

CS 725/825 & IT 725

Lecture 12

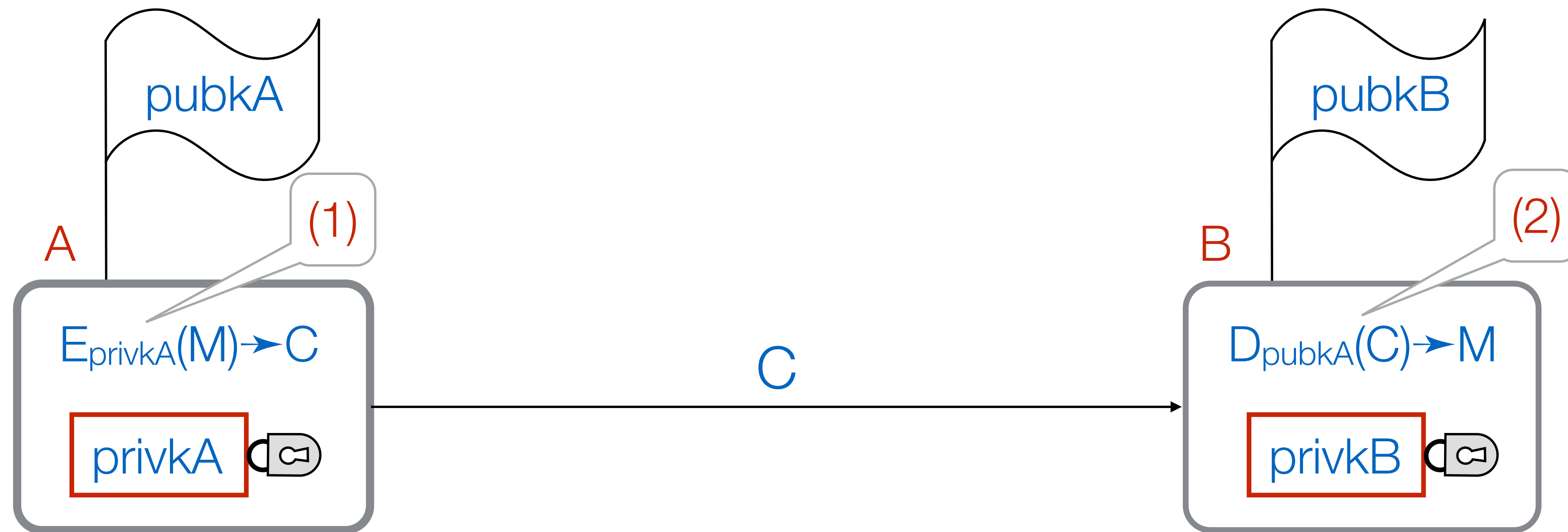
Network Security

October 6, 2025

Authentication

- ▶ Basic idea:
 - use “flipped” public/private key cryptography
 - only possessor of private key could have encrypted something that decrypts using its public key
- ▶ Problem: **Replay Attack**
 - solution: use of a nonce
- ▶ Still unaddressed: We need a trusted way to obtain someone's public key

Authentication - Basics



- (1) A “encrypts” the message: $E_{\text{privkA}}(M) \rightarrow C$ and sends it to B
- (2) B “decrypts” the message: $D_{\text{pubkA}}(C) \rightarrow M$
- (3) if M “looks OK”*, its sender is authenticated

* the assumption is that recipient can tell the difference between a valid message and a random string of bits that would be the result of decryption with a wrong key

Message Integrity

► Basic idea:

- use public/private key cryptography
- send a hash of a message (aka message digest) ...
- ... encrypted using sender's private key
- if hash of the received message agrees with decrypted received hash, it is assumed that the message was not altered in transit

► Problems:

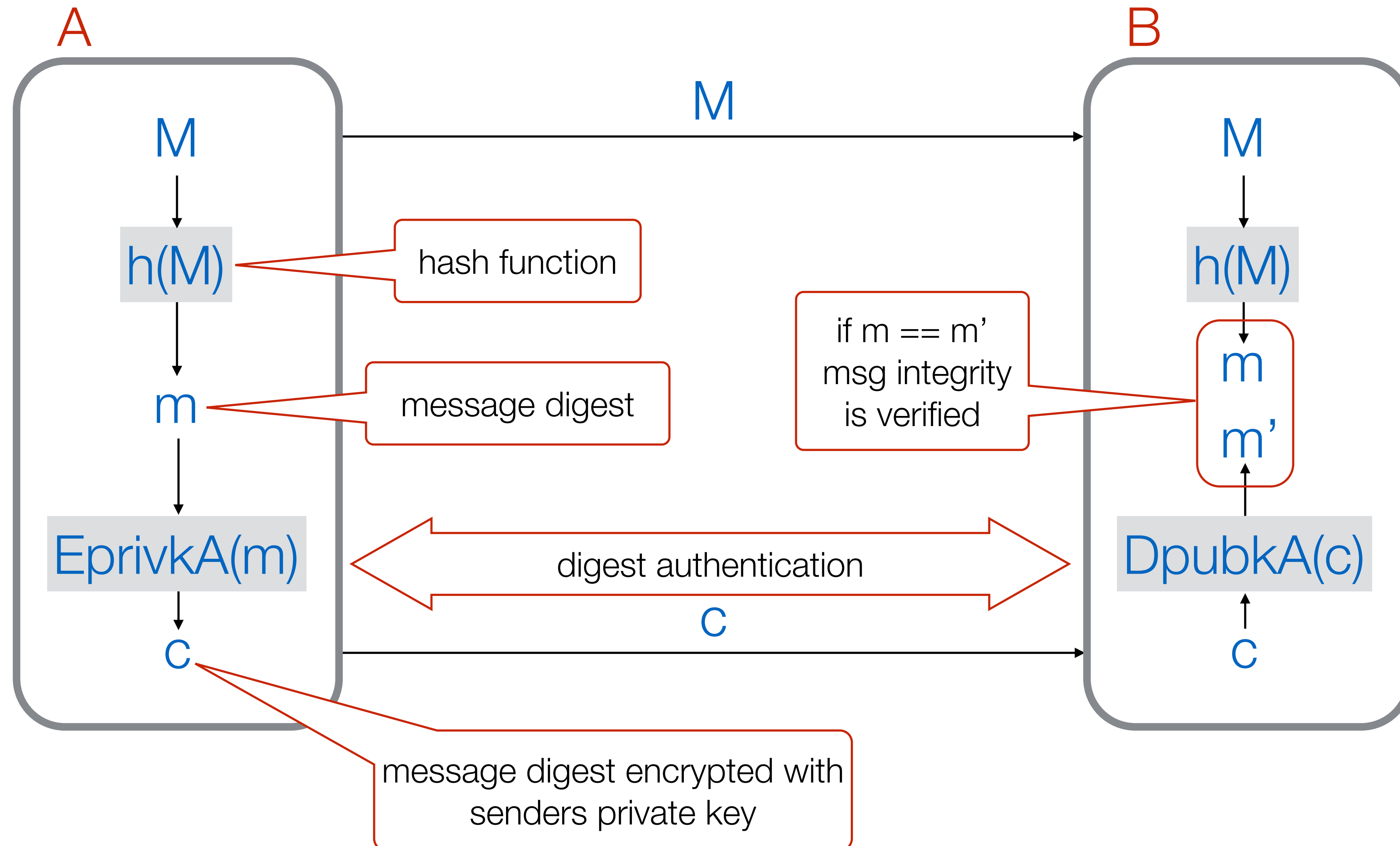
- need a **cryptographic** hash function
- need a secure method for public key distribution

Cryptographic hash function

$$H(M) \rightarrow m$$

- ▶ Even the smallest change in message M results in a significant “unpredictable” change of the hash value m
- ▶ It is infeasible to find two distinct messages, M_1 and M_2 , with the same hash value: $H(M_1) = H(M_2)$
- ▶ It is infeasible to modify a message M to get M' in a way that the hash value remains the same: $H(M) = H(M')$
- ▶ ... must be easy to compute
 - M is arbitrarily large, m is small and fixed-size (100's of bits)

Message Integrity



Cryptographic hash functions

- ▶ **MD5** - Message Digest Algorithm
 - 1992, R. Rivest, digest size 128 bits
- ▶ **SHA-1** - Secure Hash Algorithm
 - 1995, NSA, digest size 160 bits
- ▶ **SHA-2**
 - 2001 NSA designed, variants SHA-256 and SHA-512
- ▶ **SHA-3**
 - 2012 public competition winner, 2015 NIST-designated hashing standard (SHAKE variant allows arbitrary output length)

Certificates

- ▶ Solving the public key distribution problem
- ▶ Shared trust (having somebody's public key) helps:
 - When both A and B trust C
 - $\Rightarrow$ A can establish trust with B (and vice versa)
- ▶ Where to start?
 - who to trust
 - how is the initial trust established
- ▶ Solution: **Certificate Authority** (CA)

Certificates

- ▶ **Goal:** A wants to prove its identity to B
- ▶ **Given:** A and B trust CA (both have CA's public key)
- ▶ Broad approach: **Public key certificate**
 - A's public key encrypted with CA's private key (ensures integrity of the key)
 - ... plus additional information
- ▶ **Use:** A presents its certificate when initiating communication with B

Certificates - Questions

- ▶ **Man in the Middle Attack:** How does B know that it is A's certificate and not an impostor's one?
 - Include A's identification (hostname, human-readable info)
- ▶ **Replay Attack:** Attacker overhears/requests A certificate and presents it when pretending to be A
 - Use nonce encrypted with A's private key during communication
- ▶ **Compromised certificate:** Either A's or CA's private keys are compromised
 - Limited validity and certificate revocation

Certificates - Issuance

- ▶ **A** gets a certificate from a **CA**
 - **A** generates public/private key pair
 - **A** generates a Certificate Signing Request (**CSR**)
 - **CA** (hopefully) makes sure that it interacts with **A** and not an impostor
 - **CA** encrypts the certificate (public key + **A**'s identification + ...) with its own private key
 - certificate is delivered to **A** (can be done securely since **A** has **CA**'s public key)

Certificates - Use

- ▶ **B** authenticates **A**
 - **B** requests a certificate from **A** together with a nonce
 - **A** sends back the certificate together with the nonce encrypted with **A**'s private key
 - **B** decrypts the certificate with **CA**'s public key and verifies that it was issued to **A**
 - **B** decrypts the nonce using **A**'s public key and verifies that the value matches the value sent