CSE 3400/5850 - Introduction to Computer & Network Security / Introduction to Cybersecurity

Lecture 8

Shared Key Protocols – Part I

Ghada Almashaqbeh

UConn

Adapted from the Textbook Slides

Outline

- Cryptography protocols.
- Session or record protocols.
- Entity authentication protocols.

Modeling Cryptographic Protocols

- A protocol is a set of PPT (efficient) functions or algorithms
  - Each receiving (state, input), outputting (state, output)
  - Two (or more) parties, each has its own state
- Including *Init, In,* [and if needed *Wakeup*] functions
  - And task-specific functions, e.g., *Send*
- The execution process is a series of function invocations based on which the protocol proceeds.
- Our discussion (from here) is focused on shared-key, two-party protocols, MitM adversary.

Record Protocols

Secure communication between two parties using shared keys.

Two-party, shared-key **Record protocol**

- Parties/peers: *Alice* (sender), *Bob* (receiver)
  - Simplest – yet applied – protocol
  - Simplify: only-authentication for what Alice sends to Bob
    - Goal: Bob outputs *m* only if Alice had *Send(m)*

- Let's design the protocol! define the protocol functions
  - $Init(k)$ [Initialize Alice/Bob with secret key *k*]
  - $Send(m)$*:* Alice sends message *m and* a tag over *m* (to Bob)
  - $In(m)$: Bob receives (*m, tag) and accepts m* is the *tag* is valid.

Two-party, shared-key **Record protocol**

- Design has many simplifications, easily avoided:
  - Only message authentication
    - No confidentiality!
  - Only ensure same message was sent
    - Does not address duplication, out-of-order, `stale' messages, losses
- To add confidentiality: use encryption
  - Namely, employ EtA (encrypt then authenticate).

Two-party record protocol with Confidentiality

- $Init(k)$ [Initialize Alice/Bob with secret key *k*]
  - $\{s \leftarrow (k_E = F_k(`E`), k_A = F_k(`A`))$
- $Send(m)$: Alice sends message *m* (to Bob)
  - $\{Output\ x = (E_{k_E}(m), MAC_{k_A}(E_{k_E}(m)))\ ;\ \}$
- $In((c, tag))$: Bob receives (c, tag) from adversary
  - $\{Output\ D_k(c)$ if $(tag = MAC_{k_A}(c))\ ;\}$

So, security guarantees …

What does a secure shared-key two-party record protocol mean?

How about the security of the one with confidentially?

Shared-key Entity Authentication Protocols

Ensure the identity of an entity (or a peer) involved in communication.

Mutual Authentication Protocols

- Our focus.
- In mutual authentication, each party authenticates herself to the other.
  - Alice knows that she is communicating with Bob, and vice versa
- This requires, at least, one exchange of messages.
  - A message from Alice and a response from Bob (or vice versa).
- Such a flow is called a ***handshake***.

Handshake Entity-Authentication protocol

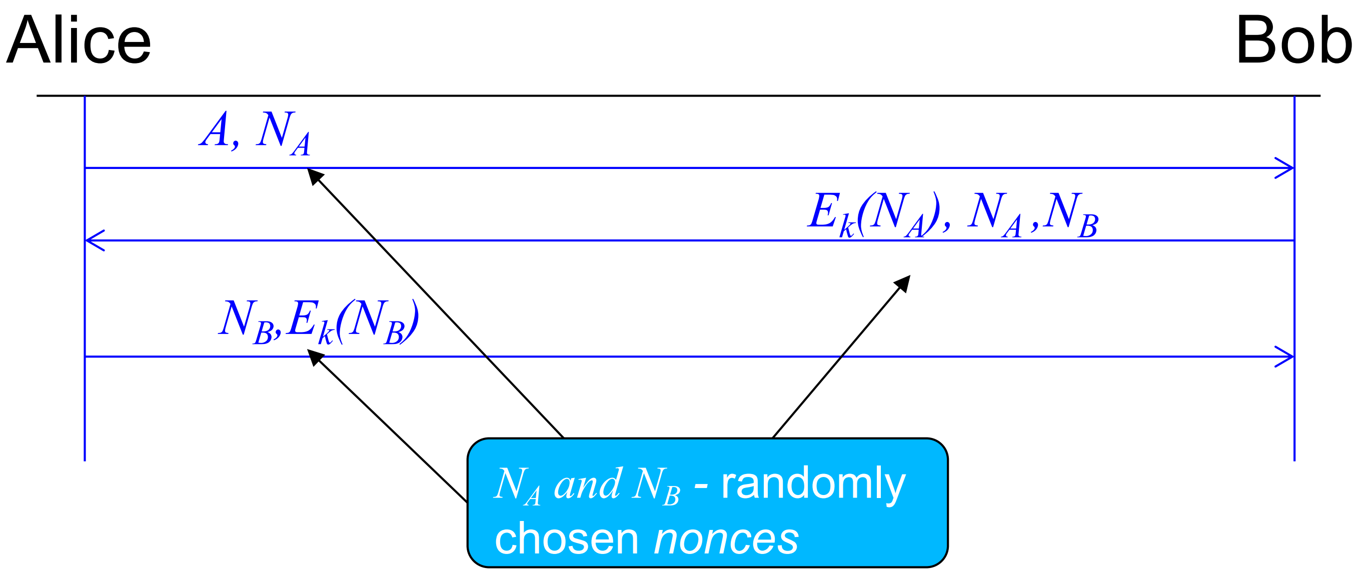
- A protocol to open **sessions** between parties
  - Each party assigns its own unique ID to each session, and maps peer's-IDs to its own IDs
    - Alice maps Bob's i_B to its identifier $ID_A(i_B)$
    - Bob maps Alice's i_A to its identifier $ID_B(i_A)$
- 'Matching' goal: $i_A = ID_A\big(ID_B(i_A)\big)\,, i_B = ID_B\big(ID_A(i_B)\big)$
- Allow concurrent sessions and both to open
  - Simplify: no timeout / failures / close, ignore session protocol, …

Handshake Entity-Authentication protocol

- Protocol functions
  - $Init(k)$: Initialize Alice/Bob with secret key *k*
  - $Open$: Alice/Bob open a session
  - $Out(x)$: party sends x to peer
  - $In(x)$: party receives x from channel (via MitM)
- Protocol outputs
  - $Open(i)$: party opened session i
  - (and received messages).

Example : IBM's SNA Handshake

- First dominant networking technology
- Handshake uses encryption with shared key k

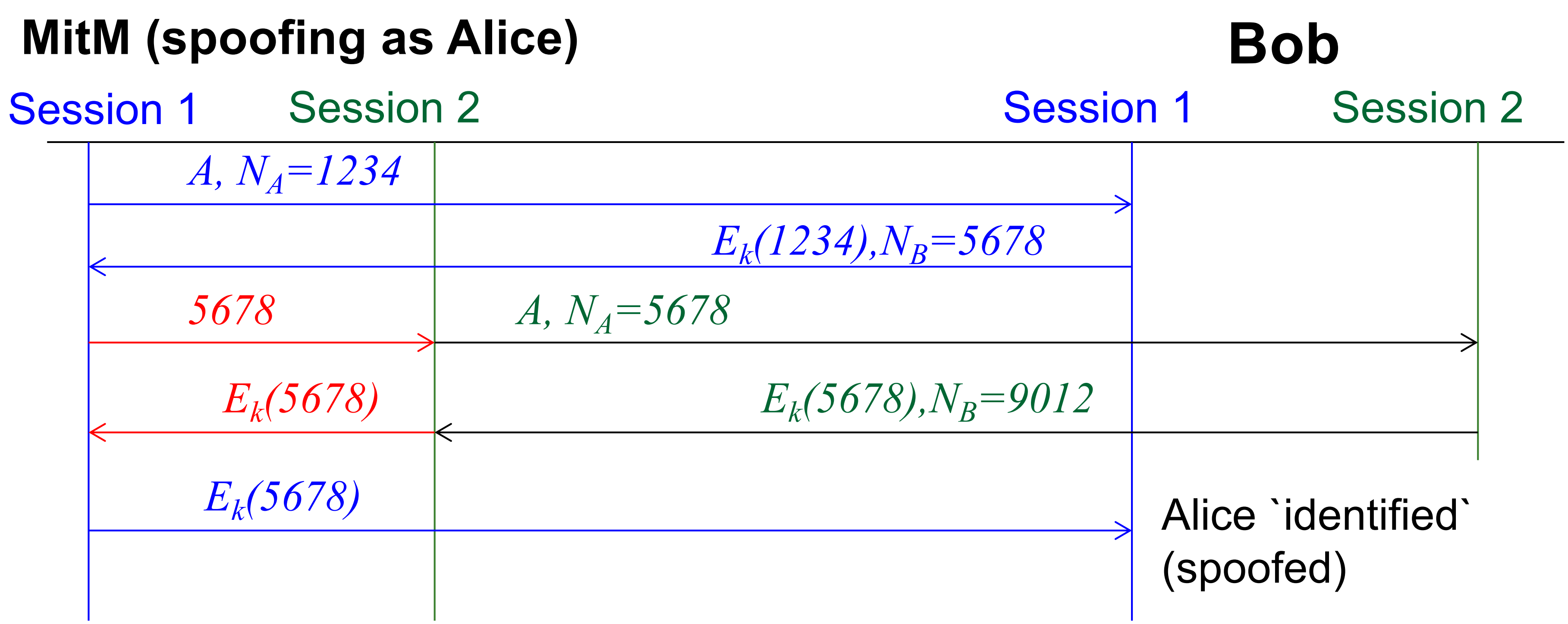


Insecure !! Why ?

SNA (Systems Network Architecture): IBM's proprietary network architecture, dominated market @ [1975-1990s], mainly in banking, government.

Attack on SNA's Handshake

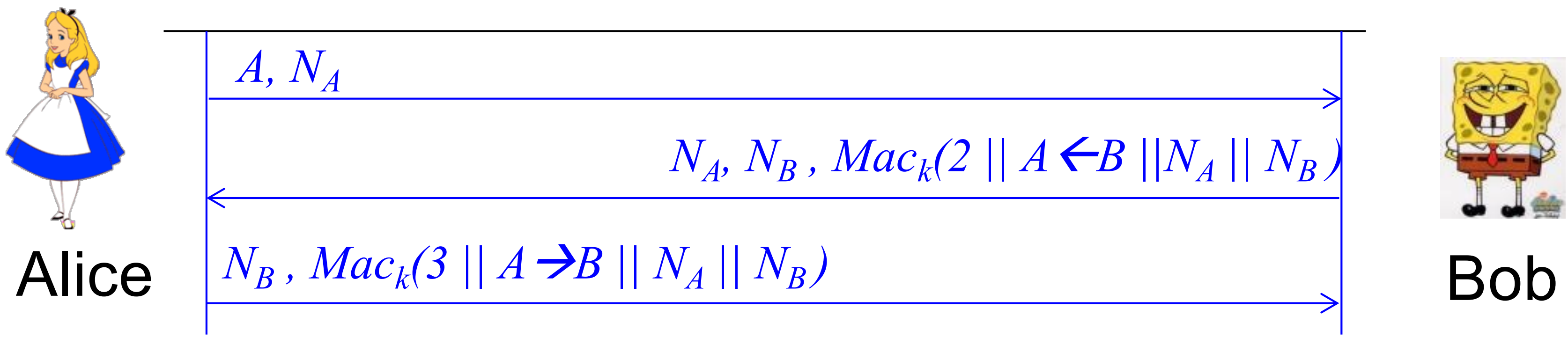
- MitM opens two sessions with Bob, sending N_B to Bob in 2nd connection to get $E_k(N_B)$
  - SNA is secure for sequential mutual authentication handshakes but not concurrent ones.



Fixing Mutual Authentication

- Encryption does not ensure authenticity
  - Use MAC to authenticate messages!
- Prevent redirection
  - Identify party in challenge
  - Better: use separate keys for each direction
- Prevent replay and reorder
  - Identify flow and connection
  - Prevent use of old challenge: randomness, time or state
- Do not provide the adversary with an oracle access!
  - Do not compute values from Adversary
  - Include self-chosen nonce in the protected reply

Secure Two-Party Handshake Protocol (2PP)



- ✓ Use MAC rather than encryption to authenticate
- ✓ Prevent redirection: include identities (A,B)
- ✓ Prevent replay and reorder:
  - Nonces (N_A,N_B)
  - Separate 2^{nd} and 3^{rd} flows: 3 vs. 2 input blocks
- Provably secure [formal proof is out of scope]

Covered Material From the Textbook

- Chapter 5
  - Sections 5.1 and 5.2

Thank You!