CS 5430

Secure Channel

Prof. Clarkson
Spring 2016

Review: Encryption, MACs

- We can protect confidentiality and integrity of a message against Dolev-Yao attacker

- But what if we want to have a conversation not just a single message...?

Protection of channel

- **Threat:** attacker who controls the network
  - Dolev-Yao model: attacker can read, modify, delete messages
- **Harm:** conversation can be learned (violating confidentiality) or changed (violating integrity) by attacker
- **Vulnerability:** communication channel between sender and receiver can be controlled by other principals
- **Countermeasure:** cryptography

SECURE CHANNEL

Secure channel

Channel:

- Bidirectional communication between two principals
- But their roles are not identical
  - Client and server, initiator and responder, etc.
  - We'll call them Alice and Bob
  - Same two principals might well have two parallel conversations in which they play different roles
- Communication might be...
  - spatial: over network
  - temporal: over storage
    - "Conversation with yourself"

Secure channel

Secure:

- The channel does not reveal anything about messages except for their timing and size (Confidentiality)
- If Alice sends a sequence of messages m1, m2, … then Bob receives a subsequence of that, and furthermore Bob knows which subsequence (Integrity)
  - And the same for Bob sending to Alice

Secure channel

Implications of security goals...

- No guarantee that any messages are ever received (subsequence could be empty) (no Availability goal)
- No attempt at anonymity
- No attempt to defend against traffic analysis
- Received messages:
  - are in order (or at least orderable)
  - are not modified
  - are attributable to the other principal

Secure channel

Pieces of the puzzle:

- Use authenticated encryption to protect confidentiality and integrity
  - Block cipher + mode
  - MAC
- Use message numbers to further protect integrity
- Use a key establishment protocol to create shared session key

Secure channel

Pieces of the puzzle:

- Use authenticated encryption to protect confidentiality and integrity
  - Block cipher + mode
  - MAC
- Use message numbers to further protect integrity
- Use a key establishment protocol to create shared session key

Message numbers

- Aka sequence numbers
- Every message that Alice sends is numbered
  - 1, 2, 3, ...
  - numbers increase monotonically
  - never reuse a number
- Bob keeps state to remember last message number he received
- Bob accepts only increasing message numbers
- And ditto all the above, for Bob sending to Alice
  - so each principal keeps two independent counters: messages sent, messages received

Message numbers

What if Bob detects a gap? e.g. 1, 2, 5

- Maybe Mallory deleted messages 3 and 4 from network
- Maybe Mallory detectably changed 3 and 4, causing Bob to discard them
- In either case, channel is under active attack
  - Absent availability goals, time to PANIC: abort protocol, produce appropriate information for later auditing, shut down channel

What if network non-maliciously dropped messages or will deliver them later?

- Let's assume underlying transport protocol guarantees that won't happen (e.g. TCP)

Message numbers

- Message number usually implemented as a fixed-size unsigned integer, e.g., 32 or 48 or 64 bits
- What if that `int` overflows and wraps back around to 0?
  - Message number **must** be unique within conversation to prevent Mallory from replaying old conversation
  - So conversation **must** stop at that point
  - Can start a new conversation with a new session key

Secure channel

Pieces of the puzzle:

- Use authenticated encryption to protect confidentiality and integrity
  - Block cipher + mode
  - MAC
- Use message numbers to further protect integrity
- Use a key establishment protocol to create shared session key [part 1]

Session keys

- For now, let's assume Alice and Bob already have a single shared session key k
  - Recall: *session key* is used for limited time then discarded
  - Here, the session duration is a single conversation
- But a single key isn't good enough…
  - Need a key for the block cipher
  - Need a key for the MAC
- And recall:
  - **Principle:** every key in system should have unique purpose
  - Implies: should not use same key for both Enc and MAC algorithms
  - Also implies: should not use same keys for
    - Alice –> Bob, vs.
    - Bob –> Alice

Key derivation

- Have one key: k
- Need four keys:
  1. kea: Encrypt Alice to Bob
  2. keb: Encrypt Bob to Alice
  3. kma: MAC Alice to Bob
  4. kmb: MAC Bob to Alice
- How to get four out of one: use a cryptographic hash function H to derive keys...
  1. kea = H(k, "Enc Alice to Bob")
  2. keb = H(k, "Enc Bob to Alice")
  3. kma = H(k, "MAC Alice to Bob")
  4. kmb = H(k, "MAC Bob to Alice")

Key derivation

- Why hash?
  - Destroys any structure in input
  - Produces a fixed-size output that can be truncated, as necessary, to produce key for underlying algorithm
  - Unlikely to ever cause any of four keys to collide
  - Even if one of four keys ever leaks, hard to invert hash to recover k and learn the other keys
- Small problem: maybe the output of H isn't compatible with the output of Gen
  - For most block ciphers and MACs, not a problem
    - they happily take any uniformly random sequence of bits of the right length as keys
  - For DES, it is a problem
    - has weak keys that Gen should reject
  - For many asymmetric algorithms, it would be a problem
    - keys have to satisfy certain algebraic properties

Secure channel

Pieces of the puzzle:

- Use authenticated encryption to protect confidentiality and integrity
  - Block cipher + mode
  - MAC
- Use message numbers to further protect integrity
- Use a key establishment protocol to create shared session key

Authenticated encryption

- We saw three notions last time:
  - Enc and MAC
  - Enc then MAC
  - MAC then Enc
- Let's unify all with a pair of algorithms:
  - AuthEnc(m; ke; km): produce an authenticated ciphertext x of message m under encryption key ke and MAC key km
  - AuthDec(x; ke; km): recover the plaintext message m from authenticated ciphertext x, and verify that the MAC is valid, using ke and km
    - Abort if MAC is invalid
  - (And later in lecture: provide only a single key as argument instead of two, since we know we can derive keys)

To send a message from A to B

```
1. A:
      increment sent_ctr;
      if sent_ctr overflows then abort;
      x = AuthEnc(sent_ctr, m; kea; kma)
2. A -> B: x
3. B:
      i,m = AuthDec(x; kea; kma);
      increment rcvd_ctr;
      if i != rcvd_ctr then abort;
      output m
```

To send a message from B to A

```
1. B:
      increment sent_ctr;
      if sent_ctr overflows then abort;
      x = AuthEnc(sent_ctr, m; keb; kmb)
2. B -> A: x
3. A:
      i,m = AuthDec(x; keb; kmb);
      increment rcvd_ctr;
      if i != rcvd_ctr then abort;
      output m
```

Secure channel

Pieces of the puzzle:

- Use authenticated encryption to protect confidentiality and integrity
  - Block cipher + mode
  - MAC
- Use message numbers to further protect integrity
- Use a key establishment protocol to create shared session key [part 2]

Session key generation

Back to this assumption:

For now, let's assume Alice and Bob already have a single shared session key k

We need a means for Alice and Bob to generate that key...

Key establishment

Theorem [Boyd 1993]: impossible to establish secure channel between principals who do not already...

- share a key with each other, or
- separately share a key with a trusted third party, or
- have the means to ascertain a public key for each other

...i.e., you can't get something for nothing

If one of those requirements is met, can run protocol to establish session key

Key establishment

- Terminology:
  - **user** is a principal who will use the generated session key for further communication
  - other **principals** might be involved but won't learn or use the key
- Key transport protocol: session key is generated by one principal then transferred to all users
- Key agreement protocol: session key is generated as a function of inputs from all users and transferred to all users

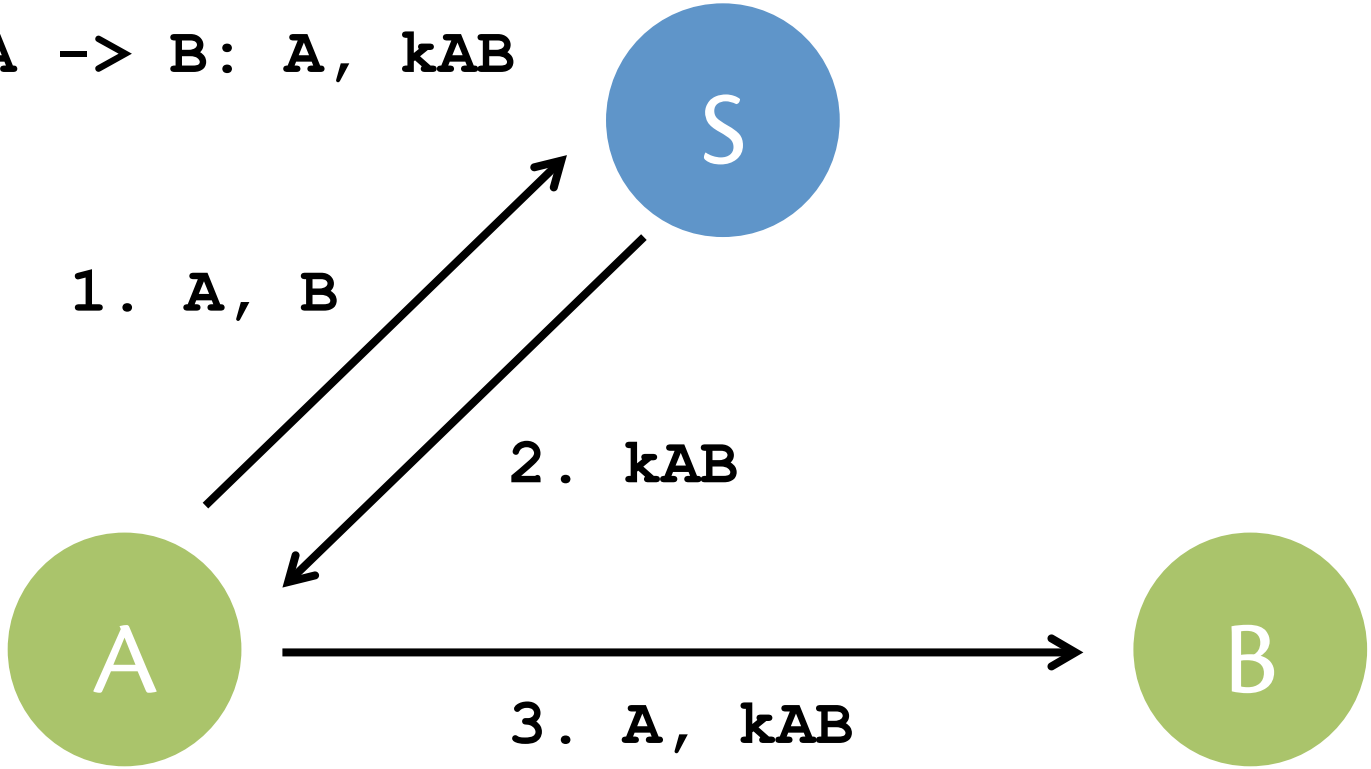
Let's build something really simple... a key transport protocol with a trusted server who picks the session key

Transport protocol

- **Assume:** trusted server S with whom A and B already share a long-term key
  - A shares kAS with S
  - B shares kBS with S
- **Output:** new session key kAB generated by S (then immediately forgotten by S)
- **Goals:**
  1. only A and B (and S) know that key (confidentiality)
  2. …

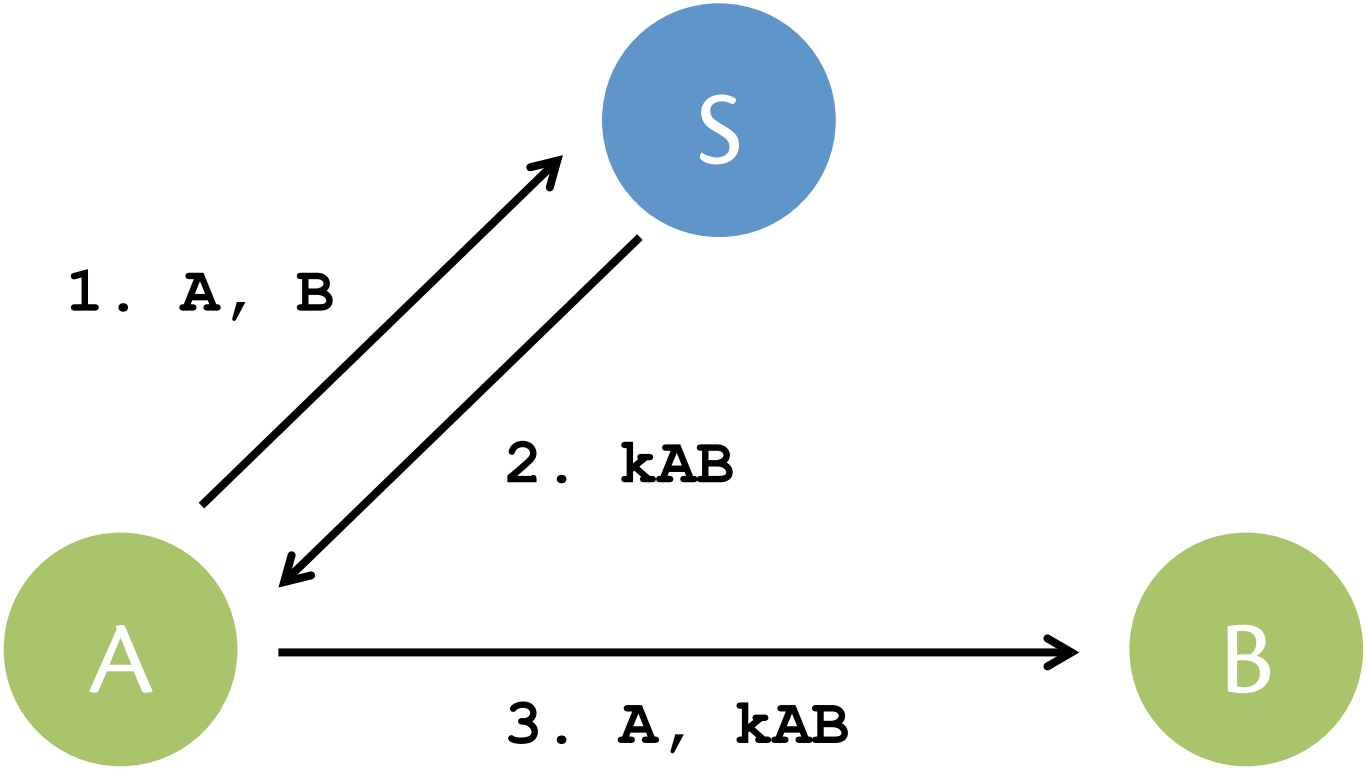
Protocol 1

```
1. A -> S: A, B
2. S -> A: kAB
3. A -> B: A, kAB
```



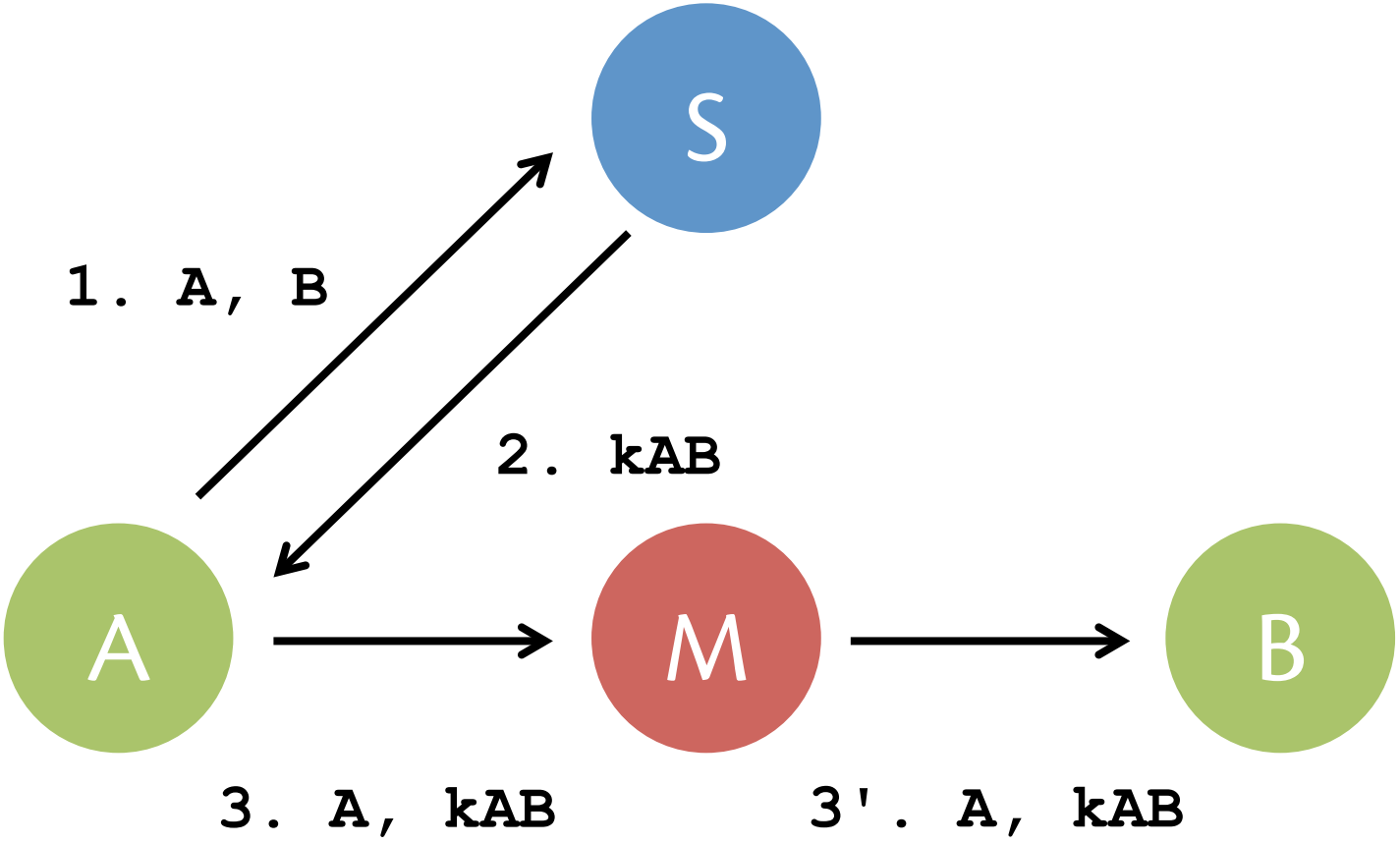
Protocol 1

Glaring problem: attacker learns kAB, violating goal 1

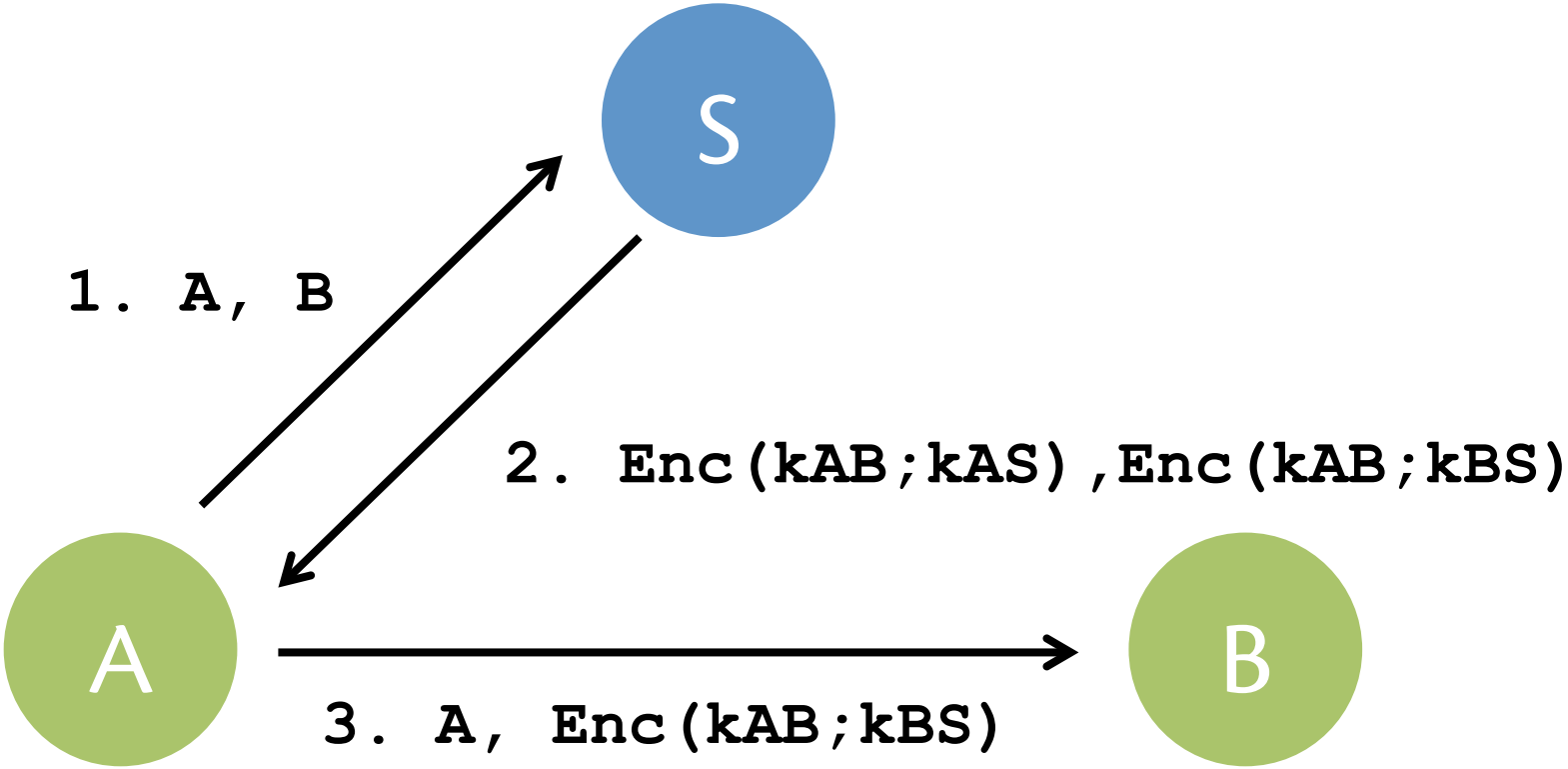


Protocol 1: Attack

Glaring problem: attacker learns kAB, violating goal 1



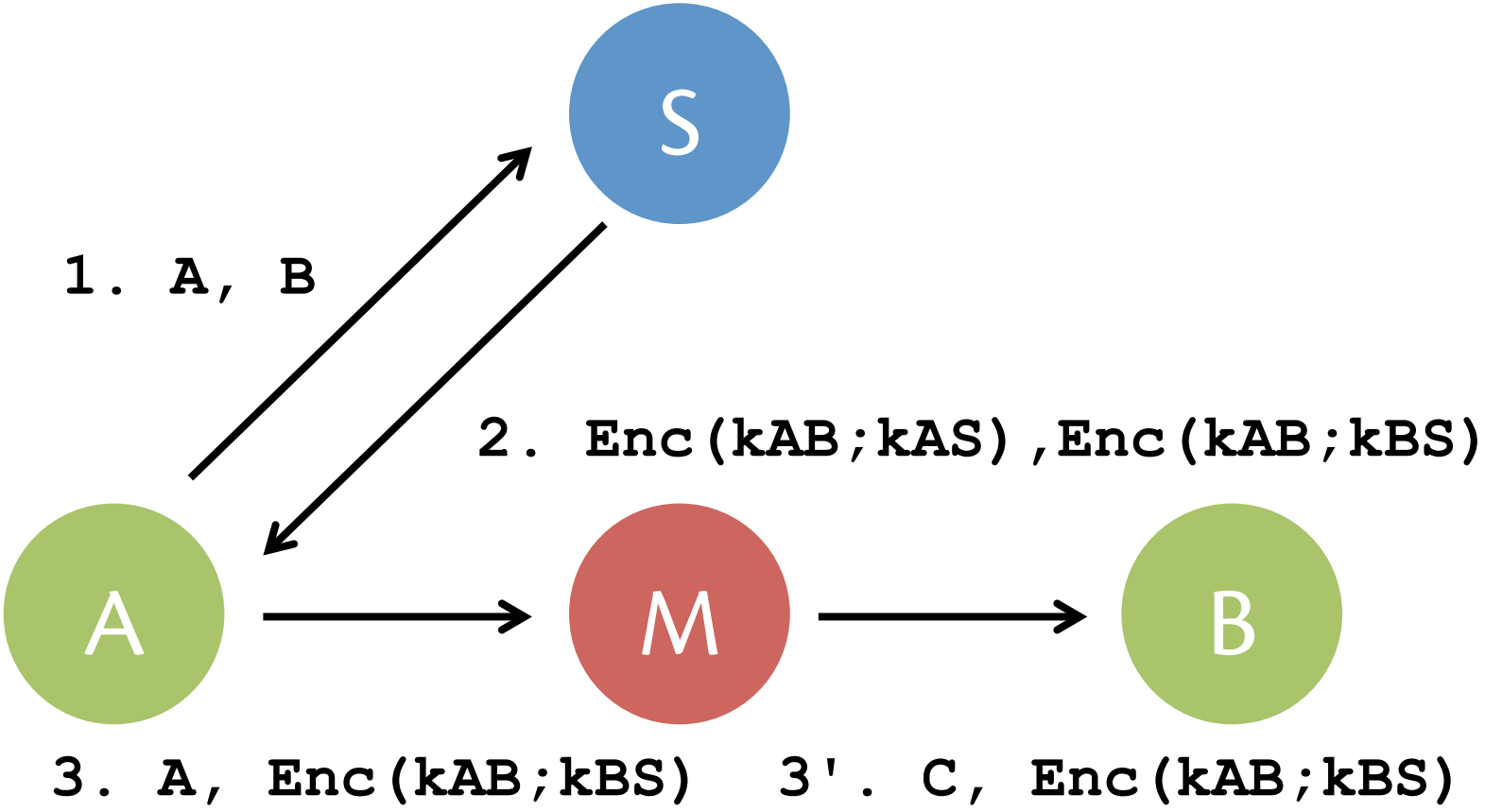
Protocol 2



Protocol 2: Attack 1

Problem: B believes key is shared with C rather than A

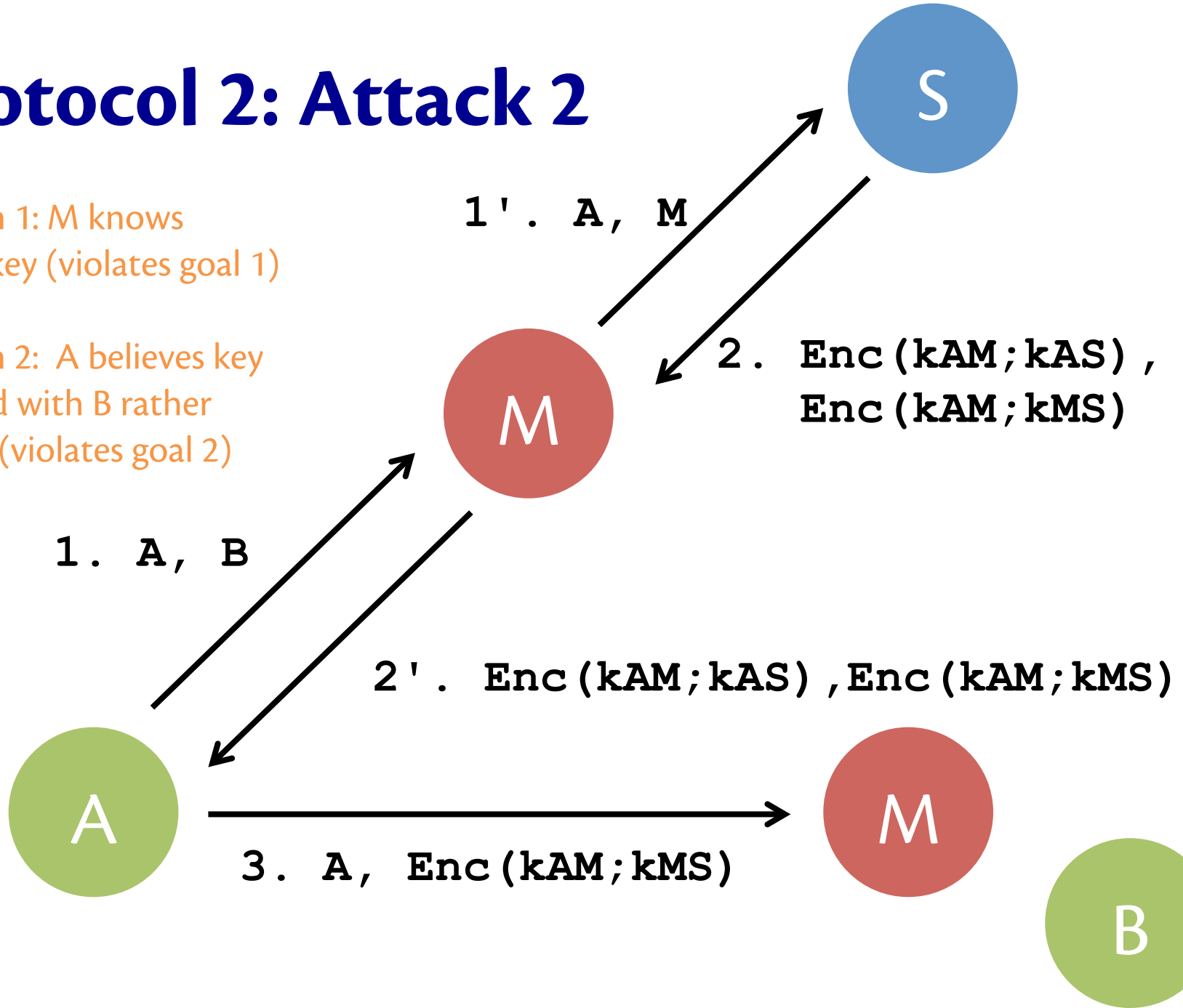
Goal: 2. Users associate key with correct principal identities (integrity)



Protocol 2: Attack 2

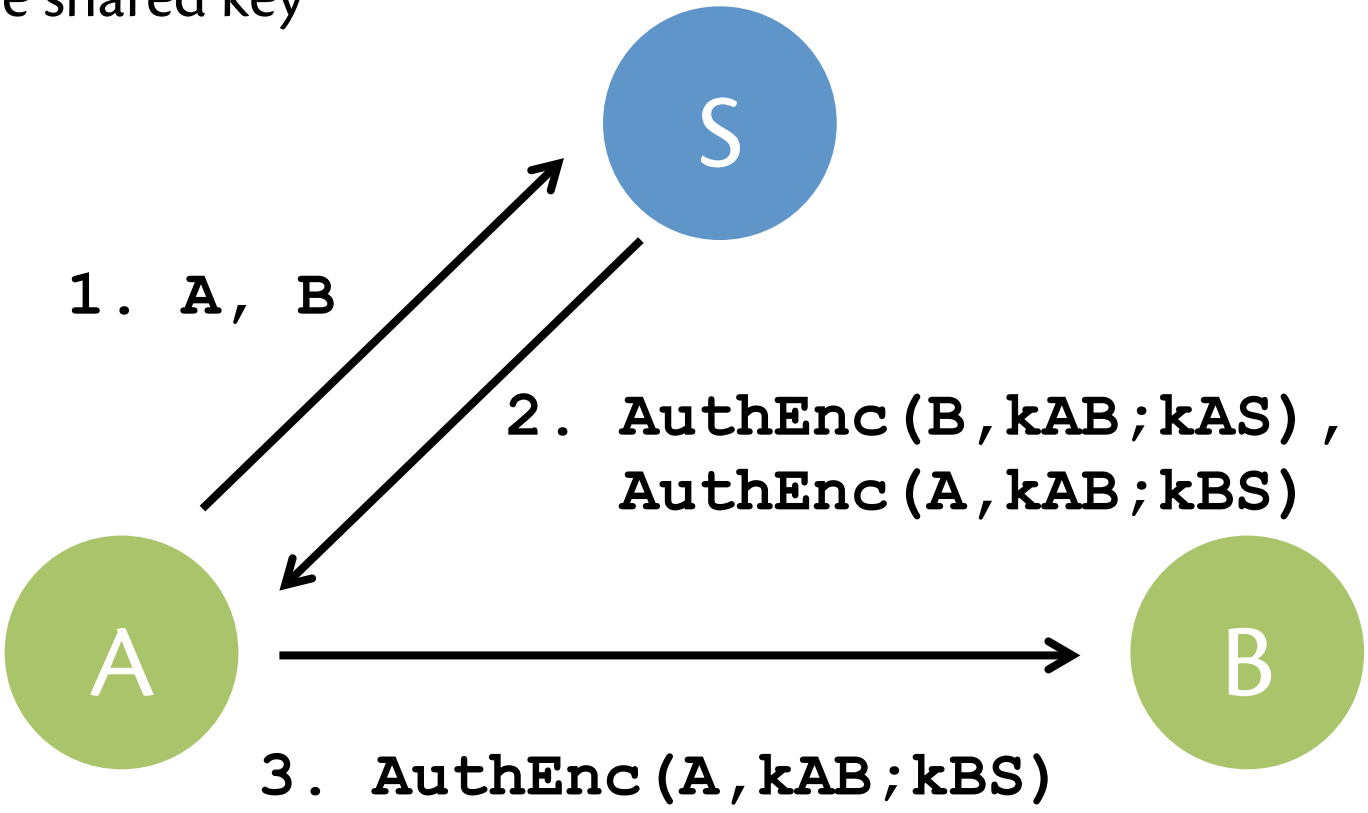
Problem 1: M knows shared key (violates goal 1)

Problem 2: A believes key is shared with B rather than M (violates goal 2)



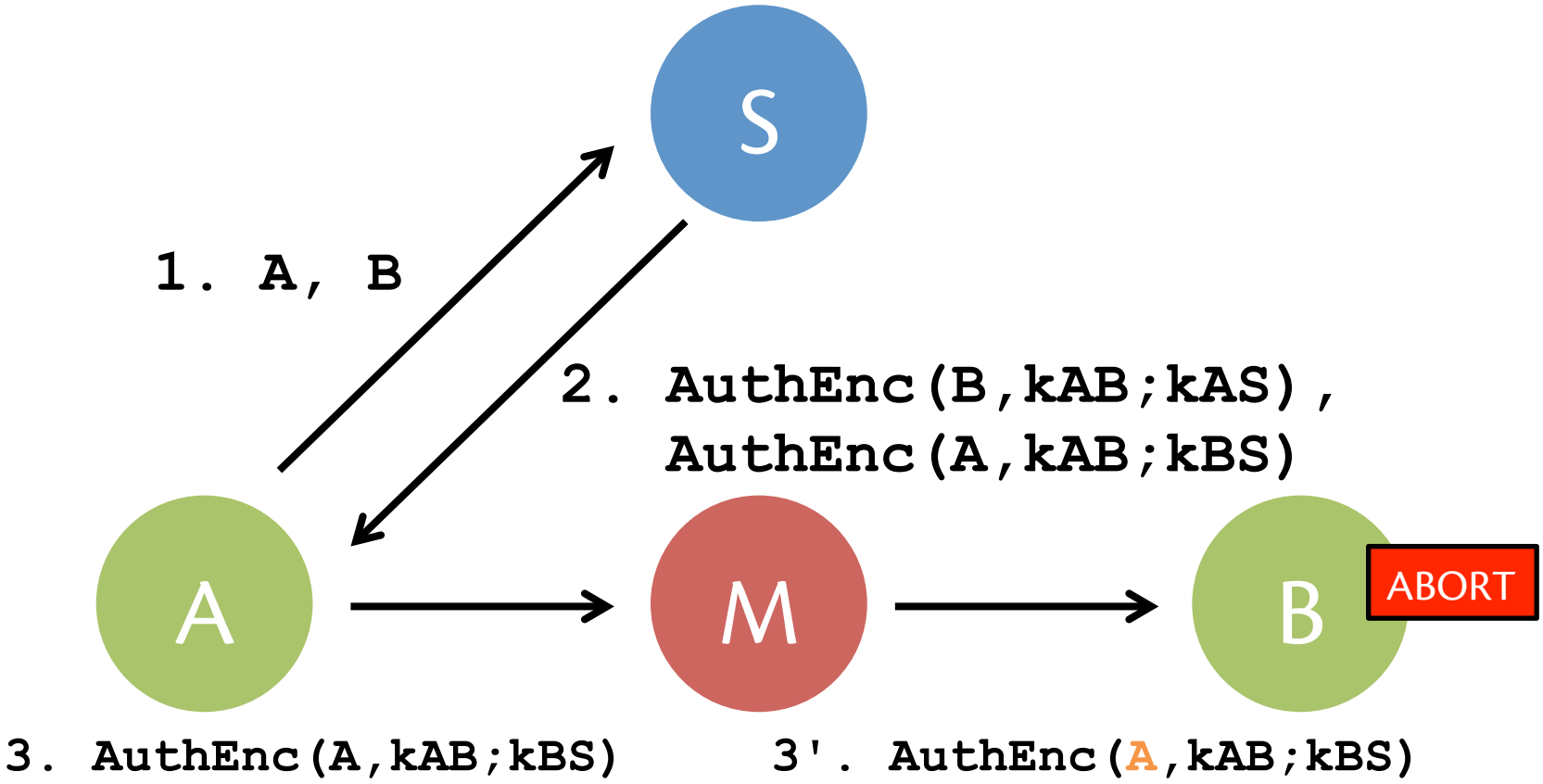
Protocol 3

Idea: use authenticated encryption to prevent M from changing principal identifiers; derive Enc and MAC key from single shared key



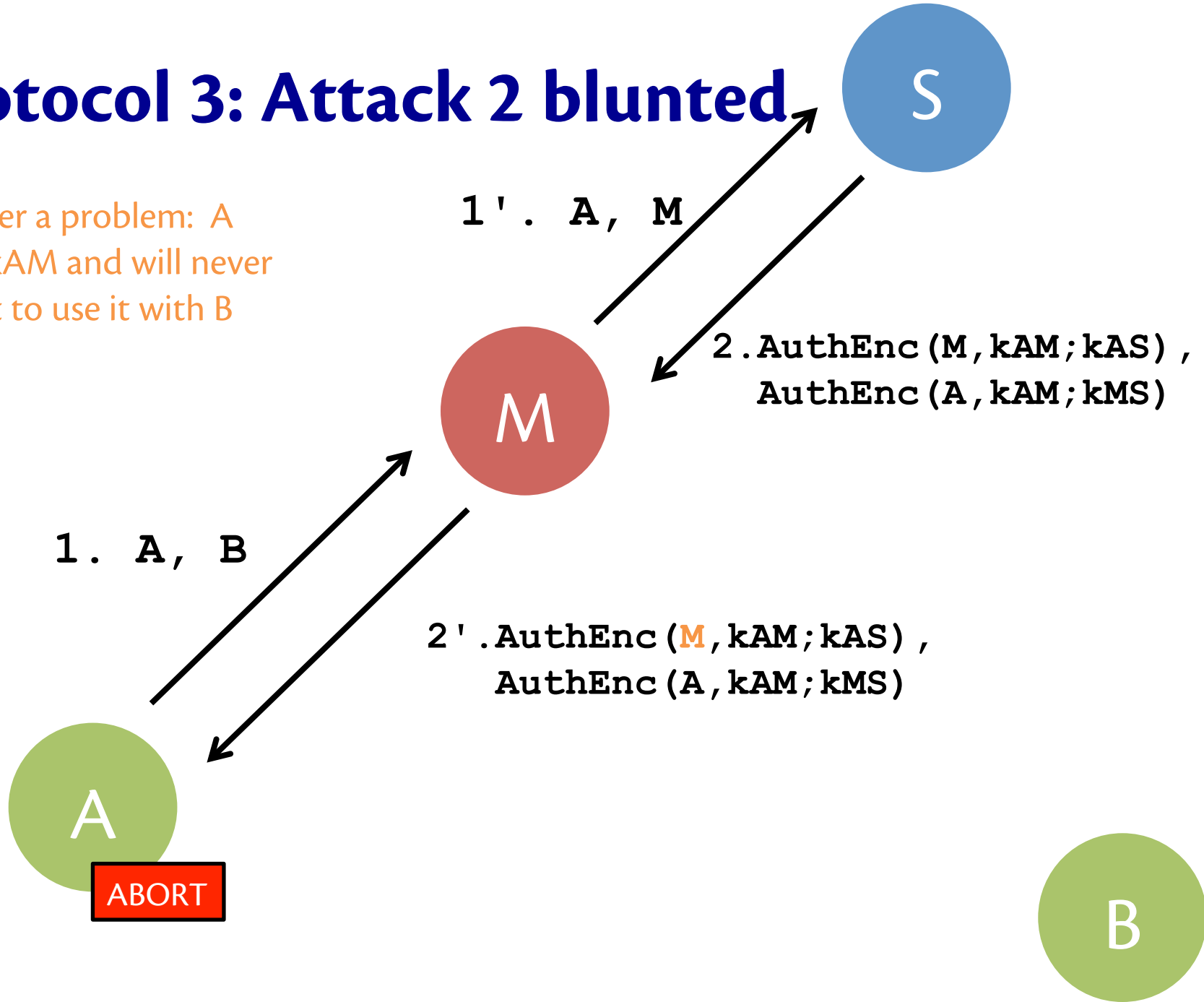
Protocol 3: Attack 1 blunted

No longer a problem: B correctly believes key is shared with A



Protocol 3: Attack 2 blunted

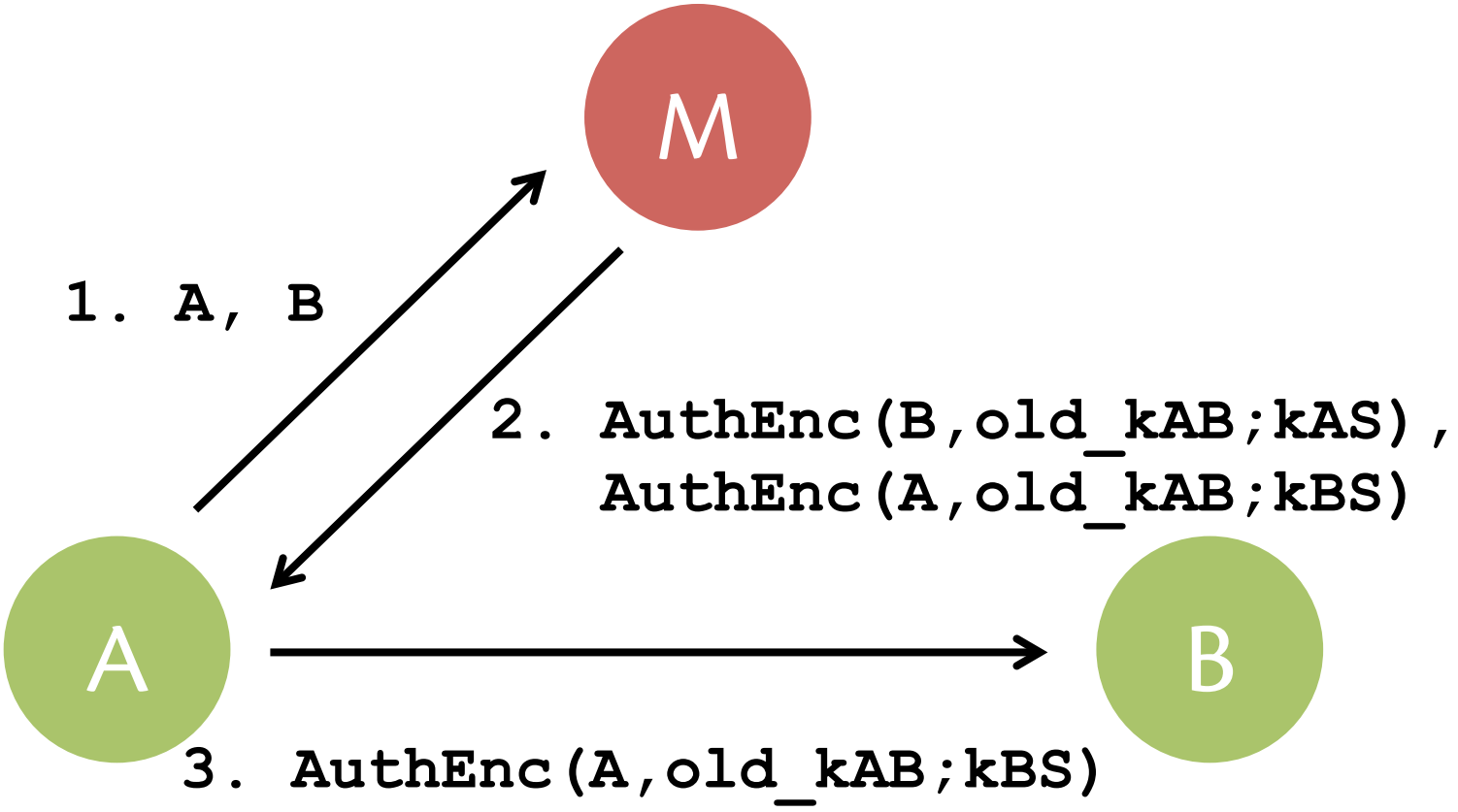
No longer a problem: A rejects kAM and will never attempt to use it with B



Protocol 3: Attack 3

Problem: M could *replay* messages from old session

Goal: 3. the session key is *fresh* (integrity)



TO BE CONTINUED...

Upcoming events

- [today] A3 out

Most conversations are simply monologues delivered in the presence of a witness. – Margaret Millar