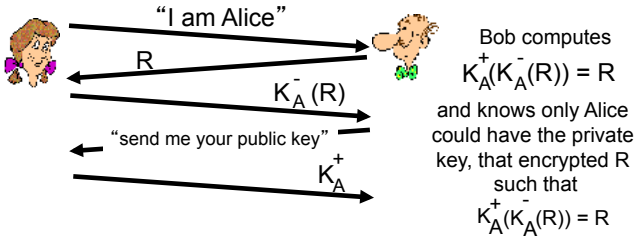
Network Security (II)

Public Key Exchange Problem

- Everything hinges on A getting B's public key …
  - once that's done, all is set
- Everyone can get B's public key, so the intruder can pretend to be A, uses B's public key and standard encryption algorithm to encode messages, and send the cipher to B/
  - Alice or anyone claiming to be Alice can get the public key and send the message.
- Need:
  - authentication

Authentication

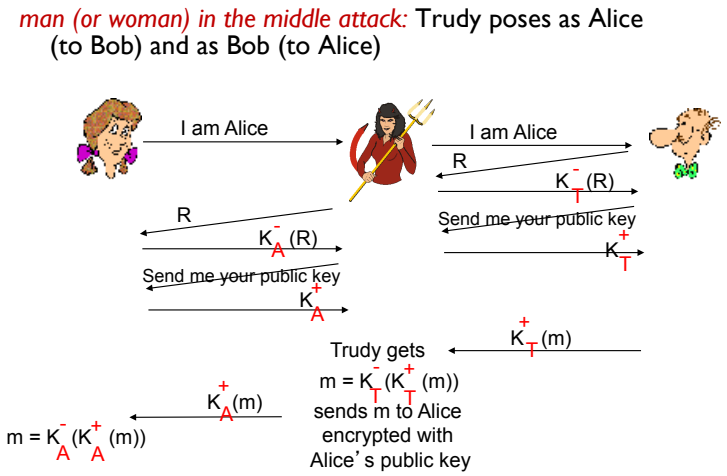
- Basic idea:
  - B verifies A by using public/private key cryptography
  - based on the idea that only possessor of private key could have encrypted something that decrypts using its public key



- Question: How could Trudy impersonate Alice without knowing Alice's private key? [Record Alice's signed message and replay it.]

- Problem: Replay Attack
  - replay attack: the intruder need only eavesdrop on A's communication, record the encrypted version of A's identification message ($K_A^-(R)$), and play back the encrypted version of the message to B to pretend he/she is A
  - solution: use of a nonce
    - number (R) used only once-in-a-lifetime
    - A use K_A^- to encrypt R sent by B ($K_A^-(R)$), B uses K_A^+ to decrypt and check R
    - Even if the intruder recorded ($K_A^-(R)$) and use it to pretend to be Alice later, B will know the number R is out-of-date after deception and know this Alice is a fake Alice.
    - ($K_A^-(R)$) digital signature
    - Once-in-a-lifetime may be too rigorous. A nonce need not be unique for the lifetime of the universe; it must satisfy the freshness and uniqueness requirements of the protocol. A sufficiently unpredictable challenge is appropriate here.

- Problem: Man-in-the middle attacks
  - Show the blow diagram but cover everything beyond R. Question: if Trudy intercepts Alice's request for Bob's public key, what would happen? [Trudy could intercept the message without letting Alice or Bob know.]



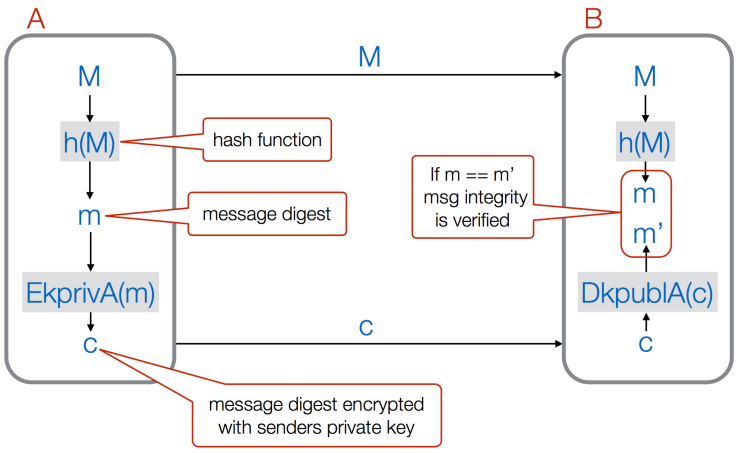
  - security hole: difficult to detect, all three people receive all messages
  - Question: Would strong encryption for message communication prevent this attach? [No. Strong encryption does not help if the key establishment or authentication has been compromised.]
  - Still unaddressed: We need a trusted way to obtain someone's public key

---

Message Integrity

- Basic idea:
  - with the assumption that there are multiple paths between A and B, messages can be delivered to B using different paths, and intruders won't intercept all paths.
  - use public/private key cryptography for a long message is computational expensive. Therefore, fixed-length, easy- to-compute digital "fingerprint" would be easier.
  - Use hash function to get a message digest, send encrypted (with senders private key — digital signature) message digest
  - if hash of the received message agrees with decrypted received message digest, it is assured that the message was not altered in transit
- Hash function properties:
  - many-to-one: cryptographic hash functions that accept arbitrarily long inputs and produce a fixed-length output are necessarily many-to-one. Because there are more possible input messages than output values, different messages must sometimes produce the same hash. If two different messages have the same digest, it's called hash collision. Hash collision is unavoidable. What matters is how difficult it is to find collisions.
  - fixed-length, easy- to-compute digital "fingerprint"

- given message digest *x*, computationally infeasible to find *m* such that *x = H(m)*



- Problems:
  - need a cryptographic hash function
  - need a public key distribution method

Cryptographic Hash Function

- MD5 - Message Digest Algorithm
  - 1992, R. Rivest, digest size 128 bits
  - widely used (RFC 1321)
  - `echo "hello world" | md5`
  - `echo "hello world" | openssl md5`
- SHA-1 - Secure Hash Algorithm
  - 1995, NSA, digest size 160 bits
  - SHA-2, SHA-3 competition at NIST
  - `echo "hello world" | openssl sha1`
  - `echo "hello world" | openssl sha256`
  - `echo "Hello World" | openssl sha256`
    - `openssl`: multi-platform, opensource SSL/TLS toolkit
- Cryptographic hash functions are designed to have an unpredictable-looking change in the digest even if a small input change.
- SHA-256, 256-bit output, considered more secure against practical collision attacks, while MD5 and SHA-1 are unsuitable for modern collision-resistant digital-signature applications.

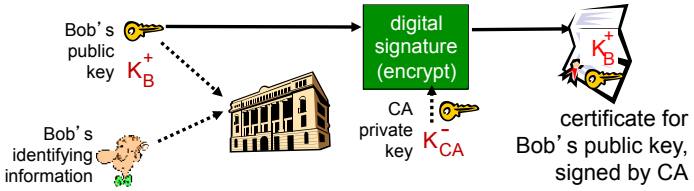
Certificates

- Solving the public key distribution problem
- Trust (having somebody's public key) is transitive
  - A trusts C and B trusts C ⇒ A can establish trust with B

- Where to start?
  - who to trust?
  - how is the initial trust established?
- Solution: Certificate Authority (CA)

- Goal: A wants to prove its identity to B
- Given: A and B trust CA and both have CA's public key
- Use: A presents its certificate when initiating communication with B
- So, what is a certificate and how to use it?

- A digital certificate binds A's public key to additional identifying information (e.g., A's identification) and is signed by a CA (aka encrypted with CA's private key).
- The digital certificate is publicly available and doesn't include the corresponding private key.



Certificates - Issuance

- If A gets a certificate from a CA:
  - A generates public/private key pair
  - A generates Certificate Signing Request (CSR) and sends it to CA
  - 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
  - The certificate is delivered to A (can be done securely since A has CA's public key and CA is trusted by A)

Certificates - Use

- If B needs to authenticate A
  - B sends a request to A for A's certificate coupled with a nonce
  - A sends back its certificate and the encrypted nonce using A's private key
  - B decrypts the certificate with CA's public key (as the certificate is encrypted using CA's private key, and B trusts CA) and verifies that it was issued to A
  - B decrypts the encrypted nonce using A's public key and verifies that the nonce matches the original nonce sent to A.
  - If the decrypted nonce matches the original value, B can tell A is the real A and can use A's public key to establish a secure communication (e.g., exchange the symmetric key).

Inspect Certificate Information

- We can use a web browser to inspect the information of a certificate of a website running over HTTPS. We can also use `openssl` to inspect the information too.

- `openssl s_client -connect www.google.com:443 | openssl x509 -text -noout`
  - `openssl s_client -connect <hostname>:<port>` test connectivity to a server.
  - `openssl x509` certificate display and signing utility
  - `-text` outputs the certificate's in the text format.
  - `-noout` prevents output the encoded version

Field	Question	Answer from the Output
Subject Alternative Name (SAN)	Which hostnames does this certificate cover?	DNS:www.google.com — the certificate covers www.google.com.
Issuer	Who issued this certificate?	Google Trust Services (WE2) — an intermediate Certificate Authority.
Validity	When does it become valid and expire?	Valid from September 28, 2026, 18:41:21 UTC to December 21, 2026, 18:41:20 UTC.
Public Key Algorithm	Is the key RSA or elliptic-curve-based?	Elliptic-curve cryptography (ECC), not RSA. It uses a 256-bit public key on the NIST P-256 curve (prime256v1).
Signature Algorithm	Which algorithm did the issuer use to sign it?	ECDSA with SHA-256 (ecdsa-with-SHA256). The issuer uses ECDSA to sign the certificate, with SHA-256 as the hash function.
Authority Information Access	Where might the issuer's certificate be obtained?	http://i.pki.goog/we2.crt — a URL where the WE2 intermediate CA certificate can be obtained.

Disclaimer: Notes adapted from the textbook and online resources.