP connection; support more services than application level gateway; hard to handle protocols like FTP; clients must be aware they are using a CLG; protect against fragmentation problem **|| SOCKS (sockets):** generic protocol for TCP/IP; provides framework for developing secure communications by easily integrating other security tech; works for TCP & UDP; transparent network access across multiple proxy servers; easy deployment of auth & encryption; application independent **|| Proxying UDP/ICMP:** TCP is proxied more than UDP/ICMP b/c TCP is connection oriented **|| Network Address Translation (NAT):** useful if org doesn’t have enough IP addr; harder to trick firewall; only need real IP address for services outside networks will originate from; Many-to-1 mapping (n-to-m); 1-to-1 (n-to-n) mapping

