Announcements

• Project 3 available on the web.
  – Get the handout in class today.
  – Project 3 is due April 4th
  – It is easier than project 1 or 2, but *don't wait to start*

• Midterm 2 is *one week* from today
  – Tuesday: April 1st.
  – Will cover all material since the last midterm.
General Definition of “Protocol”

• A protocol is a multi-party algorithm
  – A sequence of steps that precisely specify the actions required of
    the parties in order to achieve a specified objective.

• Important that there are multiple participants
• Typically a situation of heterogeneous trust
  – Alice may not trust Bart
  – Bart may not trust the network
Characteristics of Protocols

• Every participant must know the protocol and the steps in advance.
• Every participant must agree to follow the protocol
  – Honest participants

• Big problem: How to deal with bad participants?
Cryptographic Protocols

- Consider communication over a network…
- What is the threat model?
  - What are the vulnerabilities?
What Can the Attacker Do?

- Intercept them (confidentiality)
- Modify them (integrity)
- Fabricate other messages (integrity)
- Replay them (integrity)
- Block the messages (availability)
- Delay the messages (availability)
- Cut the wire (availability)
- Flood the network (availability)
Dolev-Yao Model

- Simplifies reasoning about protocols
  - doesn't require reduction to computational complexity
- Treat cryptographic operations as "black box"
- Given a message $M = (c_1, c_2, c_3, \ldots)$ attacker can deconstruct message into components $c_1$ $c_2$ $c_3$
- Given a collection of components $c_1$, $c_2$, $c_3$, ... attacker can forge message using a subset of the components $(c_1, c_2, c_3)$
- Given an encrypted object $K\{c\}$, attacker can learn $c$ only if attacker knows decryption key corresponding to $K$
- Attacker can encrypt components by using:
  - fresh keys, or
  - keys they have learned during the attack
Formal Dolev-Yao Model

- A message is a finite sequence of:
  - Atomic strings, nonces, Keys (public or private), Encrypted Submessages
  
  \[ M ::= a \mid n \mid K \mid k \mid K\{M\} \mid k\{M\} \mid M, M \]

- The attacker's (or observer's) state is a set \( S \) of messages:
  - The set of all message & message components that the attacker has seen -- the attacker's "knowledge"
  - Seeing a new message sent by an honest participant adds the new message components to the attacker's knowledge
  - If \( M_1, M_2 \in S \) then \( M_1 \in S \) and \( M_2 \in S \)
  - If \( K_A\{M\} \in S \) and \( K_A \in S \) then \( M \in S \)
  - If \( K_A\{M\} \in S \) and \( k_A \in S \) then \( M \in S \)
  - If \( M \in S \) and \( K \in S \) then \( K\{M\} \in S \)
  - If \( M \in S \) and \( k \in S \) then \( k\{M\} \in S \)
  
  \( S \) closed under these operations
Using the Dolev-Yao model

• Given a description of a protocol:
  – Sequence of messages to be exchanged among honest parties.

• "Simulate" an attacked version of the protocol:
  – At each step, the attacker's knowledge state is the (closure of the) knowledge of the prior state plus the new message
  – An active attacker can create (and insert into the communication stream) any message M composed from the knowledge state S:
    • $M = M_1, M_2, \ldots, M_n$ such that $M_i \in S$

• See if the "attacked" protocol leads to any bad state
  – Example: if K is supposed to be kept secret but $K \in S$ at some point, the attacker has learned the key.
Authentication

• For honest parties, the claimant A is able to authenticate itself to the verifier B. That is, B will complete the protocol having accepted A’s identity.
Shared-Key Authentication

- Assume Alice & Bart already share a key $K_{AB}$.
  - The key might have been decided upon in person or obtained from a trusted 3rd party.
- Alice & Bart now want to communicate over a network, but first wish to authenticate to each other
Solution 1: Weak Authentication

- Alice sends Bart $K_{AB}$.
  - $K_{AB}$ acts as a password.
- The secret (key) is revealed to passive observers.
- Only works one-way.
  - Alice doesn’t know she’s talking to Bart.
Solution 2: Strong Authentication

- Protocol doesn’t reveal the secret.
- **Challenge/Response**
  - Bart requests proof that Alice knows the secret
  - Alice requires proof from Bart
  - $R_A$ and $R_B$ are randomly generated numbers
(Flawed) Optimized Version

- Why not send more information in each message?
- This seems like a simple optimization.
- But, it’s broken… how?
Attack: Marvin can Masquerade as Alice

- Marvin pretends to take the role of Alice in two runs of the protocol.
  - Tricks Bart into doing Alice’s part of the challenge!
  - Interleaves two instances of the same protocol.
Lessons

• Protocol design is tricky and subtle
  – “Optimizations” aren’t necessarily good

• Need to worry about:
  – Multiple instances of the same protocol running in parallel
  – Intruders that play by the rules, mostly

• General principle:
  – Don’t do anything more than necessary until confidence is built.
  – Initiator should prove identity before responder takes action (like encryption)
Threats

• **Transferability**: B cannot reuse an identification exchange with A to successfully impersonate A to a third party C.

• **Impersonation**: The probability is negligible that a party C distinct from A can carry out the protocol in the role of A and cause B to accept it as having A’s identity.
Assumptions

• A large number of previous authentications between A and B may have been observed.

• The adversary C has participated in previous protocol executions with A and/or B.

• Multiple instances of the protocol, possibly instantiated by C, may be run simultaneously.
Primary Attacks

• Replay.
  – Reusing messages (or parts of messages) inappropriately

• Interleaving.
  – Mixing messages from different runs of the protocol.

• Reflection.
  – Sending a message intended for destination A to B instead.

• Chosen plaintext.
  – Choosing the data to be encrypted

• Forced delay.
  – Denial of service attack -- taking a long time to respond
    – Not captured by Dolev Yao model
Primary Controls

• Replay:
  – use of challenge-response techniques
  – embed target identity in response.

• Interleaving
  – link messages in a session with chained nonces.

• Reflection:
  – embed identifier of target party in challenge response
  – use asymmetric message formats
  – use asymmetric keys.
• Chosen text:
  – embed self-chosen random numbers (“confounders”) in responses
  – use “zero knowledge” techniques.

• Forced delays:
  – use nonces with short timeouts
  – use timestamps in addition to other techniques.
Replay

- *Replay*: the threat in which a transmission is observed by an eavesdropper who subsequently reuses it as part of a protocol, possibly to impersonate the original sender.
  - Example: Monitor the first part of a telnet session to obtain a sequence of transmissions sufficient to get a log-in.

- Three strategies for defeating replay attacks
  - Nonces
  - Timestamps
  - Sequence numbers.
Nonces: Random Numbers

• **Nonce**: A number chosen at random from a range of possible values.
  – Each generated nonce is valid only once.

• In a challenge-response protocol nonces are used as follows.
  – The verifier chooses a (new) random number and provides it to the claimant.
  – The claimant performs an operation on it showing knowledge of a secret.
  – This information is bound inseparably to the random number and returned to the verifier for examination.
  – A timeout period is used to ensure “freshness”.


Time Stamps

- The claimant sends a message with a timestamp.
- The verifier checks that it falls within an acceptance window of time.
- The last timestamp received is held, and identification requests with older timestamps are ignored.
- Good only if clock synchronization is close enough for acceptance window.
Sequence Numbers

- Sequence numbers provide a sequential or monotonic counter on messages.
- If a message is replayed and the original message was received, the replay will have an old or too-small sequence number and be discarded.
- Cannot detect forced delay.
- Difficult to maintain when there are system failures.
Unilateral Symmetric Key

- Unilateral = one way authentication
- Unilateral authentication with nonce.
Mutual Symmetric Key

- Mutual = two way authentication
- Using Nonces:

\[ K_{AB}\{n_A, n_B, B\} \]

\[ K_{AB}\{n_A, A\} \]
Mutual Public Key Decryption

- Exchange nonces

\[ K_B \{ n_A, A \} \]

\[ K_A \{ n_A, n_B \} \]

\[ n_B \]
Usurpation Attacks

• Identification protocols corroborate the identity of an entity only at a given instant in time.
  – An attacker could "hijack" a session after authentication.

• Techniques to assure ongoing authenticity:
  – Periodic re-identification.
  – Tying identification to an ongoing integrity service. For example: key establishment and encryption.
General Principles

- Don’t do anything more than necessary until confidence is built.
  - Initiator should prove identity before the responder does any “expensive” action (like encryption)
- Embed the intended recipient of the message in the message itself
- Principal that generates a nonce is the one that verifies it
- Before encrypting an untrusted message, add “salt” (i.e. a nonce) to prevent chosen plaintext attacks
- Use asymmetric message formats (either in “shape” or by using asymmetric keys) to make it harder for roles to be switched
Arbitrated Protocols

- Tom is an *arbiter*
  - Disinterested in the outcome (doesn’t play favorites)
  - Trusted by the participants (Trusted 3rd party)
  - Protocol can’t continue without T’s participation
Arbitrated Protocols (Continued)

• Real-world examples:
  – Lawyers, Bankers, Notary Public

• Issues:
  – Finding a trusted 3rd party
  – Additional resources needed for the arbitrator
  – Delay (introduced by arbitration)
  – Arbitrator might become a bottleneck
  – Single point of vulnerability: attack the arbitrator!
Adjudicated Protocols

- Alice and Bard record an *audit log*
- Only in exceptional circumstances do they contact a trusted 3rd party. (3rd party is not always needed.)
- Tom as the *adjudicator* can inspect the evidence and determine whether the protocol was carried out fairly
Self-Enforcing Protocols

- No trusted 3rd party involved.
- Participants can determine whether other parties cheat.
- Protocol is constructed so that there are no possible disputes of the outcome.

Alice

You’re cheating, Alice!

Bart