Summary From the Last Lecture

- **Public key (asymmetric) cryptography**
  - Modular exponentiation for encryption/decryption
  - Efficient algorithms for this
  - Attacker needs to factor large numbers to extract key
  - Must use brute-force
- **One-way hash functions**
  - Collision-free, collision-resistant
  - MD5, SHA
- **DES, AES (not on exams)**

When to Encrypt/Hash?

- Confidentiality, integrity, non-repudiation
  - M, E(M), H(M), E(H(M)), H(E(M))
  - M + H(M) - integrity, H(M) must be stored not sent
  - M + E(H(M)) - integrity, non-repudiation (PK)
  - M + H(E(M)) - no sense
  - E(M) + H(M) - integrity, confidentiality
  - E(M) + E(H(M)) - integrity, confidentiality, non-repudiation (use different keys)
  - E(M) + H(E(M)) - integrity, confidentiality

Key Management

- Key management is where much security weakness lies
  - Choosing keys
  - Storing keys
  - Communicating keys

Key Exchange

Peer-to-Peer Key Distribution

- Technically easy
  - Distribute shared keys to each entity we want to communicate with
- But it doesn’t scale
  - Hundreds of servers...
  - Times thousands of users...
  - Yields ~ million keys

Key Exchange w Public Key Crypto

- Alice creates a secret key, encrypts it with Bob’s public key and sends it off
- Bob decrypts the message with his private key
- Use shared key for further communication
- This is how many applications work
- Could communicate using public key cryptography but it’s slow
**Diffie–Hellman Key Exchange**
- Exchange a secret with someone you never met while shouting in a room full of people
- Alice and Bob agree on $g$ and large $n$
- Alice chooses random $a$, sends $g^a \mod n$
- Bob chooses random $b$, sends $g^b \mod n$
- Alice takes Bob’s message and calculates $g^{ab} \mod n$
- Bob does the same; now they both know shared secret $g^{ab} \mod n$

**Man-in-the-Middle Attack On Key Exchange**
- Alice sends 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 sends 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 $E_k(K)$
- She sends both the message encrypted with $K$, $E_k(M)$ and a key $E_k(K)$ to Bob
- This is how most real-world protocols work

**Incremental Key Distribution**
- Step toward Needham–Schroeder and Kerberos mechanisms
- Key-distribution tied to authentication
  - If you know who you share a key with, authentication is easy – they just send you something encrypted with that key (must be something you’ve chosen)

**Encryption Based Authentication**
- Proving knowledge of shared key
  - Nonce = Non repeating, random value
  - But where does $K_{AB}$ come from?
KDC Based Key Distribution
- **KDC = Key Distribution Center**
  - Everyone shares a key with KDC, e.g., C shares $K_C$
  - User C sends request to KDC that they want to communicate with the server S – need $K_{CS}$
- KDC generates a key: $K_{cs}$
  - Encrypted for each participant: $K_C(K_{cs})$, $K_S(K_{cs})$
  - $K_C(K_{cs})$ called ticket
  - Ticket plus $K_{cs}$ called credentials
  - Ticket is opaque and forwarded with application request
- No keys ever traverse net in the clear

Needham–Schroeder Key Exchange
- Third-party authentication service
  - Distributes session keys for authentication, confidentiality, and integrity
  - Problem: replay attack in step 3
  - Fix: use timestamps

Problem
- What happens if attacker does get session key $K_{CS}$?
  - Answer: Can reuse old session key to answer challenge-response, generate new requests, etc

Solution
- Replace (or supplement) nonce in request/reply with timestamp
  - $K_C(K_{CS}, S, N_C, t)$ and $K_S(K_{CS}, C, t)$ in steps 2, 3

Problems with Needham Shroeder
- Server has no guarantee that $K_{CS}$ is fresh
- If an attacker gets hold of $K_C$ key he can impersonate C to anyone
  - The only solution is for KDC to tell everyone that $K_C$ was revoked
- Protocol assumes all users of it are good guys

Kerberos
- Introduce Ticket Granting Server (TGS)
  - Issues timed keys to resources
  - Users log on to authentication server (AS)
  - $AS + TGS = KDC$
  - Uses timestamps with a lifetime instead of nonces
  - Fixes freshness problem from Needham–Schroeder
First Report Due in Two Weeks

- Chosen paper must talk about cryptography, authentication, authorization or policy
  - Select from venues listed on the class Web page
  - Email me your chosen paper to verify it fits the topic
- Write 2-4 page report
  - Summary of problem, why is it important and hard, solution summary, evaluation and results, your opinion and your ideas
  - Originality, clearness, writing style
  - Proof-read!!
- Start now!