Advanced Security Constructions and Key Management

Class 16

Outline

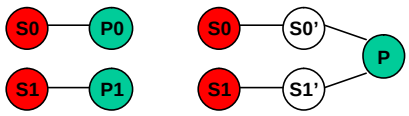
- One-Time Signatures
  - Lamport's signature
  - Improved signature constructions
  - Merkle-Winternitz Signature
- Efficient Authenticators (amortize signature)
  - One-way chains (self-authenticating values)
  - Chained hashes
  - Merkle Hash Trees
- Applications
  - Efficient short-lived certificates, S/Key
  - Untrusted external storage
  - Stream signatures (Gennaro, Rohatgi)
- Zhou & Haas's key distribution

One-Time Signatures

- Challenge: digital signatures expensive for generation and verification
- Goal: amortize digital signature

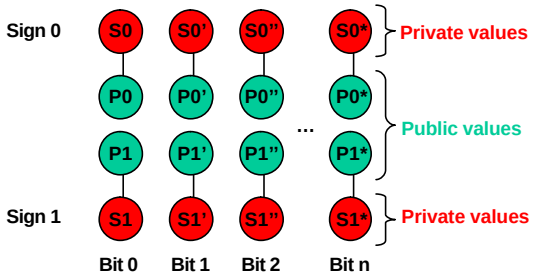
One-Time Signatures

- Use one-way functions without trapdoor
- Efficient for signature generation and verification
- Caveat: can only use one time
- Example: 1-bit one-time signature
  - P0, P1 are public values (public key)
  - S0, S1 are private values (private key)



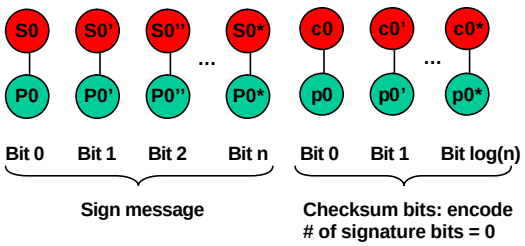
Lamport's One-Time Signature

- Uses 1-bit signature construction to sign multiple bits



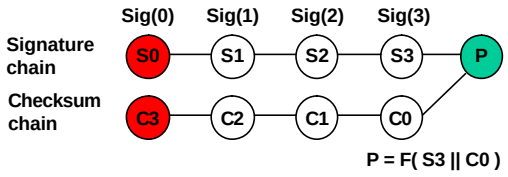
Improved Construction I

- Uses 1-bit signature construction to sign multiple bits



Improved Construction II

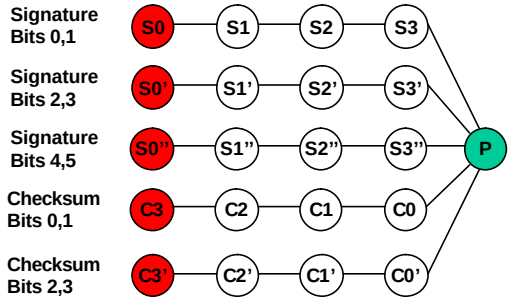
- Lamport signature has high overhead
- Goal: reduce size of public and private key
- Approach: use one-way hash chains
- S1 = F(S0)



P = F(S3 || C0)

Merkle-Winternitz Construction

- Intuition: encode sum of checksum chain



Efficient Authenticators

- One-way chains
- Chained hashes
- Merkle hash trees

Recall One-Way Hash Chains?

- Versatile cryptographic primitive
- Construction
  - Pick random r_N and public one-way function F
  - $r_i = F(r_{i+1})$
  - Secret value: r_N , public value r_0

$r_3 \xleftarrow{F} r_4 \xleftarrow{F} r_5 \xleftarrow{F} r_6 \xleftarrow{F} r_7$

- Properties
  - Use in reverse order of construction: r_1 , r_2 ... r_N
  - Infeasible to derive r_i from r_j (j<i)
  - Efficiently authenticate r_i knowing r_j (j<i): verify $r_j = F^{j-i}(r_i)$
  - Robust to missing values

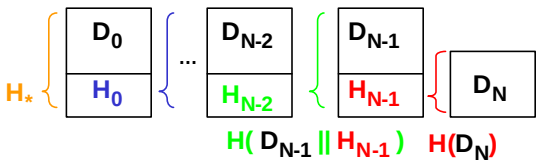
One-Way Chain Application

- S/Key one-time password system
- Goal
  - Use a different password at every login
  - Server cannot derive password for next login
- Solution: one-way chain
  - Pick random password P_L
  - Prepare sequence of passwords $P_i = F(P_{i+1})$
  - Use passwords P_0 , P_1 , ..., P_{L-1} , P_L
  - Server can easily authenticate user

$p_3 \xleftarrow{F} p_4 \xleftarrow{F} p_5 \xleftarrow{F} p_6 \xleftarrow{F} p_7$

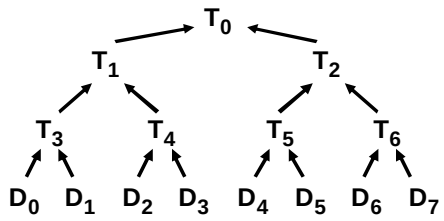
Chained Hashes

- More general construction than one-way hash chains
- Useful for authenticating a sequence of data values D_0 , D_1 , ..., D_N
- H_* authenticates entire chain



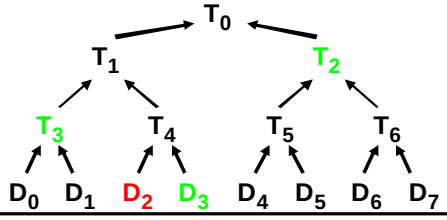
Merkle Hash Trees

- Authenticate a sequence of data values $D_0, D_1, \ldots, D_N$
- Construct binary tree over data values



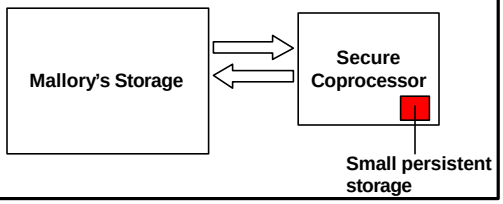
Merkle Hash Trees II

- Verifier knows T_0
- How can verifier authenticate leaf D_i ?
- Solution: recompute T_0 using D_i
- Example authenticate D_2 , send D_3 T_3 T_2
- Verify $T_0 = H(H(T_3 \| H(D_2 \| D_3)) \| T_2)$



Untrusted External Storage

- Problem: how can we store memory of a secure coprocessor in untrusted storage?
- Solution: construct Merkle hash tree over all memory pages

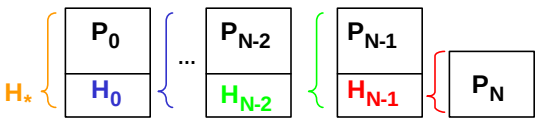


Stream Signatures

- Gennaro & Rohatgi, Crypto '97
- Problem
  - Sender sends a sequence of packets to receiver
  - Receiver wants to immediately authenticate each packet
  - Efficient authentication of packets
  - On-line case (real-time data), off-line case (stored data)

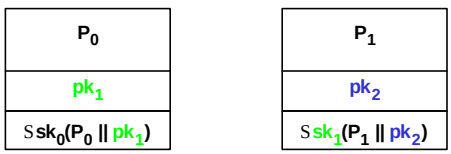
Off-line Case

- Sender know entire stream before sending
- Use chained hashes, precompute H_i
- Digitally sign the first packet $S(H_*)$
- Each packet authenticates the next packet



On-line Case

- Use a one-time signature to authenticate packets
  - Sender has regular signature (SK,PK)
  - Sender signs public key of one-time signature $S_{SK}(pk0)$
  - Sign packet P_i and one-time public key pk_i with pk_{i-1}

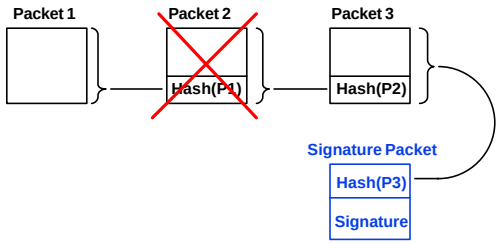


Stream Signature Discussion

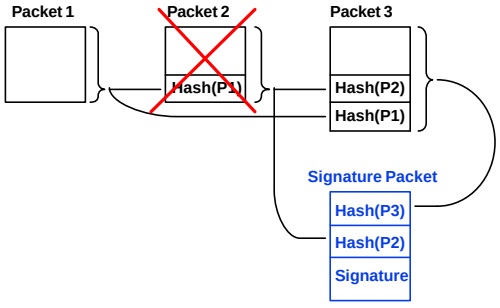
- Computation and communication cost
- Robustness to DoS attack (packet injection)
- Robustness to packet loss
  - Loss of a single packet prevents authentication of subsequent packets
  - How could we improve the loss robustness?

Alternative Stream Signature

- Add hashes to later packets
- Periodically send a signature packet



Improving Robustness



Securing Ad Hoc Networks

- Zhou & Haas, IEEE Network Magazine '99
- Security goals
  - Availability
  - Confidentiality
  - Integrity
  - Authentication
- Secure Routing
- Key management

Attacker Assumptions

- Attacker can physically compromise nodes
- "Mobile Adversary"
  - Adversary can compromise any node
  - Temporarily compromises node, then moves on to next node
  - Every node may be compromised at one time
- Attacker compromises at most t nodes at any one moment

Secure Routing

- Authenticate all routing messages, to prevent external attackers
- Proposes to use multiple paths to tolerate internal attackers
  - Drawback: internal attackers could easily fake multiple paths

Key Management Service

- Consider public-key infrastructure (PKI)
  - Everybody trusts certification authority (CA)
  - CA authenticates and signs public keys of other nodes
- PKI drawbacks
  - Revocation requires on-line PKI
  - Single point of failure, CA replication increases vulnerability to node compromise
- Solution: distributed CA

Distributed CA Model

- Private CA key is shared among set of nodes
  - Signing needs coalition of t+1 correct nodes
  - Secret sharing prevents t malicious nodes from reconstructing CA private key
- Requirements for key management service
  - Robustness: service available to answer requests correctly
  - Confidentiality: adversary never learns CA private key

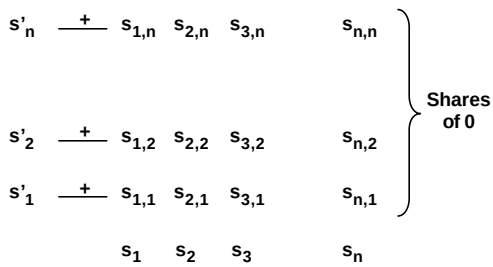
Threshold Cryptography

- Share secret S among n nodes, require t+1 nodes for reconstruction
  - (n, t+1) secret sharing scheme
- Share private key K among n nodes, require t+1 nodes for signing
  - (n, t+1) threshold signature scheme
  - Node i gets share k_i
  - For signing, nodes send partial signature to combiner
  - Combiner collects 2t+1 partial signatures

Proactive Security

- Use share refreshing against mobile adversaries
- If $(s_1, s_2, \ldots, s_n)$ is a sharing of k, and $(s'_1, s'_2, \ldots, s'_n)$ is a sharing of k', then $(s_1 + s'_1, s_2 + s'_2, \ldots, s_n + s'_n)$ is a correct sharing of k + k'
- Trick, set k' = 0, so new sharing also represents k

Share Refreshing



Discussion

- How can share refreshing tolerate faulty nodes?
- How can we tolerate compromised combiner?
  - Who decides to be a combiner?
- How can we bootstrap this system?
  - How can we introduce a new node?
- Why should node sign a message?
  - How does node authenticate message?
- Is signature combination expensive if we have t faulty nodes?