Security

Arvind Krishnamurthy
Fall 2003

Secure Routing

- Problem statement:
  - A source node "s" wants to send a packet to a destination node "d" through a network that might have Byzantine nodes
  - Primary requirement: Routing should be successful as long as there is one good path with no Byzantine nodes
  - Secondary requirements:
    - Try to use the best path with no Byzantine nodes
    - Try to minimize the amount of traffic generated

Basic Necessities

Cannot perform secure routing without the following:

1) Digital signatures:
   - When "s" sends the message "m", it actually sends:
     m E(MD5(m), Priv-K^s)
   - Where MD5 is the message-digest hash function. Can use other hash functions such as SHA-1
   - And the hash value is encrypted using the source's private key
   - All nodes can decrypt this using source's public key
2) Need pre-allocated buffers:
   - Each node has space for some number of packets being routed from a certain source
   - Packet is dropped if a source sends too many packets
   - For "ascribing blame"; need to separate Byzantine behavior from congestion
   - To avoid denial of service attacks

Routing

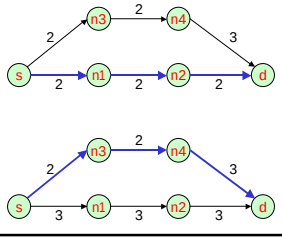
- Option 1: (Perlman's 1989 thesis)
  - Flood the network for each packet
    - Eliminate duplicates
    - Each packet will eventually find a good path
    - Pre-allocated buffers guarantee delivery
  - Problem:
    - Too much traffic
    - Packets get queued up and latency could be badly affected

Option 2

- Use link state protocol
  - Gather link information and compute good paths
  - Perform source-routing
    - Route included along with each packet
    - Routes are also digitally signed
  - Each node checks whether:
    - It is in the route determined by the source
    - Whether it received it from the correct predecessor node
  - Packet also include monotonically increasing sequence numbers
    - Good nodes remember the last sequence number received from each source
    - Avoids replay attacks

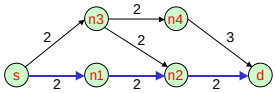
Use Time-outs

- Destination sends acknowledgement (also digitally signed and with appropriate route and sequence number)
- Source uses time-outs for packets
  - If a packet times-out, penalize every link (or node) along that path



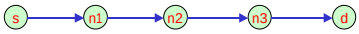
Too Heavy-handed

- Might ignore other paths that use some of the good links of the original path
  - If "n1" is defective, you can still use "n2"
  - Would like more fine-grained detection of faults



Option 3: Explicit fault announcements

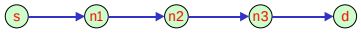
- Each intermediate node has a time-out
- A node generates FA (fault announcement) if:
  - Does not receive an acknowledgement within a certain period of time
- A node drops a packet if:
  - It receives a corrupted packet from the predecessor
- A faulty node could generate FA to accuse the successor node in the path
  - If n1 generates FA, then n1 could be bad or n2 could be bad
  - Who else could be bad in the system?



Alternative to Digital Signatures

- Digital signatures are expensive
  - Hash functions are cheap
  - Encryption using RSA is expensive
- Alternative: Message Authentication Codes using keyed-hash functions
  - Examples: H-MAC, UMAC, etc.
  - Think of it as a meta-hash-function that takes two inputs:
    - A secret key
    - A hash function (such as MD5, SHA-1)
    - Key is used during the hashing process
- Problem with this approach:
  - Use of a secret key means that the MAC value is valid only for a pair of nodes
  - Every node requires to have a separate shared secret with every other node

Sequential Message Authentication Codes



Packet data: M	MAC(d)	MAC(n3)	MAC(n2)	MAC(n1)

MAC(d) = H(M, secret-key between s and d)

MAC(n3) = H(M + MAC(d), secret-key between s and n3)

MAC(n2) = H(M + MAC(d) + MAC(n3), secret-key between s and n2)

- When a node receives the packet:
  - It checks for the integrity of the message and the integrity of the MAC values for subsequent nodes

Reason for sequential MACs

- Prevent the following kind of attack:
  - n1 is malicious:
    - It receives the packet from s, does not change packet data
    - Instead, it corrupts the MAC for n3
    - n3 receives a packet whose MAC does not match the data and drops the packet
    - n2 times-out and generate FA
      - Source thinks that n2 or n3 is bad
      - But the real bad node is n1
  - With sequential MACs:
    - n2 realizes that n3's MAC has been corrupted
    - n2 drops the packet
    - Either n1 times-out and generate FA or simply does nothing (since it is evil) – in which case, the source times-out

Sharing Fault Knowledge

- A node "s" could claim that a link (u, v) is faulty
  - Either node "s" is faulty
  - Or the link (u, v) is faulty
- When node "t" receives such a report:
  - It could attempt to find a path that does not contain both "s" and the link (u, v)
  - If it has a number of such reports: [s1, (u1, v1)], [s2, (u2, v2)], etc., then it could find paths such that the paths contains only one of each pair
  - Becomes an NP-complete problem

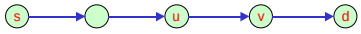
Special Conditions for Sharing

- Between adjacent routers:
  - Let “s” send a packet through r, ..., u, v, ..., d
  - Receives FA from “u”
  - It can report this information to “r”
- “r” cannot use both “s” and the link (u, v) in a path
  - When computing shortest path to some node “t”
  - It removes the link (s, r) and computes the shortest path
  - Or it reinstates the link (s, r) and removes all links about which “s” has sent accusations and then computes the shortest path
- In our example:
  - “s” could also tell “u” and “v” that (u, v) is faulty
  - A similar construction is used at “u” and “v”

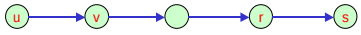
Other Issues

- Holding on to packets for too long:
  - And not long enough to generate time-outs
  - Tough problem: could ask intermediate routers to send delay values along with acknowledgement
  - Compare the sum of delay values with some estimates and declare a fault if there is too much difference
  - Or feed the delay values into the costs of edges before calculating shortest paths

Blocking Traffic



- Let “s” determine that (u, v) is faulty
- Assume that “s” receives a packet whose first link traversed (u, v)



- “s” could decide to drop this packet (pay-back time)
- “r” sends FA to “u”
- “u” would consider both “r” and “s” as defective
  - There is a growing loss of trust in the system

Alternative

- “s” sends a blocking announcement (BA) message to “u”
  - “u” decides to just block “s” and not assign any blame to “r”
  - “v” could have dropped the BA, but then “u” would start suspecting “v”
  - Now we have stability: “s” mistrusts “u” and blocks its traffic, “u” mistrusts “s” and blocks its traffic

Security Issues in Traditional Systems

- Exploiting software bugs, buffer overflows
- Security holes in protocols
- Denial of service attacks
- Solutions:
  - IP traceback
  - Firewalls
  - Authentication

Basic IP

- End hosts create IP packets and routers process them purely based on destination address alone
- Problem – End host may lie about other fields without affecting delivery
  - Source address – host may trick destination into believing that packet is from trusted source
    - Many applications use IP address as a simple authentication method (for example, “.rhosts” based authentication)

Other IP Issues

- Source routing
  - Source provides an explicit path
  - Destinations are expected to reverse source route for replies
    - Reasonable if the source wants to specify a path because the automatic route is dead
  - Problem – Can force packets to be routed through convenient monitoring point
  - Solutions:
    - reverse path forwarding checks
    - better authentication
    - disallow source routing
- Fragmentation – can consume memory resources or otherwise trick destination/firewalls
  - Solution – disallow fragments

Routing

- Routing protocol
  - Malicious hosts may advertise routes into network
  - Problem – Bogus routes may enable host to monitor traffic or deny service to others
    - Solutions
      - Use policy mechanisms to only accept routes from or to certain networks/entities
      - Routing registries and certificates
- DNS
  - Users/hosts typically trust the host-address mapping provided by DNS
  - Problems
    - Interception of requests or comprise of DNS servers can result in bogus responses
    - Solution – authenticated requests/responses
    - Zone transfers can provide useful list of target hosts

ICMP

- Reports errors and other conditions from network to end hosts
- An entity can easily forge a variety of ICMP error messages
  - Redirect – informs end-hosts that it should be using different first hop route (can be sent only by the gateway at the first hop)
  - Fragmentation – can confuse path MTU discovery
  - Destination unreachable – can cause transport connections to be dropped
- Solution: perform validity checks
  - Limit the attack to existing connections

TCP

- Each TCP connection has an agreed upon/negotiated set of associated state
  - Starting sequence numbers, port numbers
  - Knowing these parameters is sometimes used to provide some sense of security
- Problem
  - Easy to guess these values
    - Listening ports #'s are well known and connecting port #'s are typically allocated sequentially
    - Starting sequence number are chosen in predictable way
  - Solution – make sequence number selection more random

Sequence Number Guessing Attack

Attacker → Victim: SYN(ISN_x), SRC=Trusted Host
Victim → Trusted Host: SYN(ISN_s), ACK(ISN_x)
Attacker → Victim: ACK($ISN_{guess\ of\ s}$), SRC=Trusted Host
Attacker → Victim: ACK($ISN_{guess\ of\ s}$), SRC=T, data = "rm -r /"

- Attacker must also make sure that Trusted Host does not respond to SYNACK
- Can repeat until guess is accurate

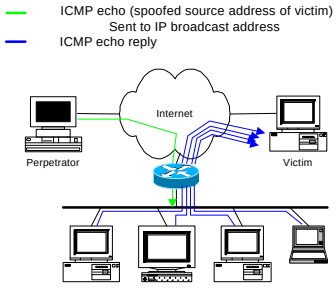
Denial of Service

- Objective of attack: make a service unusable, usually by overloading the server or network
- Example: SYN flooding attack
  - Send SYN packets with bogus source address
  - Server responds with SYNACK keeps state about TCP half-open connection
    - Eventually server memory is exhausted with this state
  - Solution: SYN cookies – make the SYNACK contents purely a function of SYN contents, therefore, it can be recomputed on reception of next ACK

Distributed Denial of Service Attacks

- Attack coming from a distributed set of computers
- These computers are either:
  - Previously compromised machines
  - Reflector machines
- Reflector attack:
  - Pick a target to attack
  - Send packets to a bunch of other legitimate machines claiming to be sent from the target
    - Send it at a slow enough rate so that the machines do not recognize this (send SYN packets for instance)
    - Legitimate machines reflect packet to the target
    - Results in target flooding
    - One example: send requests to BGP port of internet routers

Smurf attack

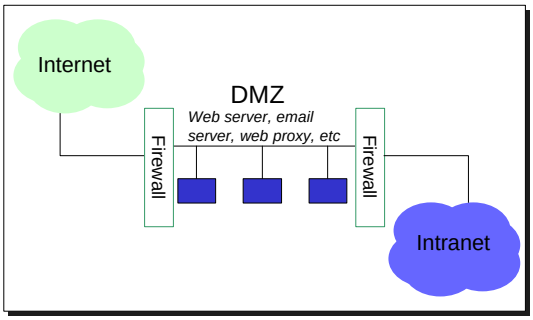


- Source IP address of a broadcast ping is forged
- Large number of machines respond back to victim, overloading it

Firewalls

- Basic problem – many network applications and protocols have security problems that are fixed over time
  - Difficult for users to keep up with changes and keep host secure
  - Solution
    - Administrators limit access to end hosts by using a firewall
    - Firewall and limited number of machines at site are kept up-to-date by administrators

Typical Firewall Topology



Types of Firewalls

- Proxy
  - End host connects to proxy and asks it to perform actions on its behalf
    - Policy determines if action is secure or insecure
  - Transport level relays (SOCKS)
    - Ask proxy to create, accept TCP (or UDP) connection
    - Cannot secure against insecure application
  - Application level relays (e.g. HTTP, FTP, telnet, etc.)
    - Ask proxy to perform application action (e.g. HTTP Get, FTP transfer)
  - Requires applications (or dynamically linked libraries) to be modified to use the proxy
  - Considered to be the most secure since it has most information to make decision

Types of Firewalls

- Packet filters
  - Set of filters and associated actions that are used on a packet by packet basis
  - Filters specify fields, masks and values to match against packet contents, input and output interface
  - Actions are typically forward or discard
  - Typically a difficult balance between the access given and the ability to run applications
    - E.g. FTP often needs inbound connections on arbitrary port numbers – either make it difficult to use FTP or limit its use
- Stateful packet filters
  - Actions can include the addition of new rules and the creation of state to process future packets
    - Often have to parse application payload to determine “intent” and determine security considerations
  - Rules can be based on packet contents and state created by past packets

Bandwidth DOS Attacks

- Possible solutions
  - Ingress filtering – examine packets to identify bogus source addresses
  - Link testing – have routers either explicitly identify which hops are involved in attack or use controlled flooding and a network map to perturb attack traffic
  - Logging – log packets at key routers and post-process to identify attacker's path
  - IP traceback

IP Traceback

- Node append (record route) – high computation and space overhead
- Node sampling – each router marks its IP address with some probability *p*
  - P(receiving mark from router d hops away) = $p(1-p)^{d-1}$
  - Must infer distance by marking rate → relatively slow
  - Doesn't work well with multiple routers at same distance → I.e. multiple attackers

IP Traceback

- Edge sampling
  - Solve node sampling problems by encoding edges & distance from victim in messages
  - Start router sets "start" field with probability *p* and sets distance to 0
  - If distance is 0, router sets "end" field
  - All routers increment distance
  - As before, P(receiving mark from router d hops away) = $p(1-p)^{d-1}$
- Multiple attackers can be identified since edge identifies splits in reverse path

Edge Sampling

- Major problem – need to add about 72bits (2 address + hop count) of info into packets
- Solution
  - Encode edge as xor of nodes → reduce 64 bits to 32 bits
  - Ship only 8bits at a time and 3bits to indicate offset → 32 bits to 11bits
  - Use only 5 bit for distance → 8bits to 5bits
  - Use IP fragment field to store 16 bits
    - Some backward compatibility issues
    - Fragmentation is rare so not a big problem

Authentication: Secure Sockets Layer

- Transport layer security to any TCP-based app using SSL services.
- Used between Web browsers, servers for e-commerce (shttp).
- Security services:
  - Server authentication
  - Data encryption
  - Client authentication (optional)
- Server authentication:
  - SSL-enabled browser includes public keys for trusted CAs.
  - Browser requests server certificate, issued by trusted CA.
  - Browser uses CA's public key to extract server's public key from certificate.

SSL (continued)

Encrypted SSL session:

- Browser generates *symmetric session key*, encrypts it with server's public key, sends encrypted key to server.
- Using private key, server decrypts session key.
- Browser, server know session key
  - All data sent into TCP socket (by client or server) encrypted with session key.
- Bootstrapping happens because there is a CA and CA knows of all servers
- If you want to communicate between two non-certified entities (for example, using ssh) then there is no clean solution
- ssh:
  - Each node picks a random value
  - Computes some function of the random value and communicates that to the other node
  - Both nodes can compute a session key (and others eavesdropping the communication cannot)
  - Prone to man-in-the-middle attack

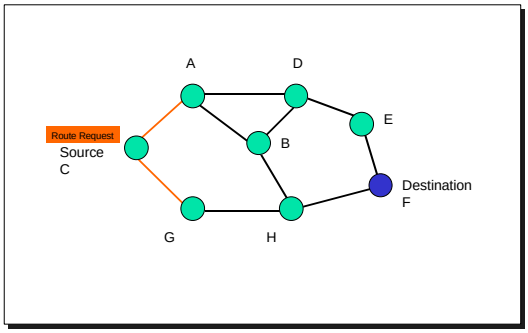
Security for ad-hoc networks

- Problems that are tough to solve w/o hardware changes:
  - Jamming frequencies (physical layer assault)
  - Sending continuous CTS signals (link layer assault)
- Focus primarily on route discovery/maintenance
  - Recall that traditional link-state and distance vector protocols do not work well with ad-hoc networks
  - Have a system whose topology is constantly changing

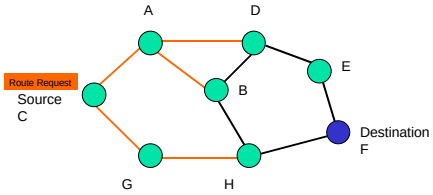
Dynamic Source Routing (DSR)

- Route discovery
  - **Source** broadcasts route-request to **Destination**
  - Each node forwards request by adding own address and re-broadcasting
  - Requests propagate outward until:
    - Target is found, or
    - A node that has a route to Destination is found
  - Requests not forwarded if:
    - Node already listed in recorded source route
    - Node has seen request with same sequence number
    - IP TTL field may be used to limit scope
  - Destination copies route into a Route-reply packet and sends it back to **Source**

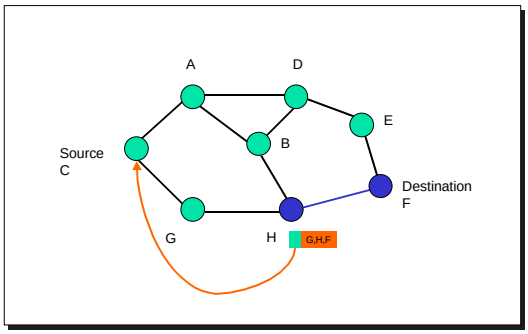
C Broadcasts Route Request to F



C Broadcasts Route Request to F



H Responds to Route Request



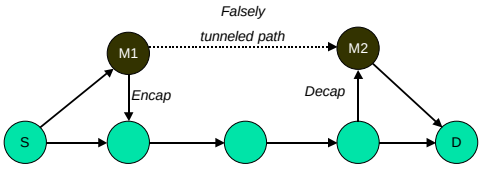
Sending Data

- Check cache for route to destination
- If route exists then
  - If reachable in one hop
    - Send packet
  - Else insert routing header to destination and send
- If route does not exist, buffer packet and initiate route discovery

Possible Attacks

- Data packets can be exploited in ways discussed before:
  - Corrupted, dropped (black-hole/grey-hole), mis-routed, replayed
  - In ad-hoc setting worry also about:
    - Power resources
    - Memory resources
- We will focus on route-discovery process. The possible attacks are:
  - Attacker advertises shorter routes (discards elements from the route accumulated in the packet)
  - Corrupts the route accumulated in the packet
  - Tunneling attacks: make paths look shorter (worm-hole attack)
  - Spoof routing and error messages (claim to be someone else)

Tunneling



Secure Route Discovery

- Assume that we can use public key cryptography
  - There is a certificate server (T)
  - Can communicate with the certificate server using its public key
  - Any value published by T signed with its private key is authentic
- How do we design a secure route discovery protocol using such a certificate server?

Tesla Key Management Scheme

- Primary motivation: avoid expensive digital signatures, use one-way hash functions to generate keys
  - For broadcast mechanisms
  - Delayed key disclosure (requires clock synchronization)
- One-way key chain
  - Each sender chooses random initial key K_N and generates one-way key chain as $K_i = H^{N-i}(K_N)$
  - Use keys in normal increasing order (K_0 K_1 K_2 ...)
- Schedule for disclosing keys
  - Disclose K_i at $T_i = T_0 + i \times t$

Key publication interval

Overview of TESLA

- Receiver can determine which key is disclosed
  - Based on loose time synchronization (Δ)
  - Let K_i be used to authenticate a packet
  - If current time $\leq t_0 + i \times t - \Delta$ implies K_i is not disclosed yet
  - Discard the packet if security condition fails
- Sender:
  - Let τ be worst-case end-to-end network delay
  - Sender picks K_i which will not be disclosed until $\tau + 2\Delta$ time passes and add MAC using K_i
- TESLA security condition
  - τ is small $\rightarrow$ may discard some packets
    τ is large $\rightarrow$ long delay for authentication
    τ does not affect security

Ariadne Assumptions

- Security assumptions
  - Pairwise shared secret keys
    - Set up n (n+1) / 2 keys for n nodes
  - TESLA
    - Set up shared secret keys between communicating nodes, distribute one authentic public TESLA key for each node
  - Digital signatures
    - Distribute one authentic public key for each node
  - A node trusts only itself
    - Avoid blackmail attacks
  - No secrecy or confidentiality

Route Discovery

- Securing ROUTE REQUEST
  - Target can authenticate the sender (using their shared secret key)
  - Initiator can authenticate each path entry using intermediate TESLA keys (message integrity checks)
  - No intermediate node can remove any other node in the REQUEST or REPLY
    - One-way hash functions verify that no hop was omitted (*per-hop hashing*)
- ROUTE REQUEST packet contains eight fields:
  - ROUTE REQUEST: label
  - *initiator*: address of the sender
  - *target*: address of the recipient
  - *id*: unique identifier
  - *time interval*: TESLA time interval of the pessimistic arrival time
  - *hash chain*: sequenced MAC hash
  - *node list*: sequence of nodes on the path
  - *MAC list*: MACs of the message using TESLA keys

Route Discovery (TESLA)

Route Request

Route to be found: $S \rightarrow A \rightarrow B \rightarrow C \rightarrow D$

$M = \langle \text{Request}, S, D, id, ti \rangle$

$S : h_0 = \text{MAC}_{K_{SD}}(M)$

$S \rightarrow * : \langle M, h_0, (), () \rangle$

$A : h_1 = H(A, h_0)$

$M_A = \text{MAC}_{K_{A_i}}\langle M, h_1, (A), () \rangle$

$A \rightarrow * : \langle M, h_1, (A), (M_A) \rangle$

$B : h_2 = H(B, h_1)$

$M_B = \text{MAC}_{K_{B_i}}\langle M, h_2, (A, B), (M_A) \rangle$

$B \rightarrow * : \langle M, h_2, (A, B), (M_A, M_B) \rangle$

$C : h_3 = H(C, h_2)$

$M_C = \text{MAC}_{K_{C_i}}\langle M, h_3, (A, B, C), (M_A, M_B) \rangle$

$C \rightarrow * : \langle M, h_3, (A, B, C), (M_A, M_B, M_C) \rangle$

Route Reply

$M = \langle \text{Reply}, D, S, ti, (A, B, C), (M_A, M_B, M_C) \rangle$

$D : M_D = \text{MAC}_{K_{DS}}(M)$

$D \rightarrow C : \langle M, M_D, () \rangle$

$C \rightarrow B : \langle M, M_D, (K_{C_i}) \rangle$

$B \rightarrow A : \langle M, M_D, (K_{C_i}, K_{B_i}) \rangle$

$A \rightarrow S : \langle M, M_D, (K_{C_i}, K_{B_i}, K_{A_i}) \rangle$

Route Discovery Description

- Upon receiving ROUTE REQUEST, a node:
  - Processes the request only if it is new
  - Processes the request only if the time interval is valid (not too far in the future, but not for an already disclosed TESLA key)
  - Modifies the request and rebroadcasts it
    - Appends its address to the node list
    - Replaces the *hash chain* with H[A, *hash chain*]
    - Appends MAC of entire REQUEST to *MAC list* using K_{Ai} where i is the index for the time interval specified in the REQUEST

Route Discovery (cont.)

- When the target receives the route request:
  - Checks the validity of the REQUEST
    - Check that the keys from the time interval have not been disclosed yet
    - Check that *hash chain* is correct
  - Returns ROUTE REPLY containing eight fields
    - ROUTE REPLY, *target, initiator, time interval, node list, MAC list*
    - *target MAC*: MAC computed over above fields with key shared between target and initiator
    - *key list*: disclosable MAC keys of nodes along the path
- Node forwarding ROUTE REPLY
  - Waits until it can disclose TESLA key from specified interval
    - Appends that key to the *key list*
    - This waiting does delay the return of the ROUTE REPLY but does not consume extra computational power

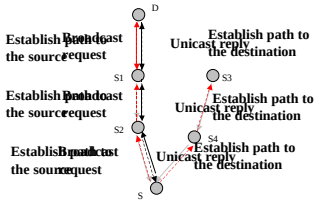
Route Discovery (cont.)

- When initiator receives ROUTE REPLY
  - Verifies each key in the key list is valid
  - Verifies that the *target MAC* is valid
  - Verifies that each MAC in the *MAC list* is valid using the TESLA keys

Ad Hoc On-Demand Distance Vector (AODV)

- DSR includes source routes in packet headers
- Resulting large headers can sometimes degrade performance
  - particularly when data contents of the packet are small
- AODV:
  - Hop-by-hop routing; routing/forwarding tables at each node
  - On-demand (just as DSR)
- Route Requests are forwarded in a manner similar to DSR
  - Requests and replies carry hop count information
  - Routes have a sequence number (which the destination increases monotonically)
  - Search specifies a lower bound for the sequence number

Route Discovery in AODV



Intermediate nodes could generate a RREP if they know route to destination

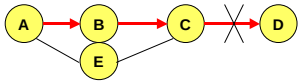
If multiple paths are found, use the shortest path based on hop-count

Data Delivery in AODV

- Data is forwarded based on routing tables
  - Unlike DSR, data packets do not contain entire path information
- Route errors:
  - When node X is unable to forward packet P (from node S to node D) on link (X,Y), it generates a RERR message
  - Node X adds one to the destination sequence number for D and includes this value (N) in the RERR
  - When node S receives the RERR, it initiates a new route discovery for D using destination sequence number at least as large as N

Why Sequence Numbers in AODV?

- To avoid using old/broken routes
  - To determine which route is newer
- To prevent formation of loops



  - Assume that A does not know about failure of link C-D because RERR sent by C is lost
  - Now C performs a route discovery for D. Node A receives the RREQ (say, via path C-E-A)
  - Node A will reply since A knows a route to D via node B
  - Results in a loop (for instance, C-E-A-B-C)

Secure AODV

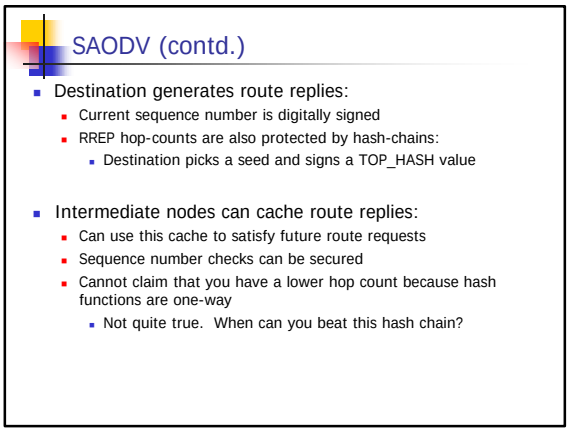
- Assume that you can use digital signatures and/or shared secret keys and/or one-way hash functions
- How do we secure AODV?

SAODV Routing Protocol (by Zapata)

- Protocol messages have components of two different types:
  - Immutable fields: can be secured by digital signatures
  - Mutable fields
- For mutable fields, we can use hash-chains
  - Attackers often modify Hop Count of a RREQ
  - Hash chains are used to protect this Hop Count field
  - A hash chain is formed by applying a one-way hash function (h) repeatedly to a seed
  - Let's say that you have a digitally signed version of $h^n(x)$
    - Start with value "x"
    - At each hop, apply "h" to the current value
    - We will use $h^{n-k}(x)$ as a proof that the packet has traveled k hops

SAODV

- Source picks random seed s
- Source calculates TOP_HASH = $h^{max}(s)$ where "max" is the maximum hop count to be tolerated by the routing protocol
- Source signs TOP_HASH with its digital signature
- Generates a RREQ:
  - Sends TOP_HASH
  - Initializes hop count to 0
  - Sends "s" as current hash value
- Each intermediate node:
  - When it receives <TOP_HASH, hops, curr_hash>, checks:
    - Whether $h^{max-hops}$(curr_hash) == TOP_HASH
  - Propagates < TOP_HASH, hops + 1, h(curr_hash)>


SAODV (contd.)
Destination generates route replies:
Current sequence number is digitally signed
RREP hop-counts are also protected by hash-chains:
Destination picks a seed and signs a TOP_HASH value
Intermediate nodes can cache route replies:
Can use this cache to satisfy future route requests
Sequence number checks can be secured
Cannot claim that you have a lower hop count because hash functions are one-way
Not quite true. When can you beat this hash chain?