Zero-knowledge Proofs and Authentication

Dawn Song

dawnsong@cs.berkeley.edu

Review

- **Secret-sharing**
  - **How does a (n,t) threshold secret sharing scheme work?**
- **Zero-knowledge proof**

How to prove knowledge of square root (I)

- **Finding square root mod N=pq is as hard as factoring**
- **A knows b s.t. $b^2 = y \mod pq$, & wishes to prove to B that she knows such b.**
- **A → B: $s =: r^2 \mod pq$ (A picks random r)**
- **B flips coin**
- **B → A: coin flip**
- **If heads**
  - **A → B: $t =: r \mod pq$**
  - **B verifies $t^2 \equiv s \mod pq$**
- **If tails**
  - **A → B: $t =: rb \mod pq$**
  - **B verifies $t^2 \equiv sy \mod pq$**
- **What if A didn't know the square root?**
- **What did B learn after the proof?**

How to prove knowledge of square root (II)

- **What if A could predict B's coin flip?**
- **What if A reuses random number r in different rounds?**
- **How is B convinced that A does know the square root?**
  - **Knowledge extractor**
- **Why is B not learning anything about the square root?**
  - **Simulator argument (out of scope)**

Administrative Matters

- **Hw1 statistics:**

Mean: 34.6
Standard deviation: 10.8
1st quartile: 29.0
2nd quartile (median): 34.0
3rd quartile: 44.0
Maximum: 54.0

Authentication

- **Alice and Bob love each other, but they live far apart**
- **We've learned how they can encrypt their messages**
- **How can they make sure they are talking to each other?**
- **This is the question of authentication**

Types of authentication

- End user → End user (Alice & Bob)
- End user → Local computer (login)
- End user → Remote computer (web site login)
- Computer → Computer (DRM)
- Local computer → End user (fake ATM check)
- Remote computer → End user (phishing check)

Basic Security Protocols

- Entity authentication protocols
  - Prove identity to each other
- Key establishment/agreement/distribution protocols
  - Establish a trusted session key between two principals
  - Usually used to set up trusted communication channel providing secrecy and authenticity
- Other protocols: secure e-commerce, e-voting, time synchronization, etc.
- We use our basic cryptographic primitives to design higher-level security properties

Protocol Design Basics

- Protocols involve principals, e.g., hosts, users, services, processes
- Secret information, e.g., symmetric keys, private keys
- Authentic information, e.g., public keys
- Basic cryptographic primitives: public-key crypto, block cipher, stream cipher, hash function, MAC, digital signatures, zero-knowledge proofs
- Trusted entities
- Proofs of freshness, e.g., nonces and timestamps
  - NONCE = Number used only ONCE
  - Two types of nonces
    - Counter: unique (non-repeating) but predictable, may use a time stamp for this purpose
    - Random number: unique and unpredictable

“Ideal” Protocol Wish List

- Efficient protocol
  - Low computation overhead
  - Low communication overhead
- As little trust as necessary
- As few assumptions as necessary
  - Idealized encryption???
  - Synchronized clocks?
  - Synchronized sequence numbers?
  - Randomly selected nonces and IVs?
  - Security of crypto primitives?
  - Authenticity or secrecy of keys?
- Little client/server state

Protocol Analysis

- Analyze high level security properties
  - Secrecy
  - Authentication
  - Atomicity
  - Non-repudiation
- Assume cryptographic primitives secure
  - Signature: secure against existential forgery
  - Public key/Private key encryption: secure against adaptive chosen-ciphertext attack
- Security protocols are notoriously hard to get right

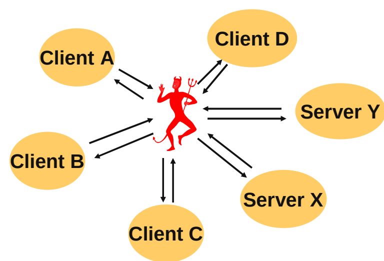
Active Attacker

- An active attacker may
  - Eavesdrop on previous protocol runs, even on protocol runs by other principals, replay messages at a later time
  - Inject messages into the network, e.g., fabricated from pieces of previous messages
  - Alter or delete a principal’s messages
  - Initiate multiple parallel protocol sessions
  - Run dictionary attack on passwords
  - Run exhaustive attack on low-entropy nonce

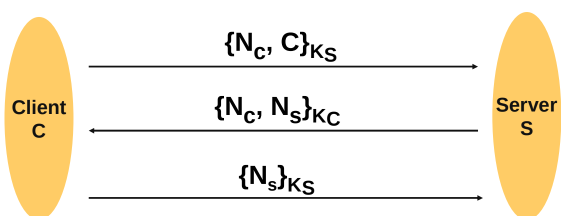
Intruder Model

Intruder can

- Intercept, drop, generate messages, full control of network
- Collude with malicious parties

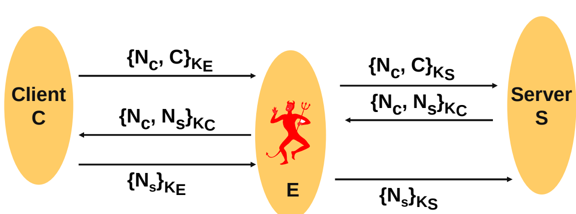


Example: Needham-Schroeder Protocol



- K_S, K_C are public keys of S and C respectively
- Goal:
  - Mutual authentication: C→S, S→C
  - Shared secret: N_C , N_S

Flaw in Needham-Schroeder



Flaw (discovered 18 years after publication):

- Authentication: C→E, S→C
- Secrecy: E knows N_C , N_S
- How to fix it?
  - The second message should be $\{S, N_C, N_S\}_{KC}$