
CSE 3400 - Introduction to Computer & Network Security
(aka: Introduction to Cybersecurity)

Lecture 8

Shared Key Protocols – Part I

Ghada Almashaqbeh

UConn

From Textbook Slides by Prof. Amir Herzberg

UConn

Outline

- ❑ Modeling cryptography protocols.
- ❑ Session or record protocols.
- ❑ Entity authentication protocols.

Modeling Cryptographic Protocols

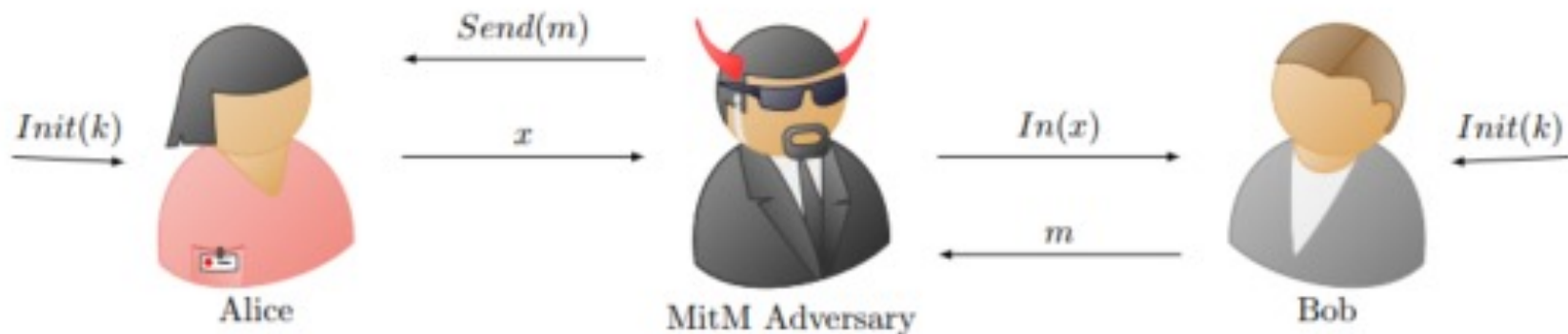
- ❑ A protocol is a set of PPT (efficient) functions
 - ❑ Each receiving (state, input), outputting (state, output)
 - ❑ Two (or more) parties, each has its own state
- ❑ Including *Init*, *In*, [and if needed *Wakeup*] functions
 - ❑ And task-specific functions, e.g., *Send*
- ❑ Adversary can invoke any function, handle outputs
- ❑ The execution process is a series of function invocations based on which the protocol proceeds.
- ❑ Our discussion (from here) is mostly informal
 - ❑ Definitions of protocols, execution, goals are hard
 - ❑ Focus 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, Alice sends to Bob
 - ❑ Goal: Bob outputs m only if Alice had $\text{Send}(m)$
 - ❑ $\text{Init}(k)$: shared key, unknown to adversary

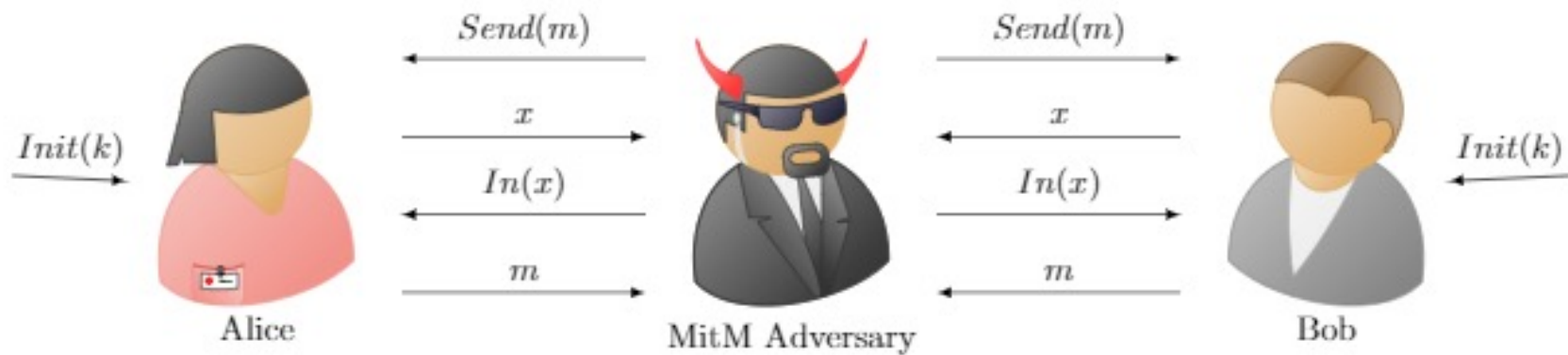


- ❑ Let's design the protocol !

Design of Two-party, shared-key Record protocol

- ❑ Design: define the protocol functions
 - ❑ $Init(k)$ [Initialize Alice/Bob with secret key k]
 - ❑ $\{s.k \leftarrow k; \}$
 - ❑ Save received key k in state-variable $s.k$ (part of s)
 - ❑ $Send(m)$: party asked to send m to peer
 - ❑ Code even simpler if both can send, receive
 - ❑ E.g., Alice instructed to send message m to Bob
 - ❑ $\{Output\ x \leftarrow (m, MAC_k(m)); \}$
 - ❑ $In((m, \sigma))$: Party receives (m, σ) from adversary
 - ❑ $\{Output\ m\ if\ (\sigma = MAC_k(m)); \}$
 - ❑ Output the message only if validated Ok
- ❑ Define adversary capabilities; access and computational.

Design of Two-party, shared-key Record protocol



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

- ❑ Design has many simplifications, easily avoided:
 - ❑ Only message authentication
 - ❑ No confidentiality!
 - ❑ Only ensure same message was sent
 - ❑ Allow duplication, out-of-order, 'stale' messages, losses
 - ❑ Also: no retransmissions, compression, ...
- ❑ To add confidentiality: use encryption

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)
 - ❑ $\{Output\ x = (E_{k_E}(m), MAC_{k_A}(E_{k_E}(m))) ; \}$
- ❑ $In((c, \sigma))$: Bob receives (c, σ) from adversary
 - ❑ $\{Output\ D_k(c)$ if $(\sigma = MAC_{k_A}(c)) ; \}$
- ❑ Ok! (but still allows dups/re-ordering, etc.)



Execution Process – Record Protocol

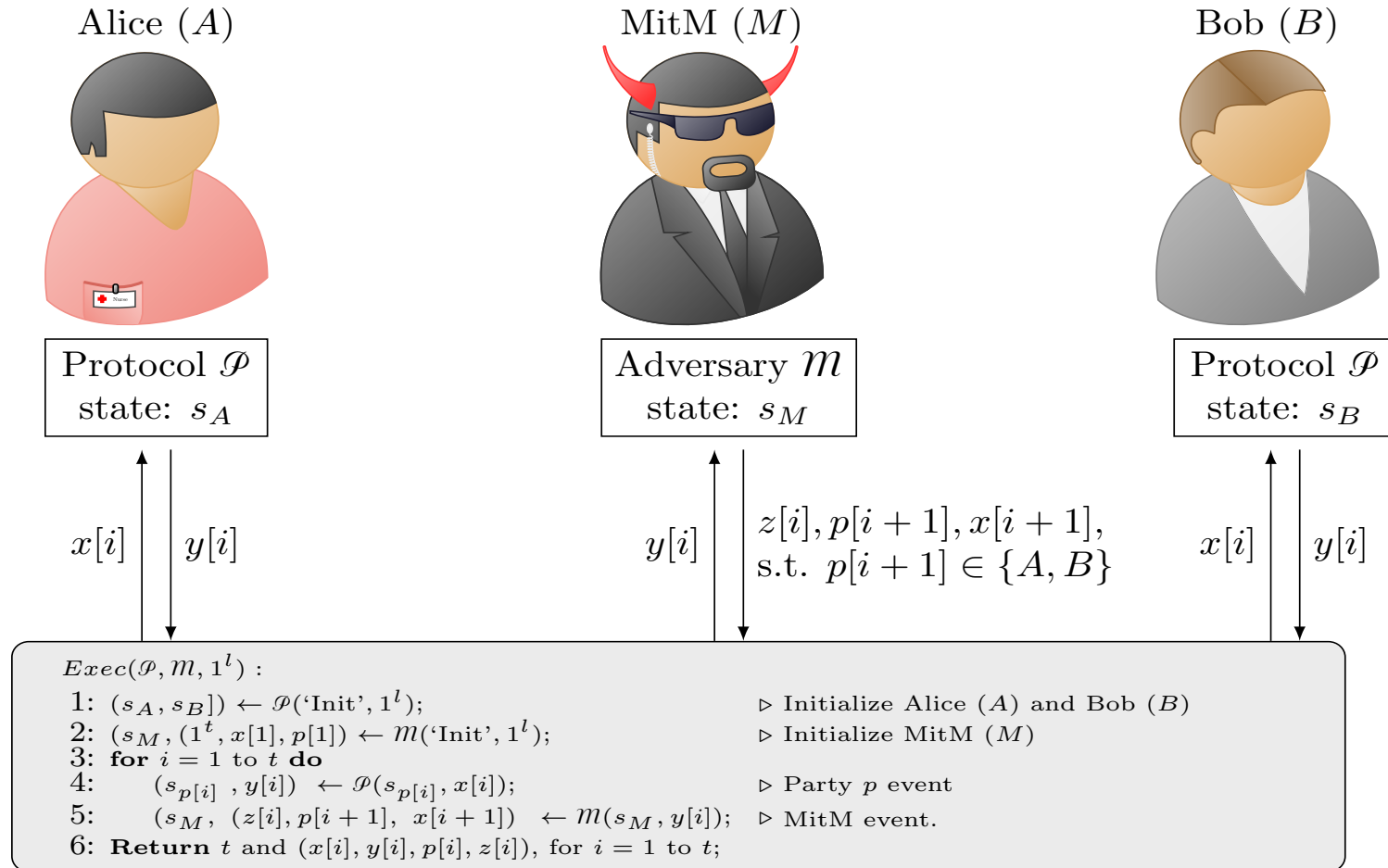
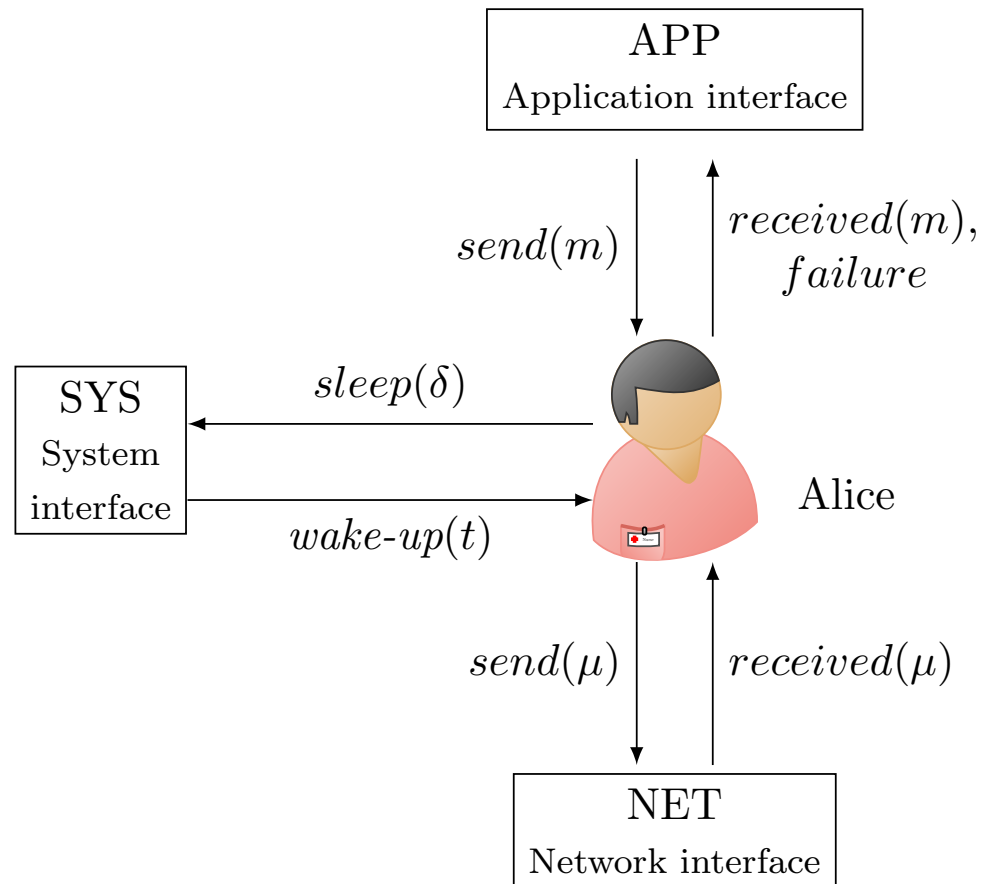


Figure 5.1: Execution process $Exec(\mathcal{P}, \mathcal{M}, 1^l)$ for two benign parties Alice (A) and Bob (B) running shared-key protocol (algorithm) \mathcal{P} , MitM adversary (M) running algorithm \mathcal{M} , and security parameter 1^l .

Labels and Interfaces



Bob has similar interfaces.

Defining Security of Record Protocols

The existential-unforgeability advantage $\varepsilon^{EUF-Session}(\mathcal{P}, \mathcal{M}, 1^l)$ of adversary \mathcal{M} against session/record protocol \mathcal{P} is defined as:

$$\begin{aligned} \varepsilon^{EUF-Session}(\mathcal{P}, \mathcal{M}, 1^l) &\equiv \\ &\equiv \Pr \left(\begin{array}{l} T \xleftarrow{\$} \text{Exec}(\mathcal{P}, \mathcal{M}, 1^l); \\ M^{rcv}(T) \text{ is not a prefix of } M^{sent}(T) \end{array} \right) \end{aligned} \quad (5.1)$$

Where the probability is taken over the random coin tosses of \mathcal{M} and \mathcal{P} during the execution resulting in transcript T , and where $M^{sent}(T)$, $M^{rcv}(T)$ are defined as above.

A session/record protocol \mathcal{P} is existentially unforgeable if for all PPT algorithms \mathcal{M} , the advantage of \mathcal{M} against \mathcal{P} is negligible, i.e.: $\varepsilon^{EUF-Session}(\mathcal{P}, \mathcal{M}, 1^l) \in NEGL(l)$.

Entity Authentication Protocols

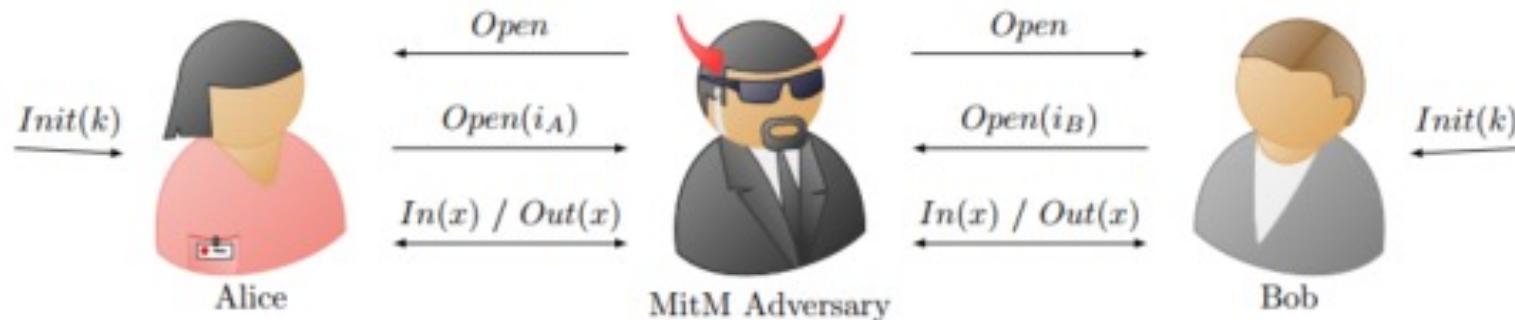
Ensure the identity of an entity (or a peer) involved in communication.

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 map 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: $i_A = ID_A(ID_B(i_A))$, $i_B = ID_B(ID_A(i_B))$
- ❑ Allow concurrent sessions and both to open
 - ❑ Simplify: no timeout / failures / close, ignore session protocol, ...



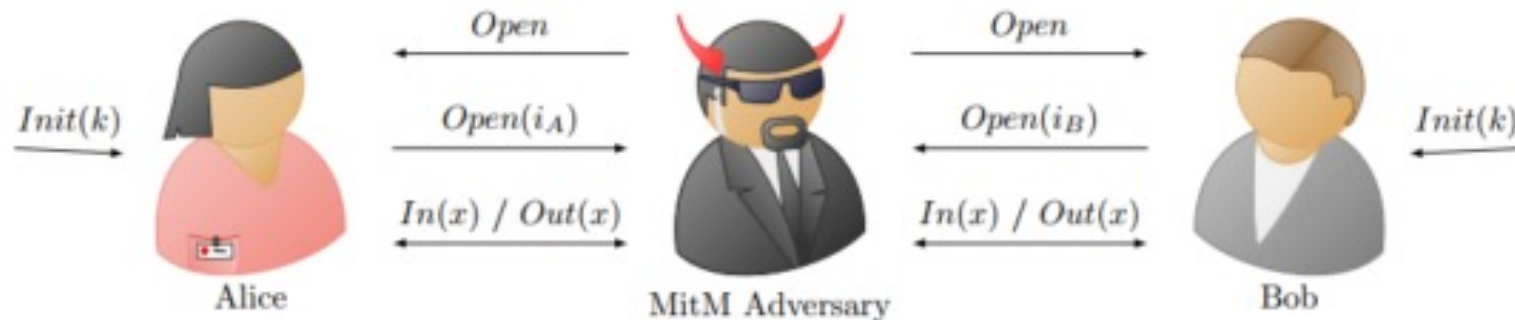
Handshake Entity-Authentication protocol

□ Protocol functions

- $Init(k)$: Initialize Alice/Bob with secret key k
- $Open$: instruct Alice/Bob to open session
- $In(x)$: party receives x from channel (via MitM)

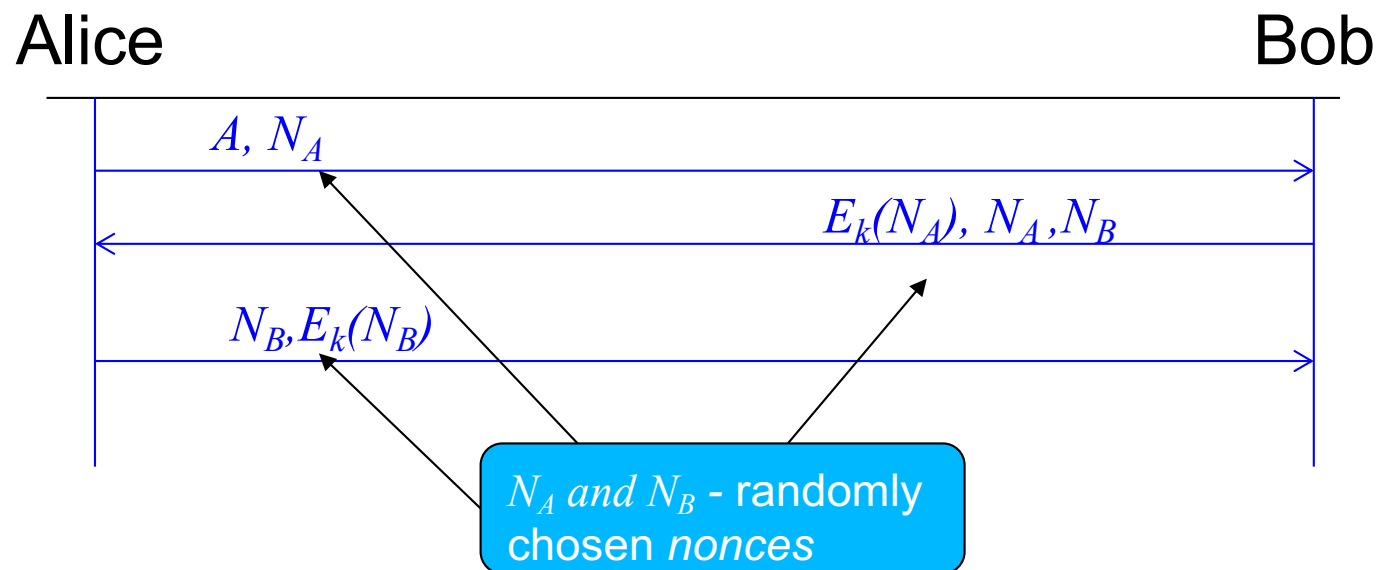
□ Protocol outputs

- $Open(i)$: party opened session i
- $Out(x)$: party asks to send x to peer



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.

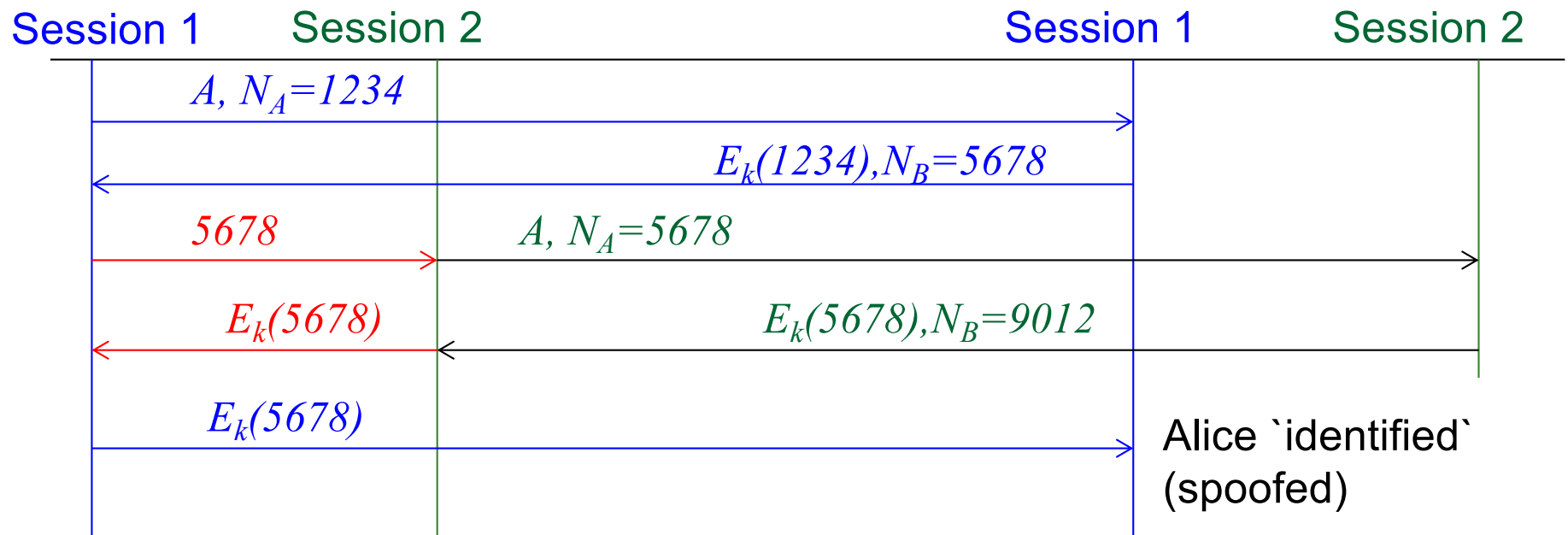
Attack on SNA's Handshake

❑ **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.

MitM (spoofing as Alice)

Bob



Fixing Mutual Authentication

- Encryption does not ensure authenticity
 - Use MAC to authenticate messages
 - Although, a block cipher is a PRP, and a PRP is a PRF, and a PRF is a MAC, but domain is limited!
- 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

Two-Party Handshake Protocol (2PP)



Alice

A, N_A

$N_B, Mac_k(2 \parallel A \leftarrow B \parallel N_A \parallel N_B)$

$Mac_k(3 \parallel A \rightarrow B \parallel N_A \parallel N_B)$



Bob

- ✓ Use MAC rather than encryption to authenticate
- ✓ Prevent redirection: include identities (A, B)
- ✓ Prevent replay and reorder:
 - Nonces (N_A, N_B)
 - Separate 2nd and 3rd flows: 3 vs. 2 input blocks
 - Secure against arbitrary attacks [proved formally in the literature]

Covered Material From the Textbook

- ❑ Chapter 5
 - ❑ Sections 5.1 and 5.2

Thank You!

