CS5285
Information Security for eCommerce

Dr. Gerhard Hancke

CS Department
City University of Hong Kong

Reminder of previous lecture

- Integrity (data origin authentication/non-repudiation)
  - Hash (known algorithm, no key)
    - Only detect accidental modification
    - Finding collisions should not be easy (birthday paradox >> n/2)
  - MAC (needs key)
    - Data origin authentication (only generated of key known)
    - HMAC or CBC-MAC
    - No non-repudiation (two parties can generate MAC)
  - Digital signature (asymmetric crypto system)
    - Sign with private key (only one person)
    - Verify with public key (anyone)
    - Look at RSA version (there are other signature schemes)
    - Only mechanism to provide non-repudiation (also origin auth.)
    - Only one person can generate signature

Today’s Lecture

- Authentication
  - We have spent several lecture discussing ‘tools’
  - Encryption, hash, MAC, digital signature
  - In the lecture we start using these…build protocols for entity authentication
- CILO3 and CILO4

  (assessment on the security analyze security measures)

Entity Authentication

Authentication

- Alice proves her identity to Bob
  - Alice and Bob can be humans or computers
- May also require Bob to prove that he is Bob (mutual authentication)
- E.g. Octopus cards, ATM machines

Authentication

- Authentication on a stand-alone computer with physically secure connection is relatively simple
- Authentication over a network is much more complex
  - Attacker can passively observe messages
  - Attacker can replay messages
  - Usually need an encrypted channel to do so securely

Authentication Example: Entry to Building

1. Insert badge into reader
2. Enter PIN
3. Correct PIN?

    Yes? Enter

    No? Get challenged by security guard

Authentication Example: ATM Machine Protocol

1. Insert ATM card
2. Enter PIN
3. Correct PIN?

    Yes? Conduct your transaction(s)

    No? Machine eats card

- Authentication between a prover and a verifier with physically secure connection is relatively simple.
- Authentication over an open network is more complex.

One-way authentication over an open network

There may be eavesdroppers on an open network.

Alice $\xrightarrow{\text{Username, password}}$ **Email Server**

An eavesdropper can steal Alice’s login information and then logon to the Email Server as Alice by using Alice’s login information (**masquerade attack**).

One-way authentication over an open network

How about

Alice $\xleftarrow{\text{PK, Cert}_{server}}$ Email Server

E_{PK}(Username, password) $\rightarrow$

Secure channel

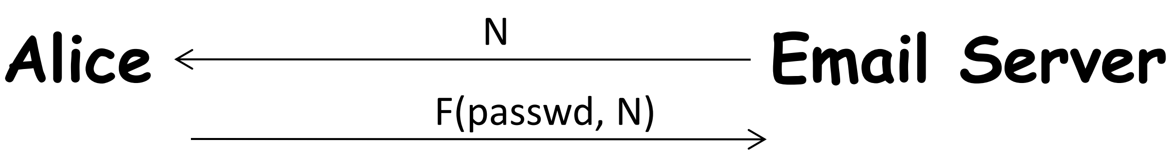
or

Alice $\xrightarrow{\text{Username, h(password)}}$ Email Server

Adversary simply replays E_{PK}(Username, password) or h(password) in the impersonation of Alice in the replay attack.

Challenge-Response One-Way Authentication

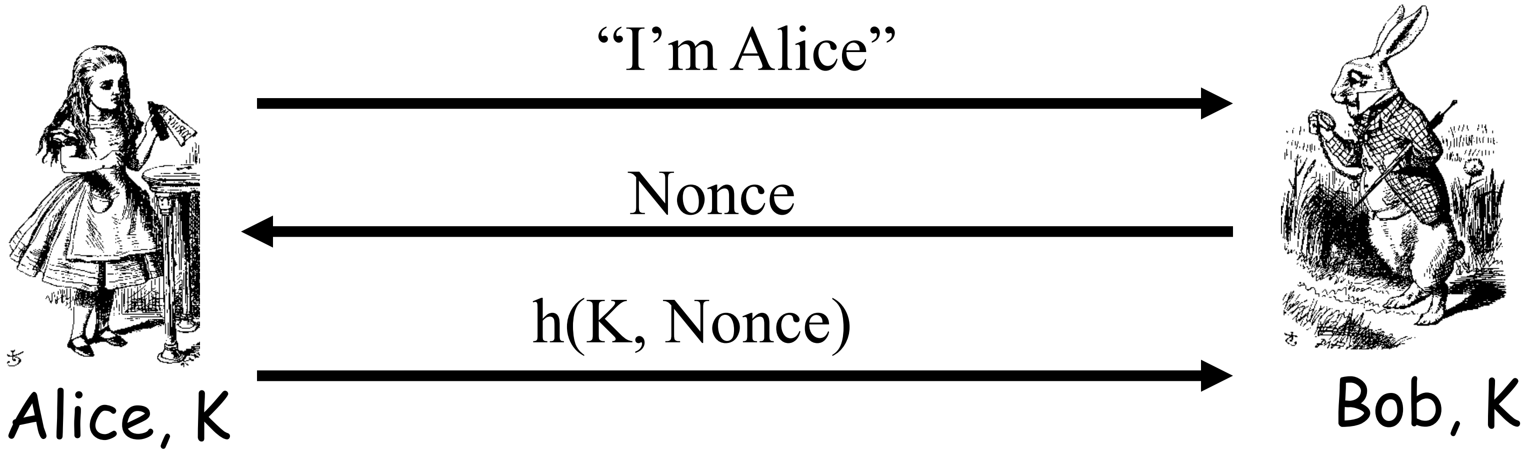
- To defend against replay attack
- Suppose Bob wants to authenticate Alice
  - Challenge sent from the verifier, Bob, to the prover, Alice
  - Only Alice should be able to provide the correct response



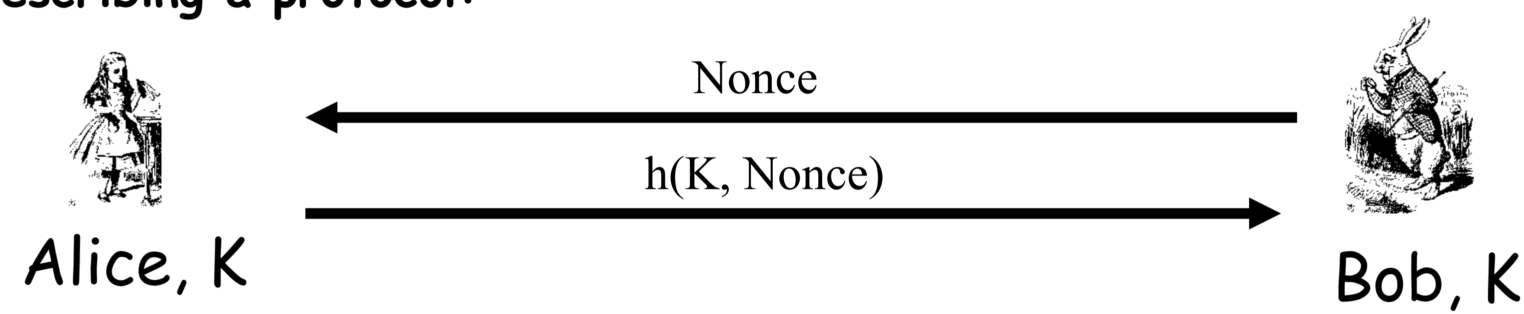
- **Challenge** N is a ***nonce*** (number used only once)
- N does not need to be a random number
- F(passwd, N) is the **response** where F is a one-way function and "passwd" is the password of Alice
- Examples of F: hash function, block cipher
- Only Alice and the Email Server know the value of passwd. Hence only Alice can provide the correct response to the Email Server.

Challenge-Response One-Way Authentication

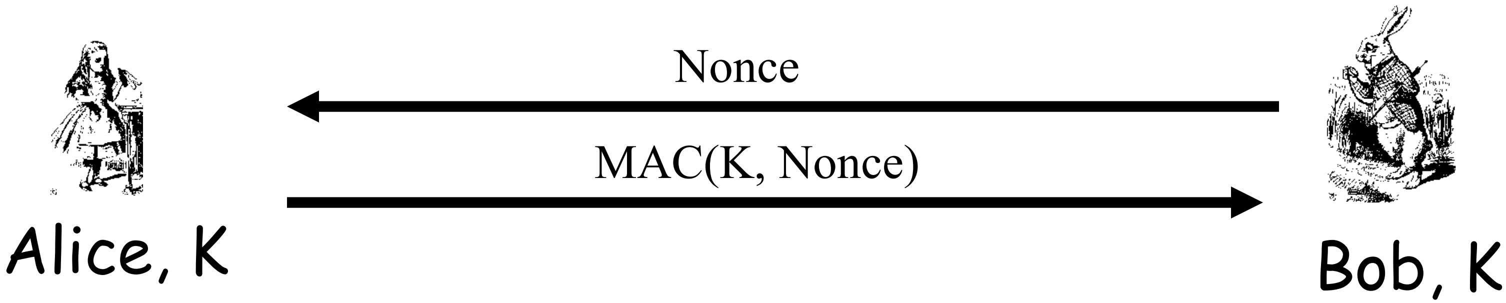
If Alice is a “device”, passwd can be changed to a symmetric key

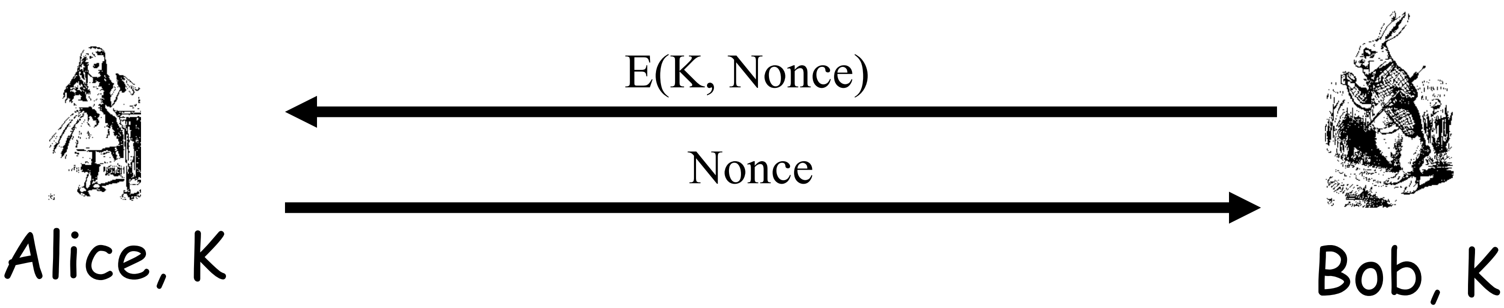


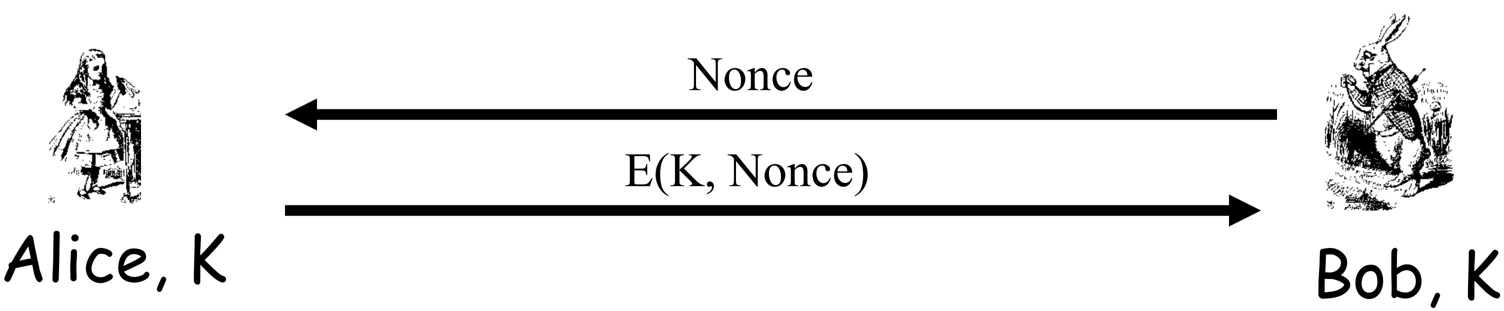
Usually, we ignore the first message flow from Alice to Bob when describing a protocol:



Other Challenge-Response Techniques (symmetric key based)



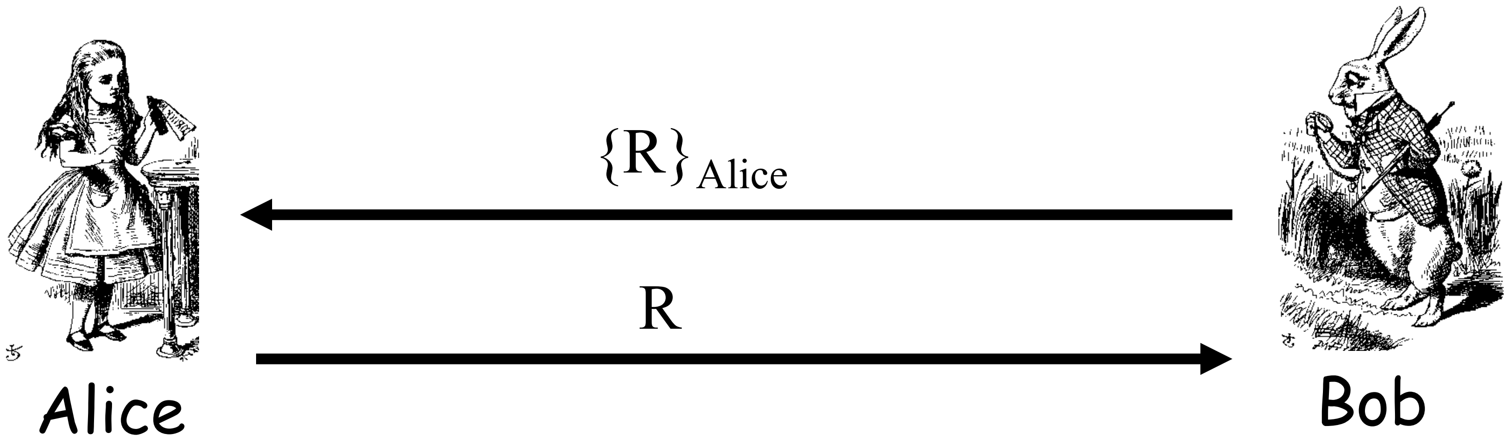


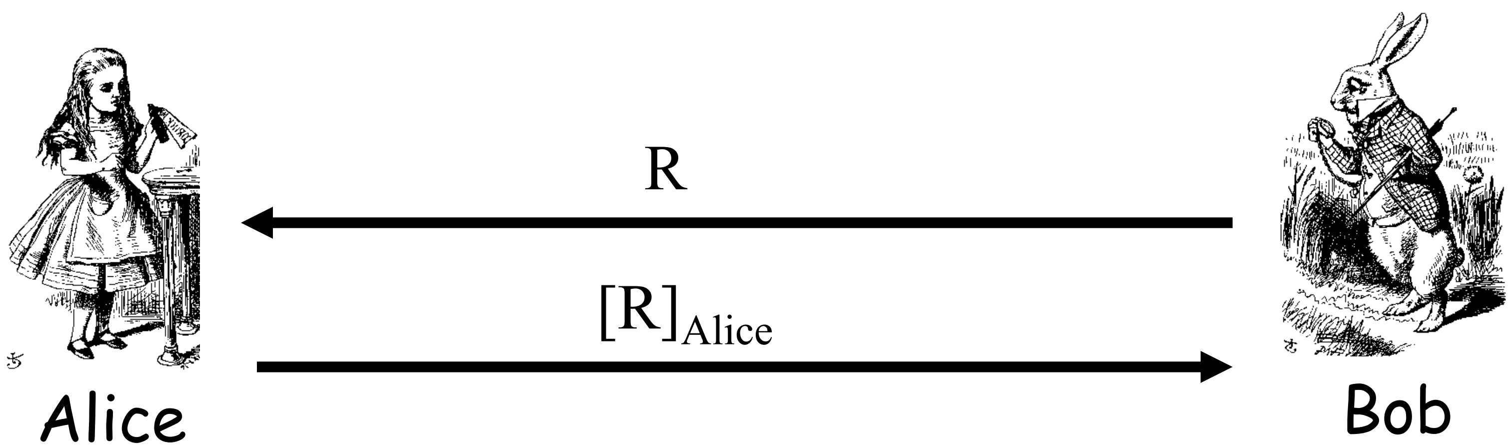


Public Key Notations and Assumption

- Encrypt M under Alice's public key: $\{\mathrm{M}\}_{\mathrm{Alice}}$
- Sign M with Alice's private key: $[\mathrm{M}]_{\mathrm{Alice}}$
- All public keys are assumed to be certified (e.g. digital certificates) and become publicly known.

Public Key Based One-Way Authentication





Entity authentication

- Some more formal definition:
  - Claimant: An entity claiming an identity.
  - Principal: The identity claimed by a claimant.
  - Verifier: The entity verifying a claim.
- An entity authentication protocol is a sequence of messages passed between a claimant and a verifier (along with the actions taken after receiving those messages) designed to confirm that identity of the claimant.

Entity authentication

Unilateral authentication:

- entity authentication which provides one entity with assurance of the other's identity but not vice versa.

- Mutual authentication:
  - entity authentication which provides both entities with assurance of each other's identity.

Entity authentication

- A verifier only sends/receives messages, i.e. digital data.
- To check that the principal is online the verifier need to establish:
  - that the messages came from the principal (origin authentication),
  - and that the messages have been recently generated (freshness).
- If both conditions are satisfied then we have authenticated the claimant.

Origin authentication

- We have already studied two mechanisms that give origin authentication:
  - Message Authentication Codes (MACs)
  - Digital Signature Schemes
- Entity authentication protocols sometimes find it useful to use symmetric encryption as an origin authentication tool…
- … but encryption doesn't provide origin authentication without additional features!

Origin authentication

- Encryption checks the integrity of a message by checking that it “makes sense”.
- It is hard for a computer to check whether a message makes sense or not.
- Append a *manipulation detection code* (MDC) to the message before encryption.
- A message “makes sense” if the MDC is correct for the decrypted message.
- What is MDC? Have we dealt with one before?
  - Could also be known or expected data in special case
  - A protocol could be design the receiver knows value

Freshness

- We have two methods to check that a message was recently generated (fresh).
  - Time stamps (both clock-based and “logical”)
  - Nonces or challenges (as in challenge-response protocols)
- Both involve computing some form of integrity protection for a unique string (the nonce or the time stamp).

Freshness

- The inclusion of a time stamp (stating the date and time the message was created) enables the recipient to check that it is fresh.
- Requires **securely** synchronised clocks.
- It is non-trivial to provide such clocks.
  - The clock drift of a typical workstation is about one second per day.
  - Initial synchronisation cannot use time stamps.

Freshness

- Clock synchronisation problems and network delays cause problems for time-stamp-based protocols.
- Necessary to accept time stamps within a “window of acceptance”.
- Necessary to store a log of received messages within the current window to avoid replays.

Freshness

- Logical time stamps (or sequence numbers) can be used in some protocols in place of "full" time stamps.
- Each entity maintains counters stating how many messages have been sent to and received from a particular entity.
- Let N_{AB} be the number of messages A has sent to B (both A and B should know this).

Freshness

- Whenever A sends a message to B, N_{AB} is increased by one and included in the message.
- When B receives a message that contains a counter value n:
  - If $n > N_{AB}$ then accept the message as fresh and reset $N_{AB} = n$.
  - If $n < N_{AB}$ then reject the message as not fresh.

Freshness

- The main alternative to the use of time stamps is the use of nonces.
  - *NONCE = Number used ONCE.*
- Alice generates a new random nonce and sends this to Bob…
- …and Bob includes this nonce in his reply.
- Therefore, Alice knows that Bob’s response is fresh.

Freshness

- Strictly speaking, a counter is a good way of producing nonces…
- … however, many protocols also require the nonce to be unpredictable to the attacker.
- Three main ways to produce random nonces:
  - Generate pseudo-randomly using a non-repeating generator
  - Generate at random and store a log
  - Generate at random and accept small chance of repeated nonce

Authentication attacks

- Many security properties of secure entity authentication protocols are defined by their resistance to certain kinds of attack.
- A masquerade attack is one in which the attacker directly generates messages that demonstrate that they are someone else.
  - Prevented by origin authentication mechanisms.

Authentication attacks

- A replay attack is one in which old messages are replayed to a verifier.
  - Prevented by freshness mechanisms.
- A reflection attack is one in which data the verifier has produced is sent back to him.
  - Prevented by including identifiers that show to whom a message is being sent.

How do we design/analyse protocols

- Recognise components of a cryptographic protocol
  - The protocol **assumptions**
    - *What needs to have happened **before** the protocol is run?*
  - The protocol **flow**
    - *Who sends a message to whom (in what order)?*
  - The protocol **messages**
    - *What information is exchanged at each step?*
  - The protocol **actions**
    - *What needs to be done between each step?*

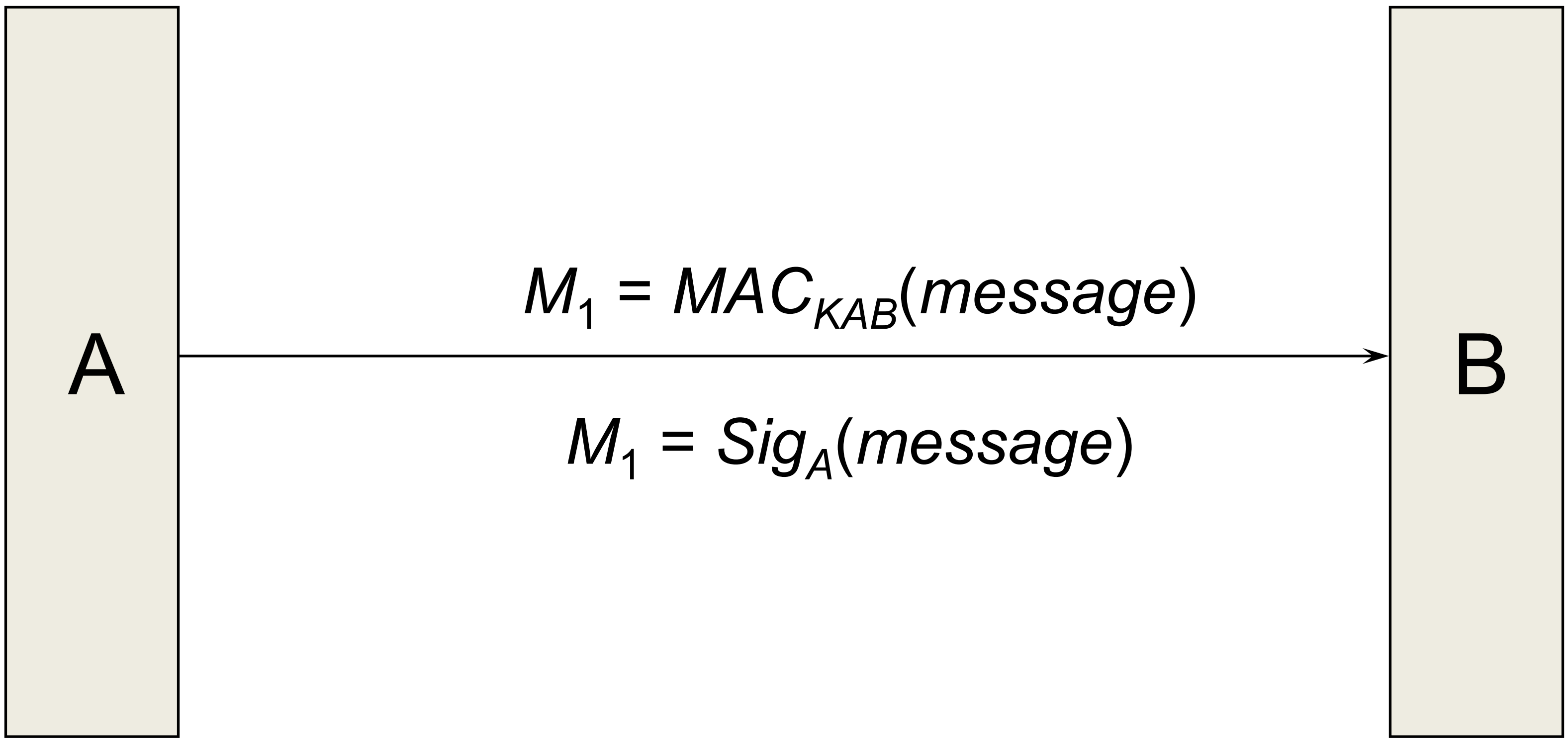
Stages of protocol design/analysis

- **What are the security objectives**
  - *What do you want to do?*
- **What are the protocol goals**
  - *Translating the security objectives into a set of cryptographic requirements to be met by the end of the protocol*
- **Define/analyse the protocol**
  - *Assumptions, flow, messages, actions*

Very simple example

- **Defining the security objectives**
  - *Bob wants to make sure that Alice was the source of a electronic purchase contract*
  - *Bob wants to the contract to be enforceable at later date (Alice should not deny it)*
- **Determining the protocol goals**
  - *Bob requires data origin authentication of the message received from Alice*
  - *Bob requires non-repudiation of the message received from Alice*

Specifying the protocol



- **If you define - ensure goals are satisfied!**
- **If you analyze – check if goals are satisfied!**

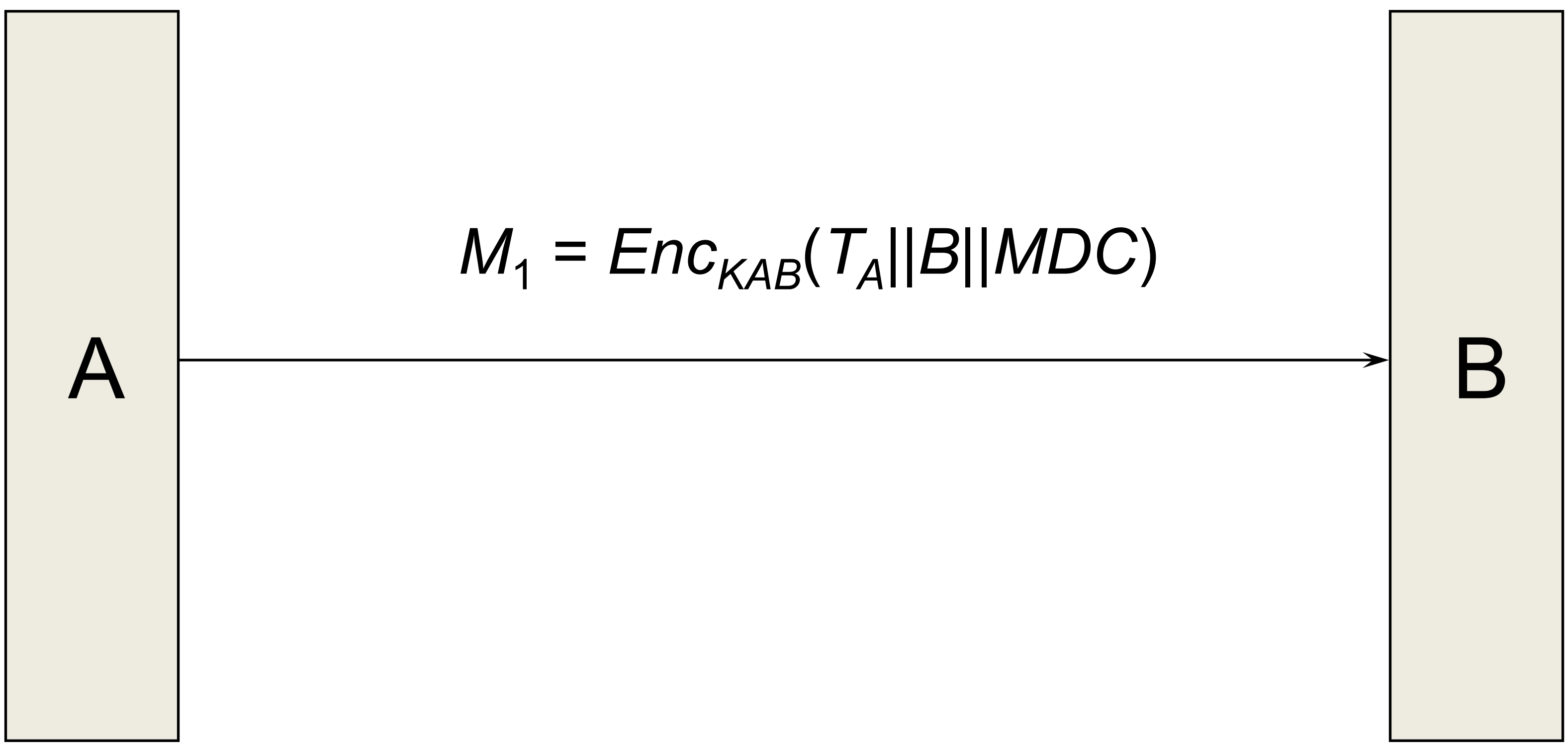
Notation

- *A* and *B* are (identifiers for) two entities who wish to engage in an authentication protocol.
- T_A is a time stamp produced by *A*.
- R_A is a random nonce generated by *A*.
- KAB is a symmetric key shared by *A* and *B*.
- *Text* is an arbitrary field that can contain data of any form, particularly an MDC.

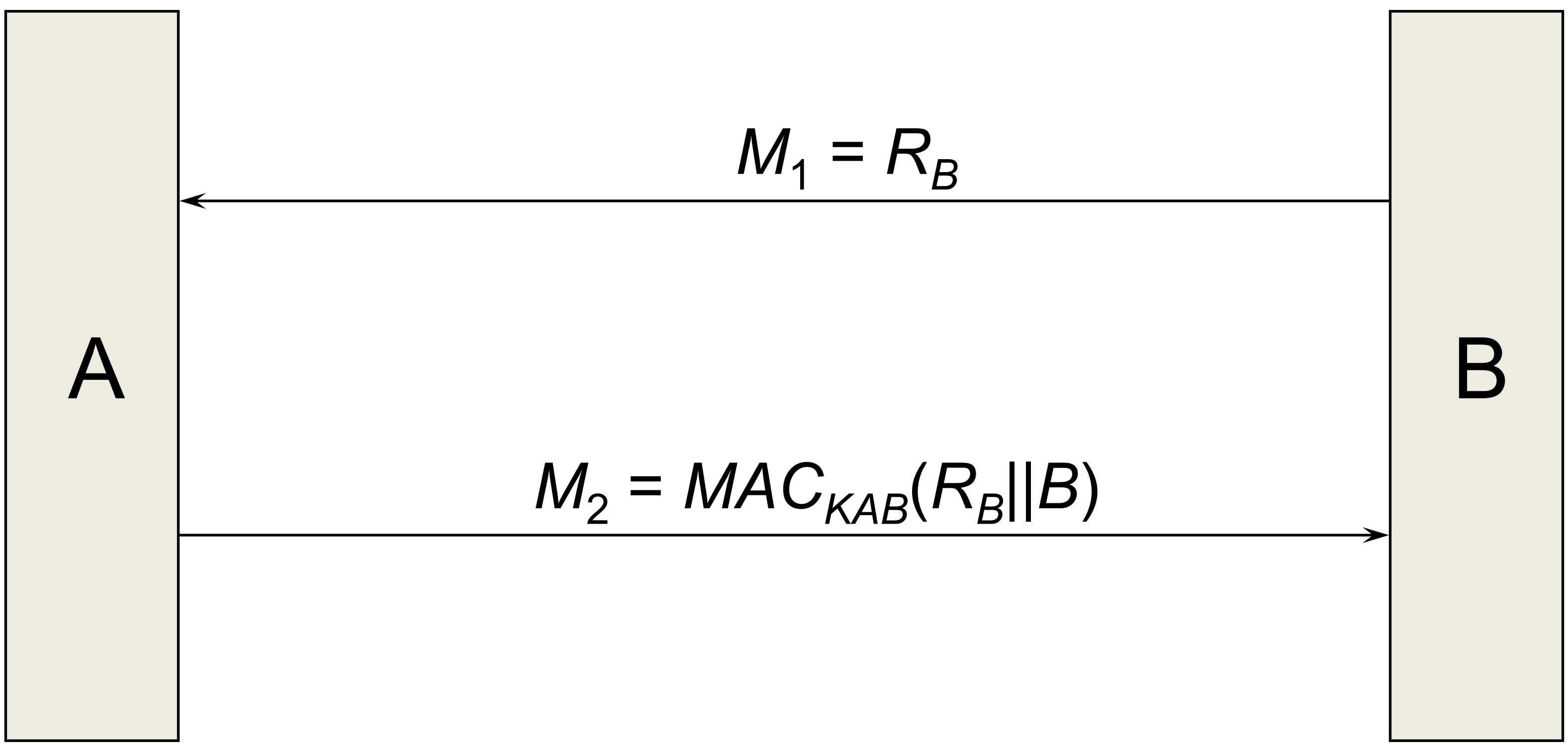
Notation

- $Enc_{KAB}(X)$ denotes the encryption of data X using a key KAB that is shared between A and B. We assume this is "integrity protected" encryption.
- $MAC_{KAB}(X)$ denotes a cryptographic check value (MAC) of data X using a key KAB that is shared between A and B.
- $Sig_A(X)$ denotes the signature (with appendix) computed by A on the data X.

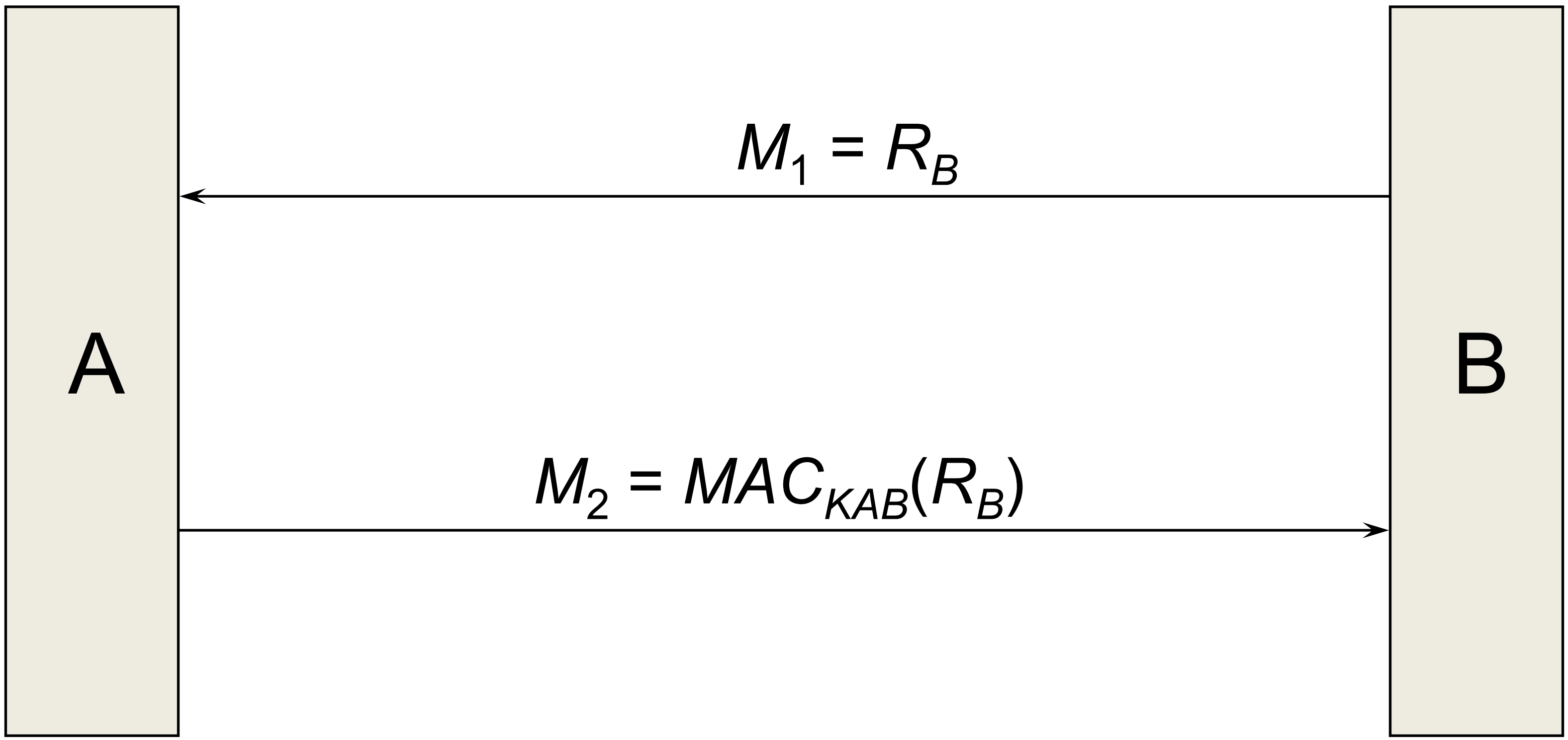
Example 1 (timestamp & encryption)



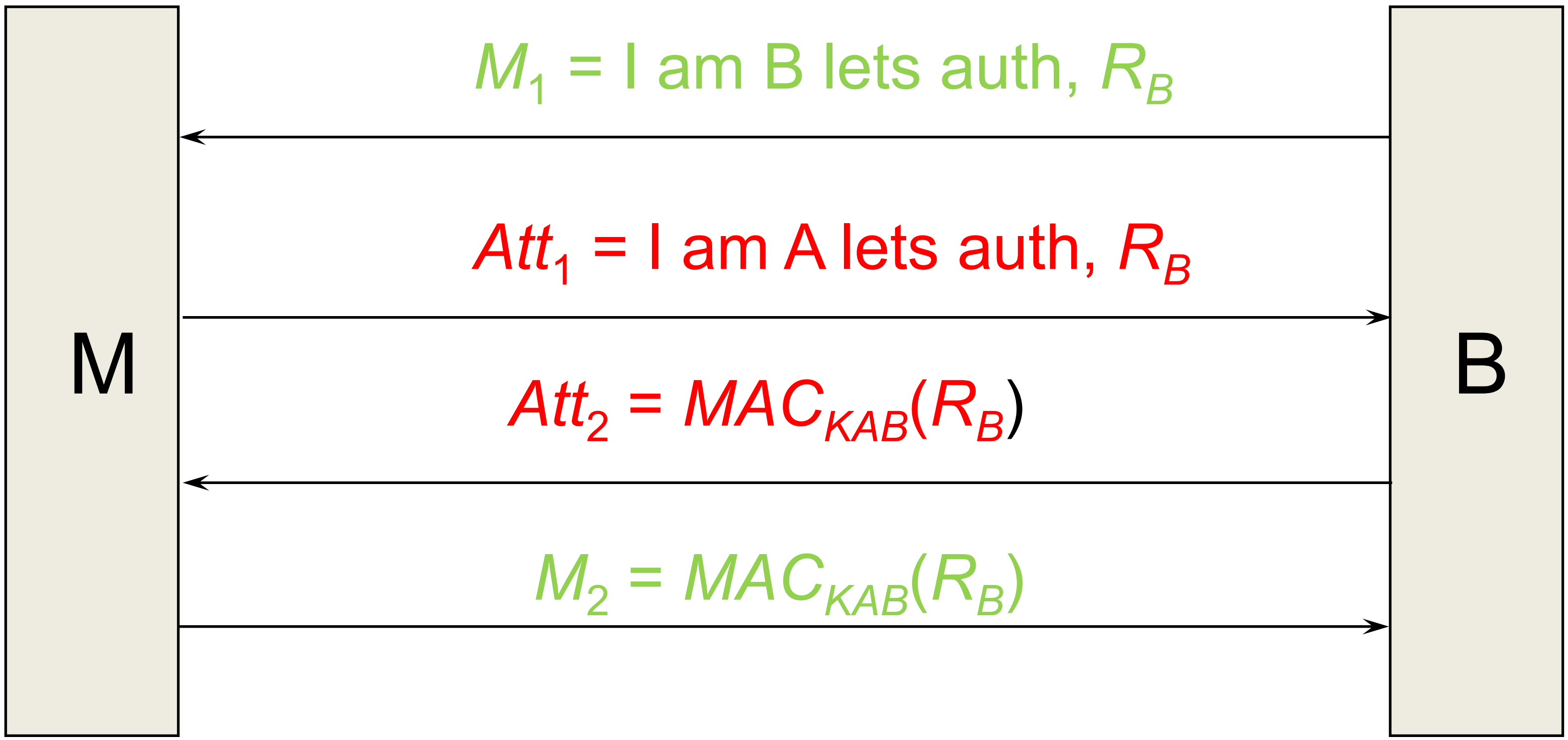
Example 2 (nonce & MAC)



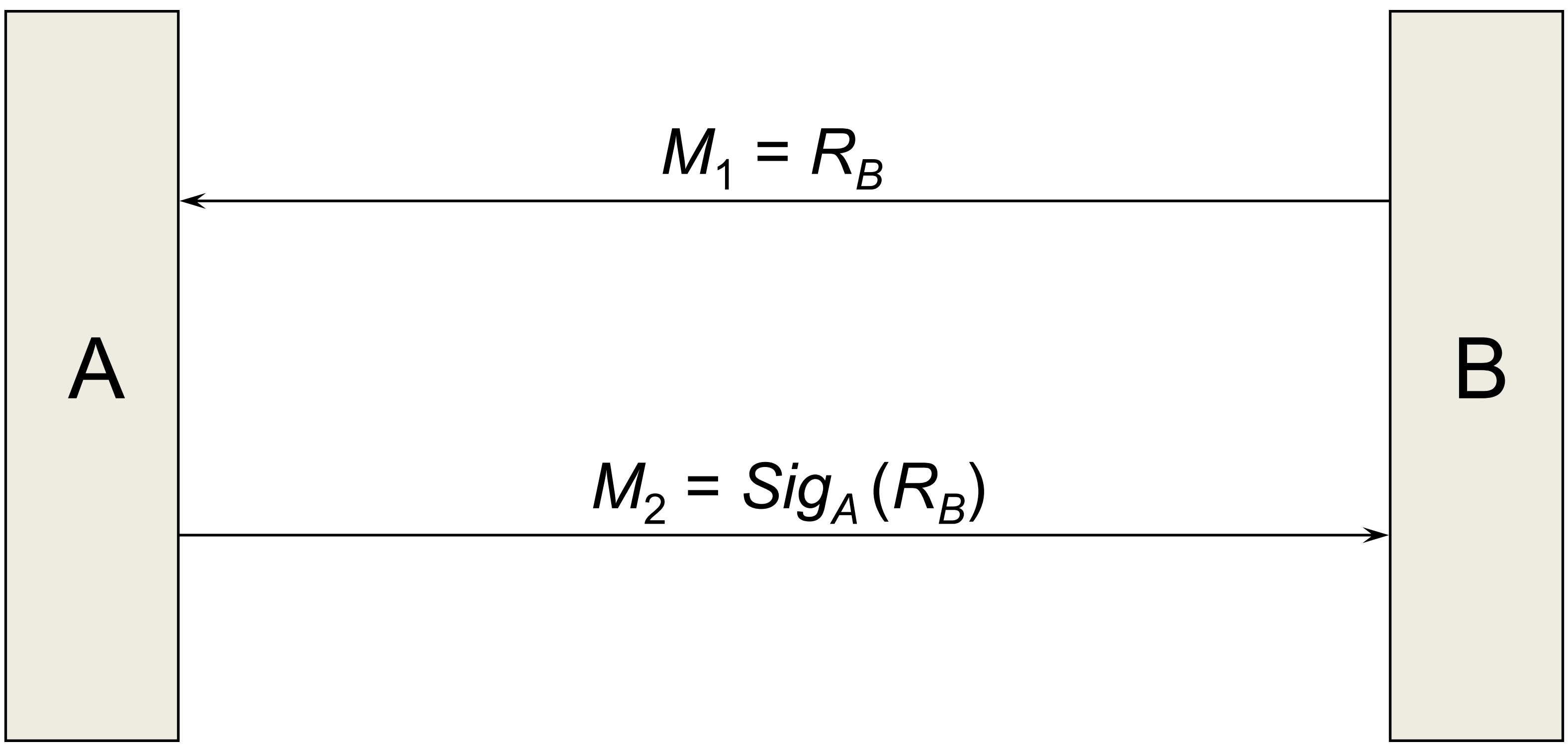
Example 2 (nonce & MAC v2)



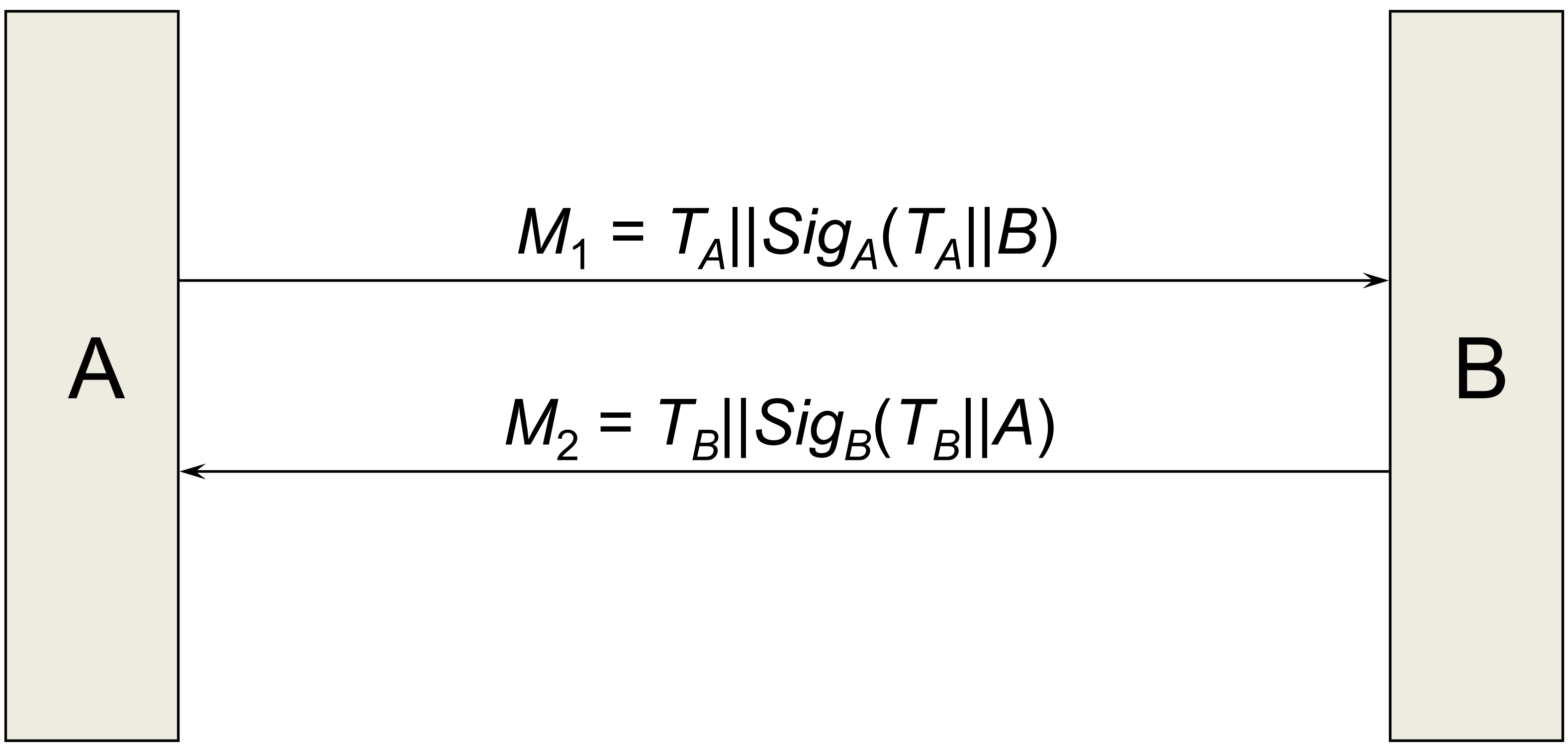
Example 2 (Reflection attack)



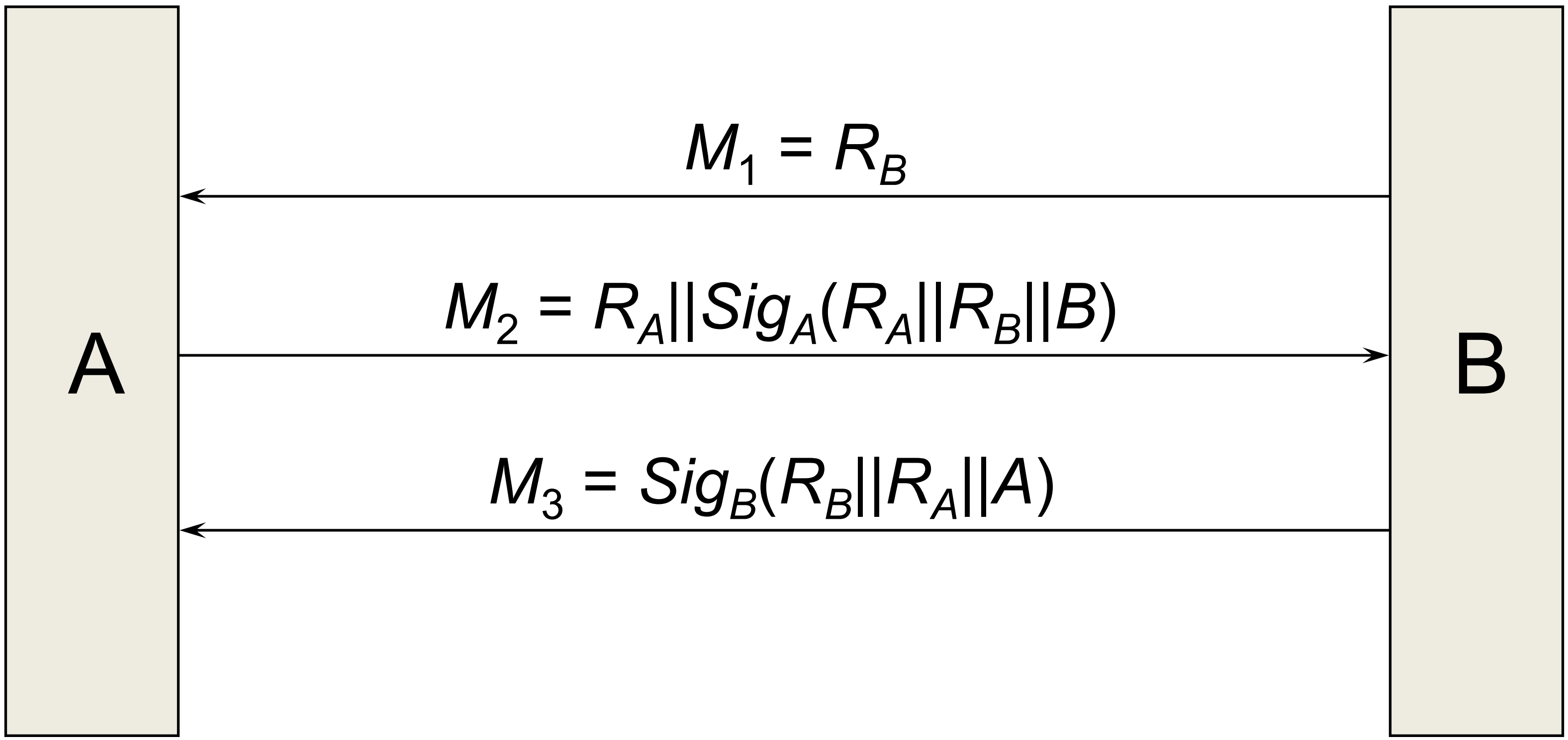
Example 2 v2 (nonce & sign)



Example 3 (timestamp & signatures)

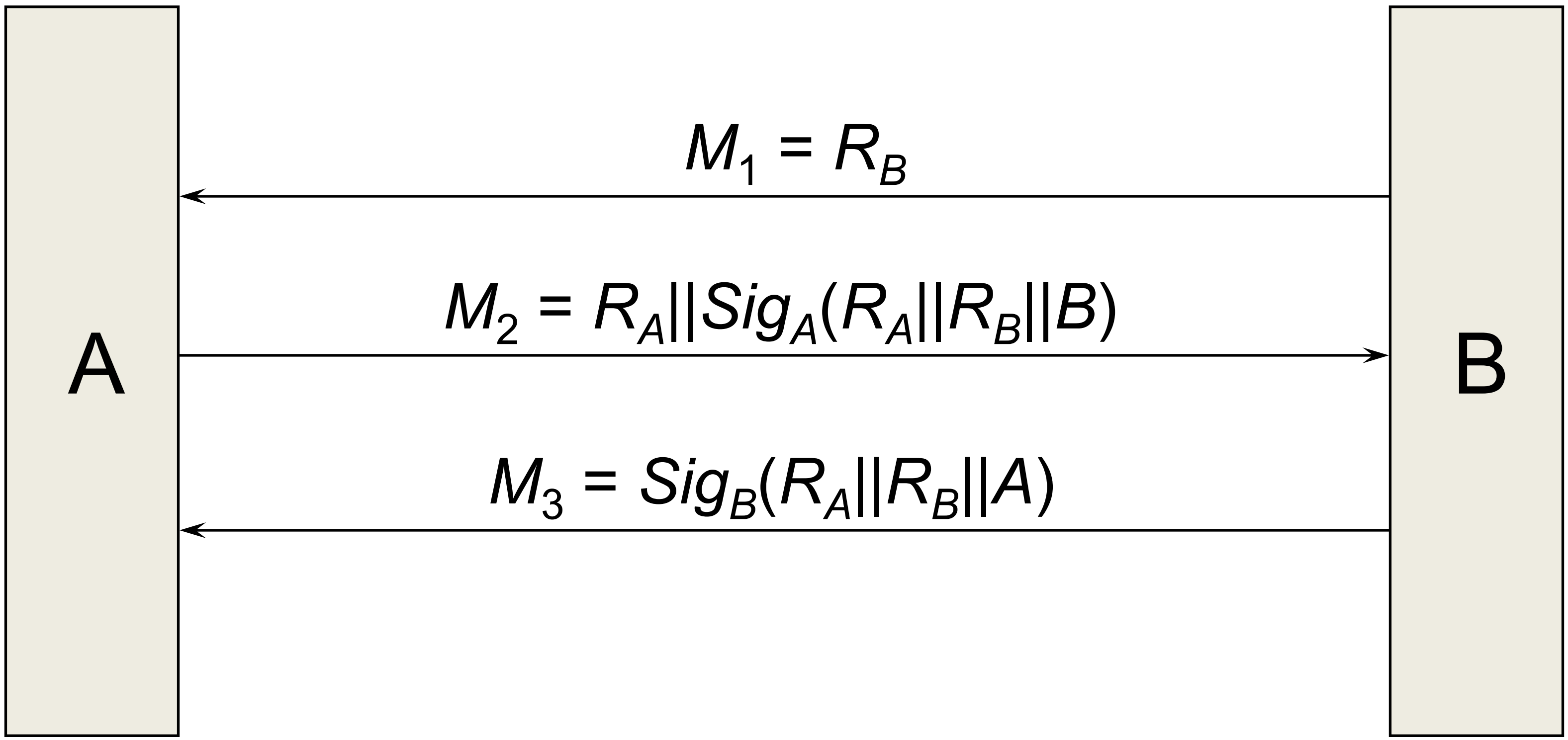


Example 4 (nonce & signature)



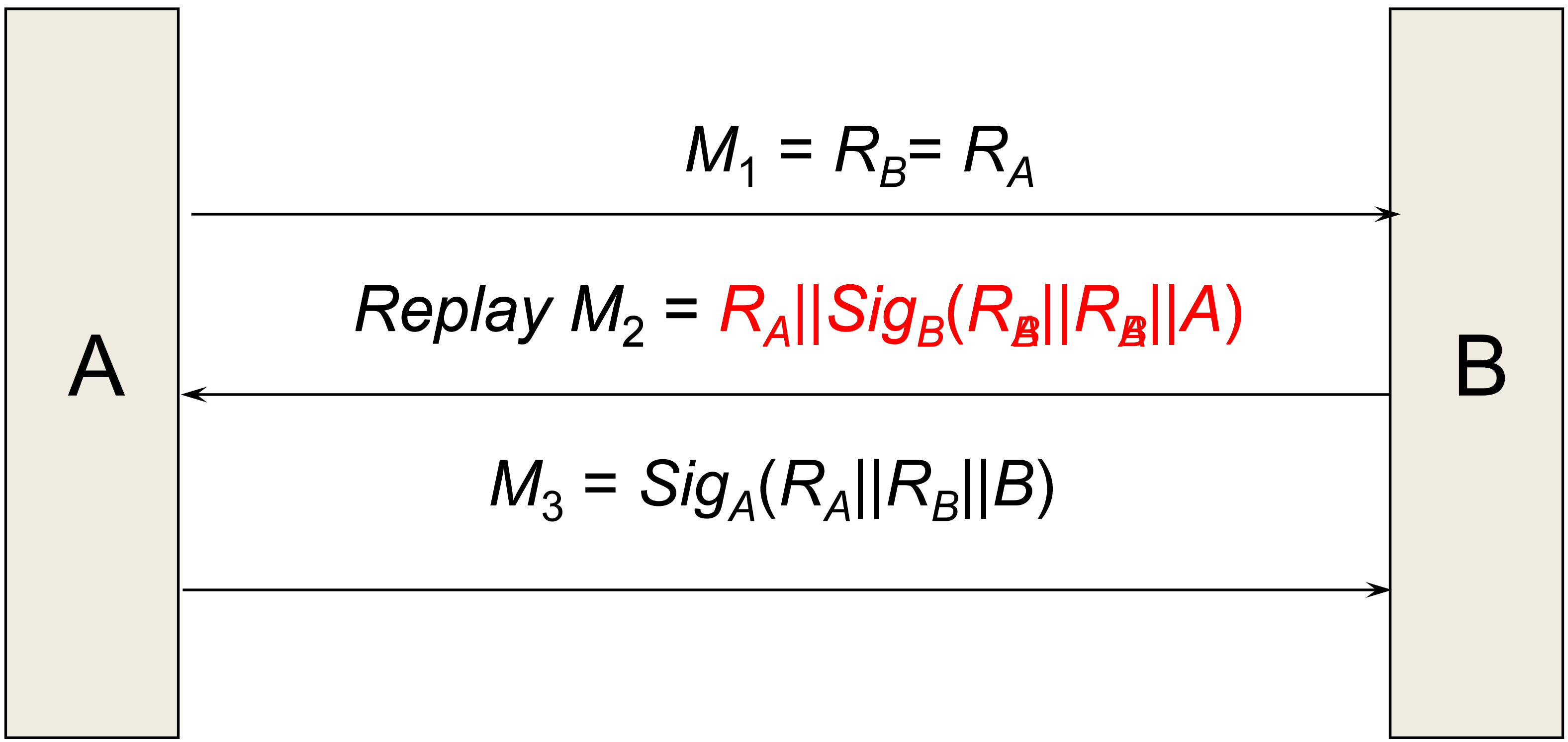
Example 5 (nonce & signature)

Assume R_B,R_A are counters $R_B > R_A$



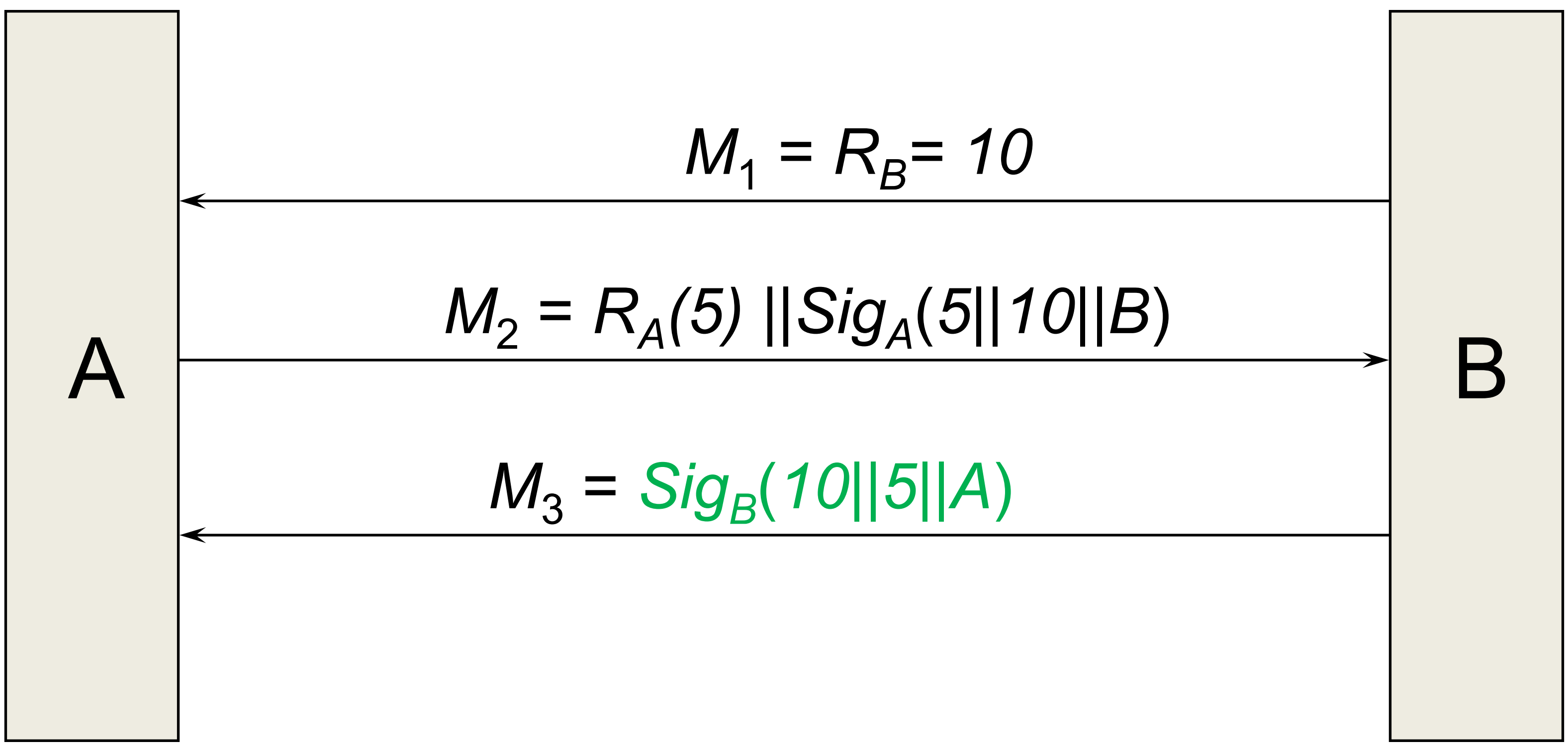
Example 5 (nonce & signature)

Assume R_B, R_A *are counters* $R_B > R_A$



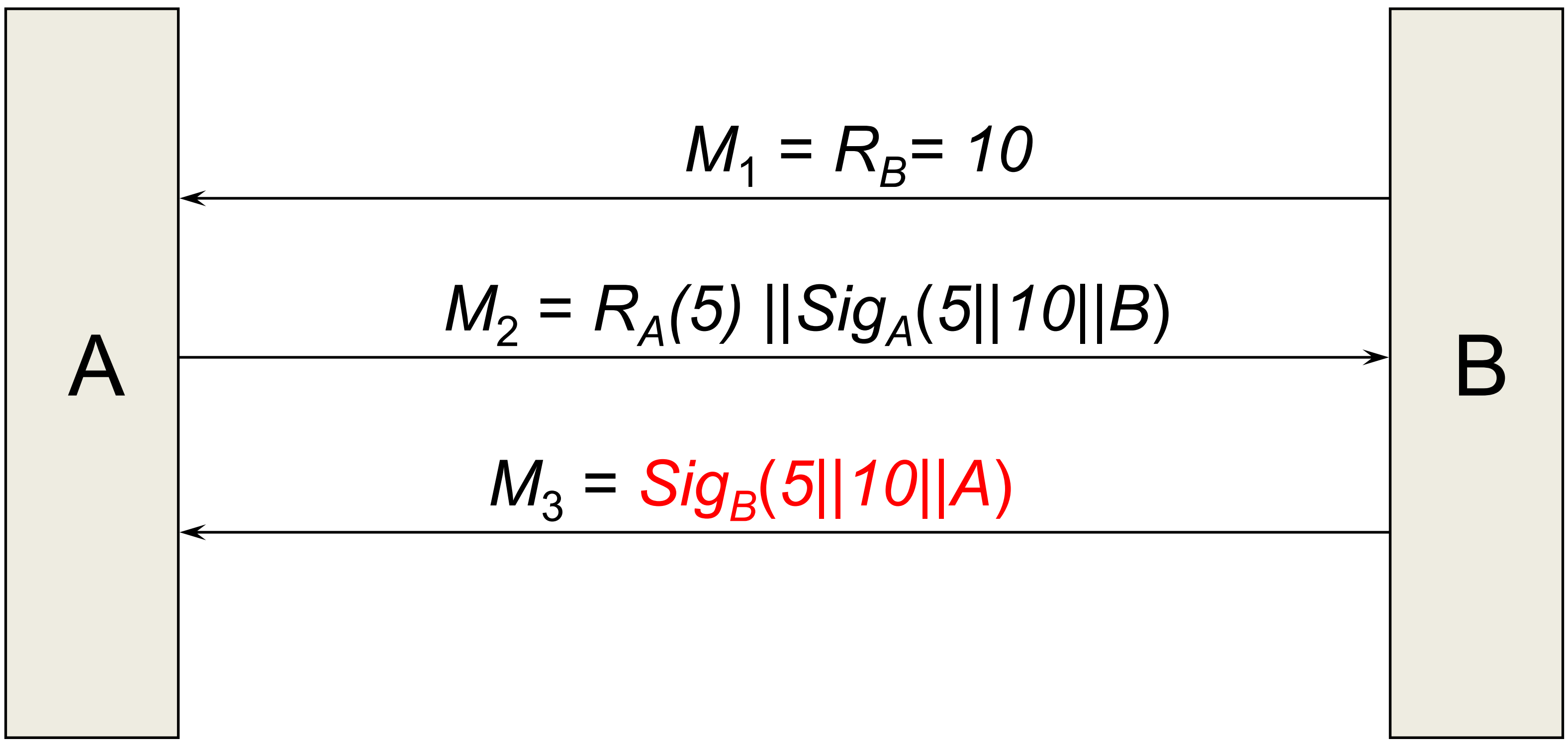
Example 5 (nonce & signature) with swopped nonces

Assume R_B=10,R_A *= 5 are counters* $R_B > R_A$

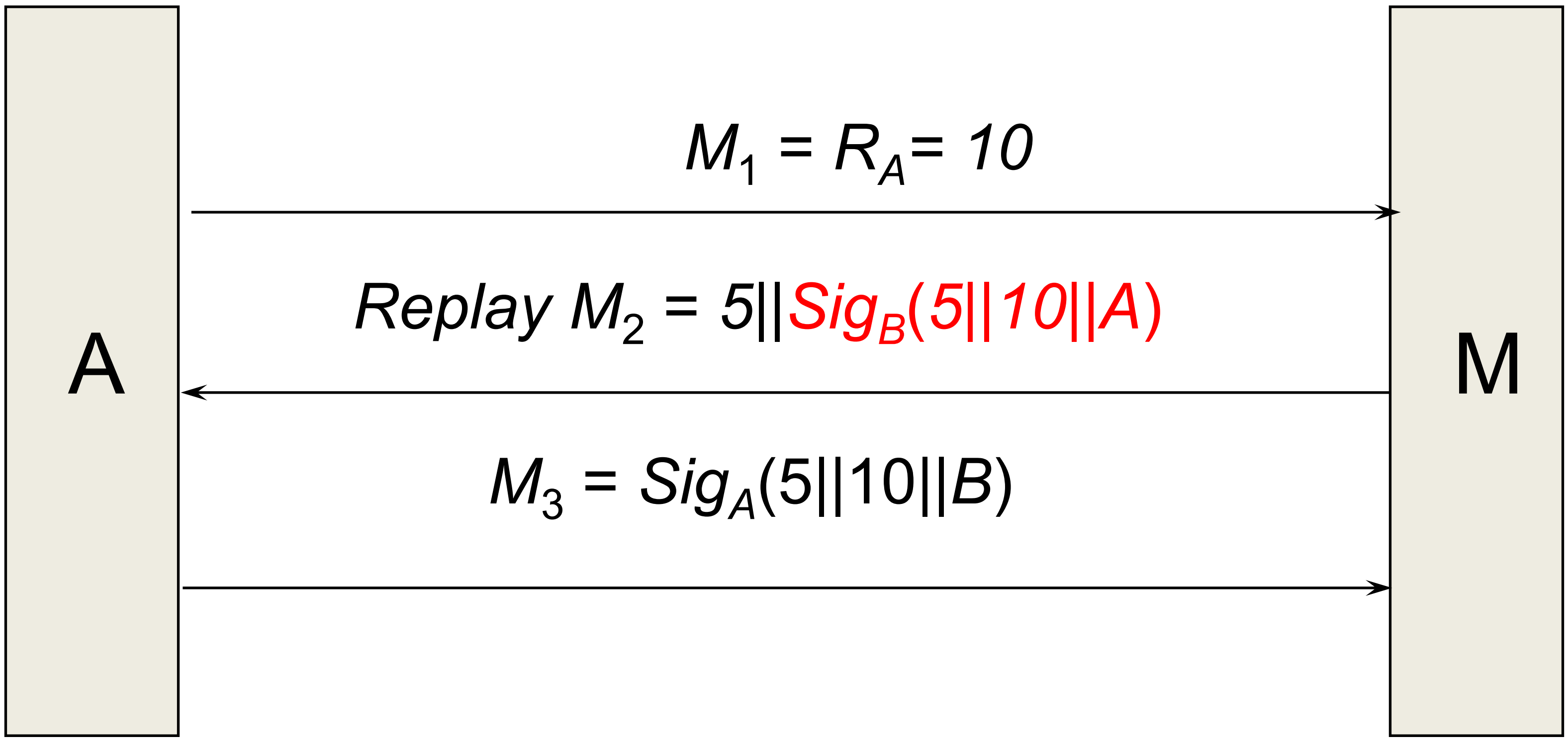


Example 5 (nonce & signature) without swopped nonces

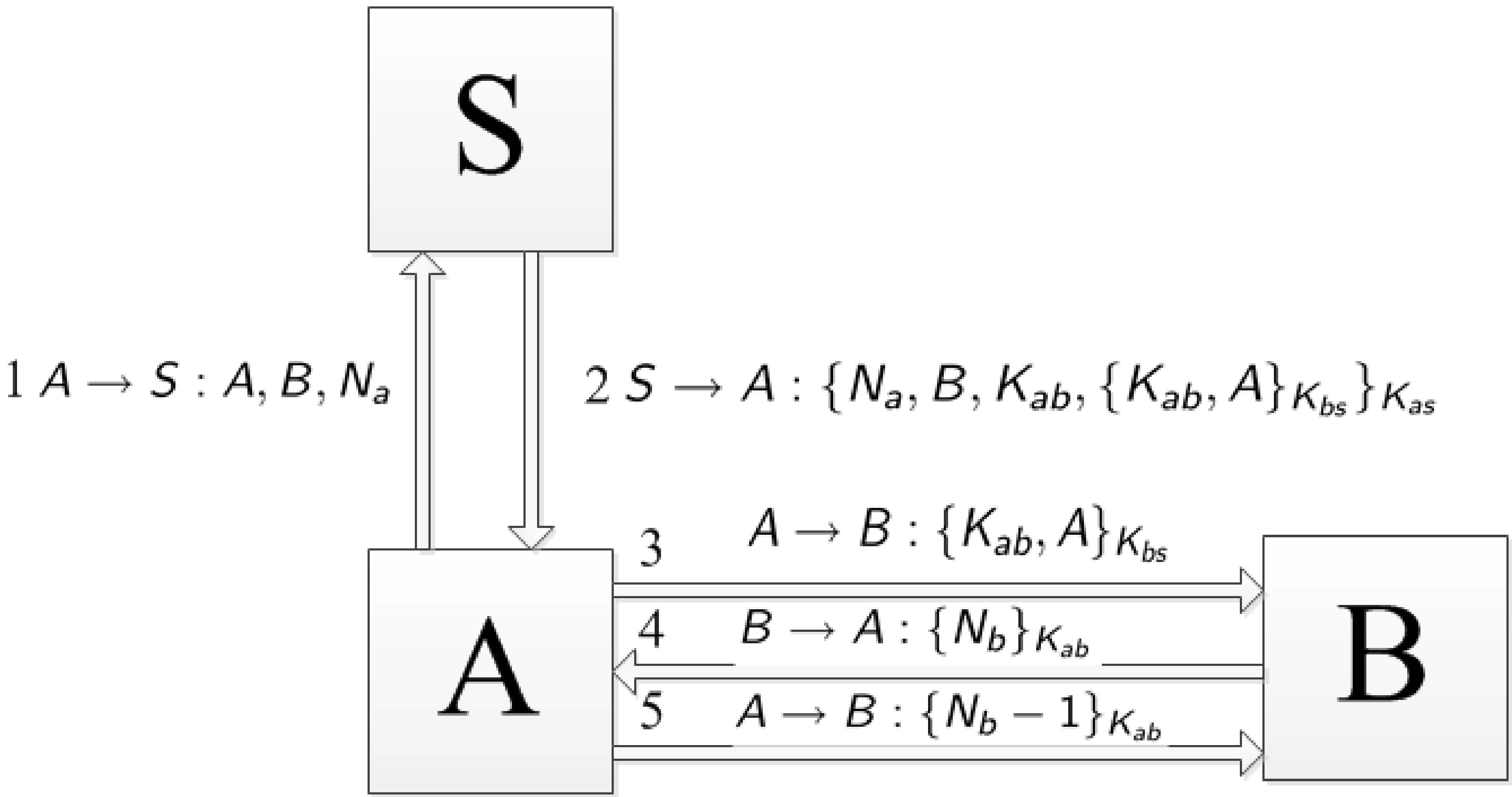
Assume R_B=10,R_A = *5 are counters* $R_B > R_A$



Later A so happens to use M1 that is equal to message M1 that B used before. This allows M to pretend to be B.



Just for fun...


S
A
B
$1\ A \rightarrow S : A, B, N_a$
$2\ S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
$3\quad A \rightarrow B : \{K_{ab}, A\}_{K_{bs}}$
$4\quad B \rightarrow A : \{N_b\}_{K_{ab}}$
$5\quad A \rightarrow B : \{N_b - 1\}_{K_{ab}}$

The end!

?

Any questions…