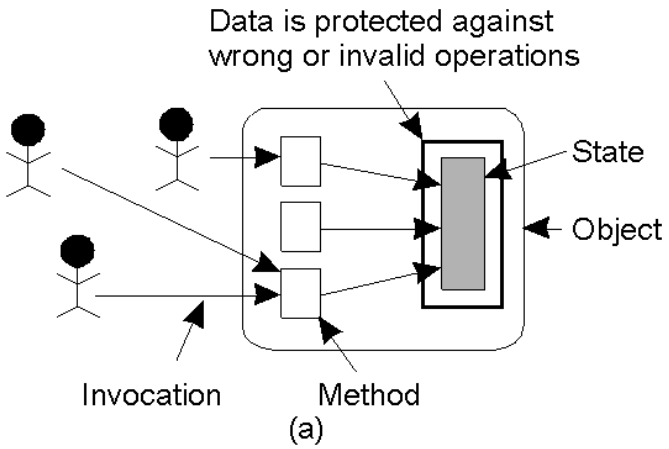
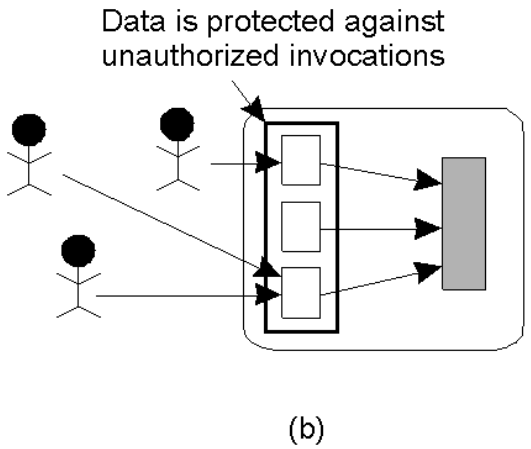
Security: Different Approaches



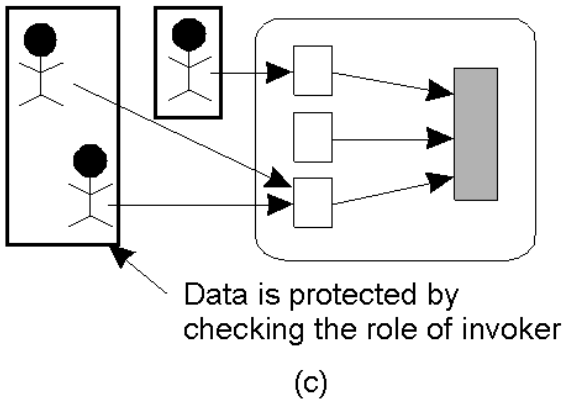


- Three approaches for protection against security threats

a) Protection against invalid operations

b) Protection against unauthorized invocations

c) Protection against unauthorized users



Authentication

- **Question:** how does a receiver know that remote communicating entity is who it is claimed to be?

Authentication Protocol (AP)

- **AP 1.0**
  - Alice to Bob: "I am Alice"
  - Problem: intruder "Trudy" can also send such a message
- **AP 2.0**
  - Authenticate source IP address is from Alice's machine
  - Problem: IP Spoofing (send IP packets with false src address)
- **AP 3.0**: use a secret password
  - Alice to Bob: "I am Alice, here is my password" (e.g., telnet)
  - Problem: Trudy can intercept Alice's password by sniffing packets

Authentication Protocol

AP 3.1: Use encryption

Use a symmetric key known to Alice and Bob

- Alice & Bob (only) know secure key for encryption/decryption

```
A to B: msg = encrypt("I am A")
B computes: if decrypt(msg)=="I am A"
    then A is verified
    else A is fradulent
```

Failure scenarios: *replay* attack (also called *man-in-the-middle*)

- Trudy can intercept Alice's message and masquerade as Alice at a later time

Authentication Using Nonces

Problem with AP 3.1: same password is used for all sessions

AP 4.0: use a sequence of passwords

pick a "once-in-a-lifetime-only" number (nonce) for each session

```
A to B: msg = "I am A"  /* note: unencrypted message! */
B to A: once-in-a-lifetime value, n
A to B: msg2 = encrypt(n) /* use symmetric keys */
B computes: if decrypt(msg2)==n
            then A is verified
            else A is fradulent
```

- Similarities to three way handshake and initial sequence number choice
- Problems with nonces?

Authentication Using Public Keys

AP 4.0 uses symmetric keys for authentication

AP 5.0: Use public keys

```
A to B: msg = "I am A"
B to A: once-in-a-lifetime value, n
A to B: msg2 = PrK_A(n)
B computes: if PubK_A (PrK_A (n))== n
                    then A is verified
                    else A is fradulent
```

Problems with Ap 5.0

- Bob needs Alice's public key $PubK_A$ for authentication
  - Trudy can impersonate as Alice to Bob

```
Trudy to Bob: msg = "I am Alice"
Bob to Alice: nonce n (Trudy intercepts this message)
Trudy to Bob:  msg2= PrK_T(n)
Bob to Alice: send me your public key (Trudy intercepts)
Trudy to Bob: send PubK_T (claiming it is PubK_A)
Bob: verify PubK_T(PrK_T(n)) == n  and authenticates Trudy as Alice!!
```

- AP 5.0 is only as "secure" as public key distribution
  - We will discuss how to make PubK distribution secure

Man-in-the-middle Attack

- Trudy impersonates as Alice to Bob and as Bob to Alice
  - Alice Trudy Bob
  - "I am A" "I am A"
  - nonce n
  - DT(n)
  - send me ET
  - ET
  - nonce n
  - DA(n)
  - send me EA
  - EA
  - Bob sends data using ET, Trudy decrypts and forwards it using EA!! (Trudy *transparently* intercepts every message)

Digital Signatures Using Public Keys

Goals of digital signatures

- *Authentication:* receiver knows that msg is from sender
- *Integrity:* channel cannot tamper msg
- *Non-repudiation:* sender cannot repudiate message never sent ("I never sent that")

Suppose A wants B to "sign" a message M

```
B sends PrK_B(M) to A
A computes:
  if PubK_B ( PrK_B(M)) == M then B has signed M
```

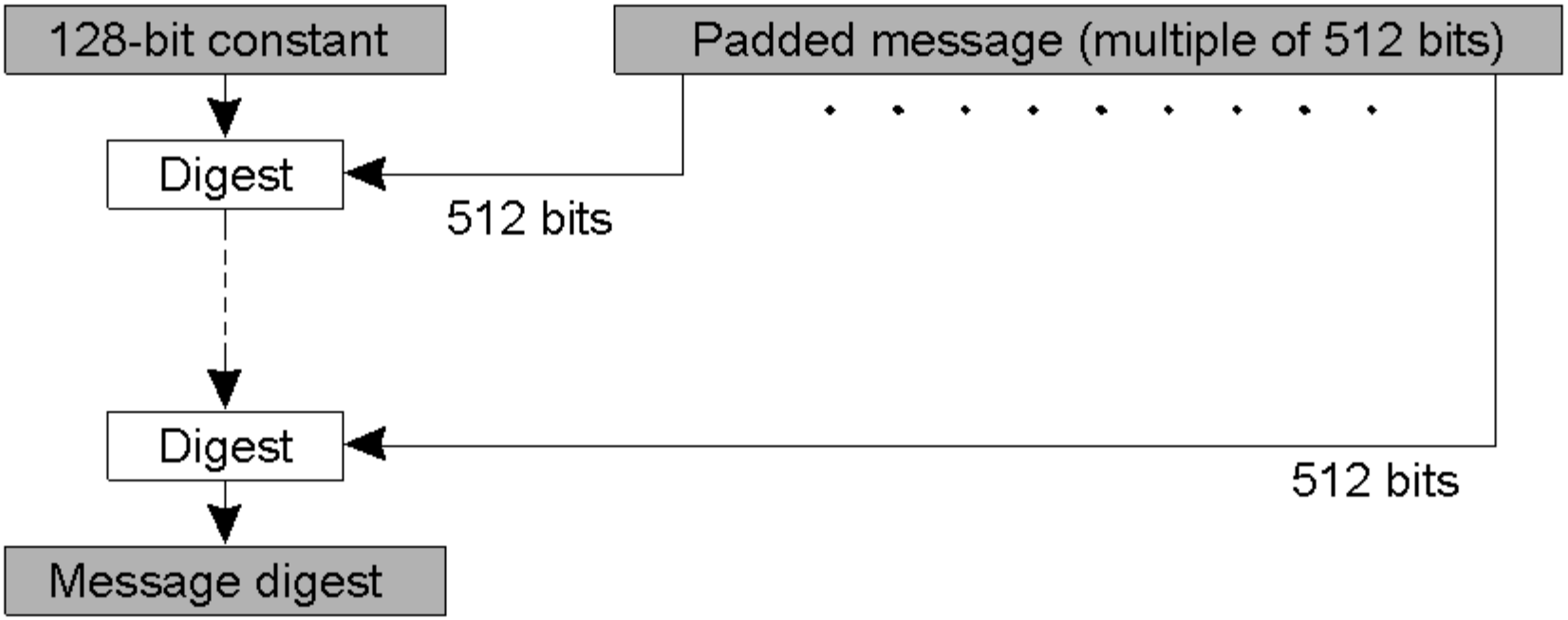
Q: can B plausibly deny having sent M?

Message Digests

- Encrypting and decrypting entire messages using digital signatures is computationally expensive
  - Routers routinely exchange data that could be corrupted
  - They use checksums to detect errors
- Message digests: similar to a checksum but secure
  - Hash function H: converts variable length string to fixed length hash
  - Digitally sign H(M)
  - Send M, PrK(H(m))
  - **Q:** how can the received authenticate?
- Property of a *cryptographically secure* hash function H
  - Given a digest x, it is infeasible to find a message y such that H(y) = x
  - It is infeasible to find any two messages x and y such that H(x) = H(y)
  - **Q:** why are these properties needed?

Hash Functions : MD5

- The structure of MD5



Hash Functions

- MD5 not secure any more

- SHA hash functions (SHA = Secure Hash Algorithm)
  - SHA-1 : 160-bit function that resembles MD5
  - SHA-2: family of two hash functions (SHA-256 and SHA-512)
  - Developed by NIST and NSA

Symmetric Key Exchange

Problem: how do distributed entities agree on a key?

- Pairwise symmetric keys don't scale

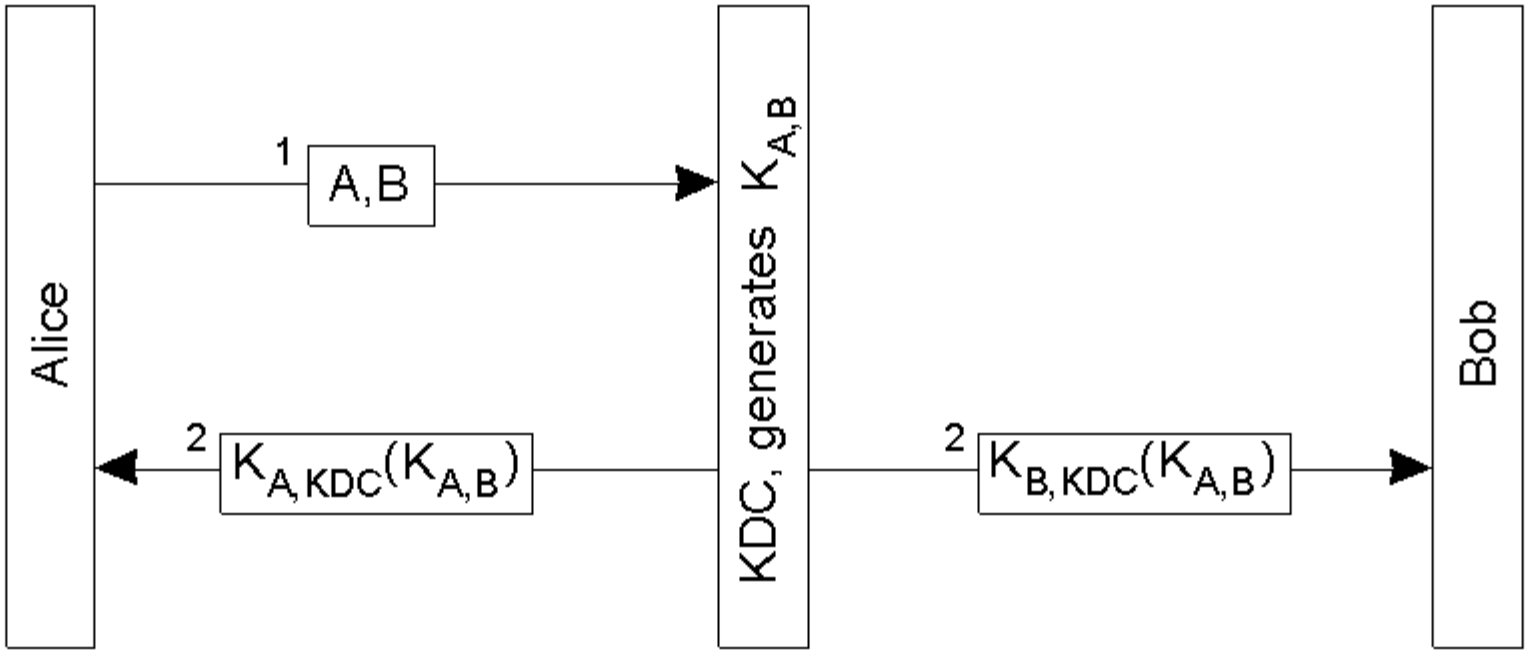
Assume: each entity has its own *single* key, which only it and a trusted *key distribution center (KDC)* server know

KDC server:

- Will *generate* a *one-time session* key that A and B use to encrypt communication
- Will use A and B's single keys to communicate session key to A, B
- Example: Kerberos

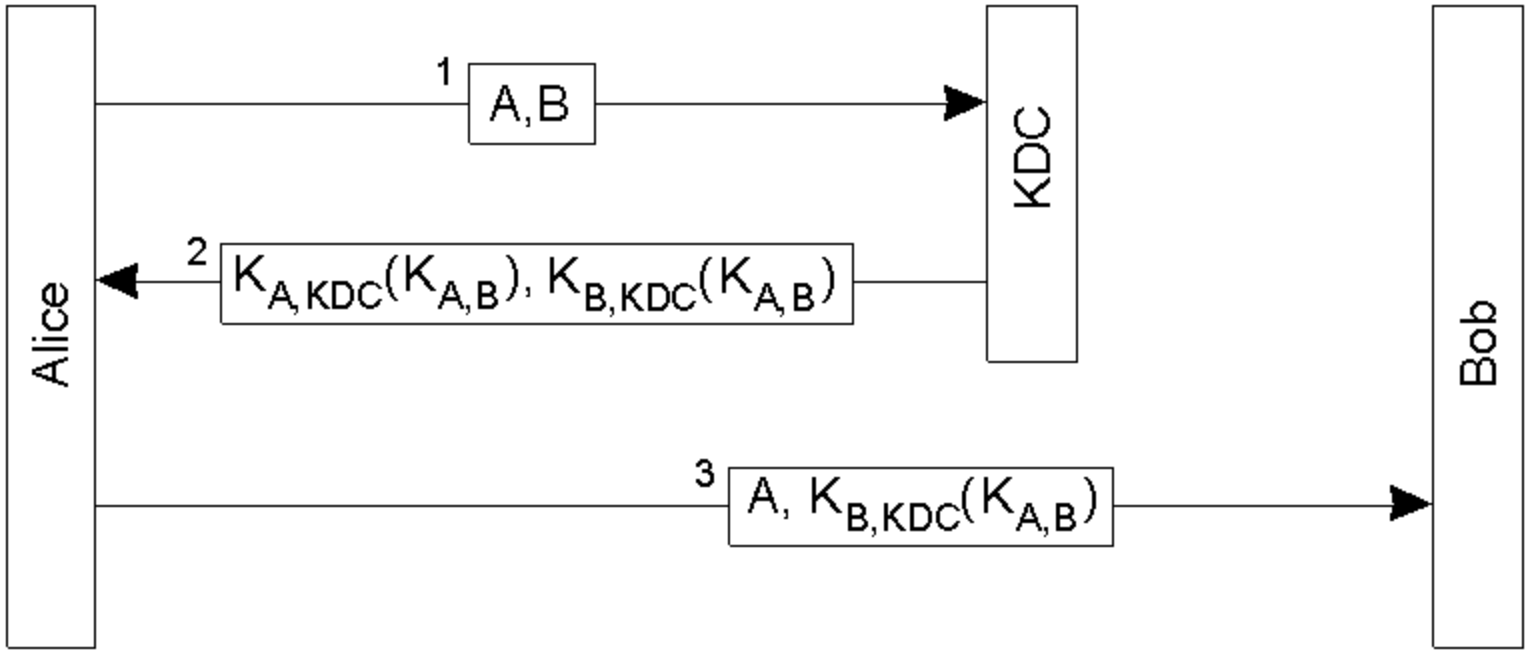
Key Exchange: Key Distribution Center

- $K_{X,Y}$ is the secret key between X and Y



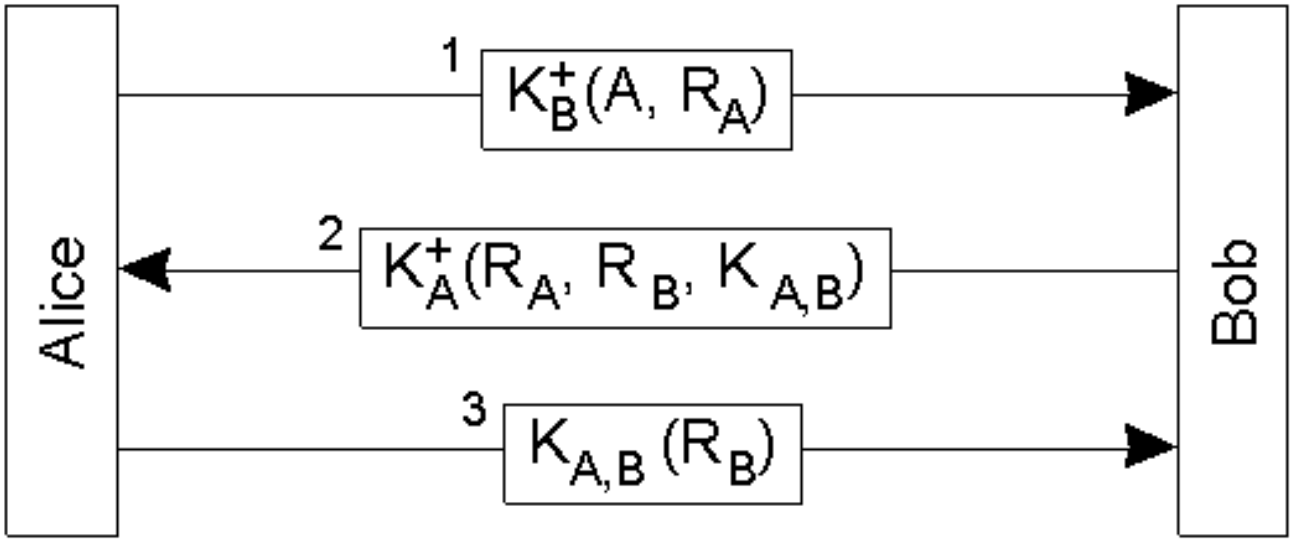
Authentication Using a Key Distribution Center

- Using a ticket and letting Alice set up a connection to Bob.



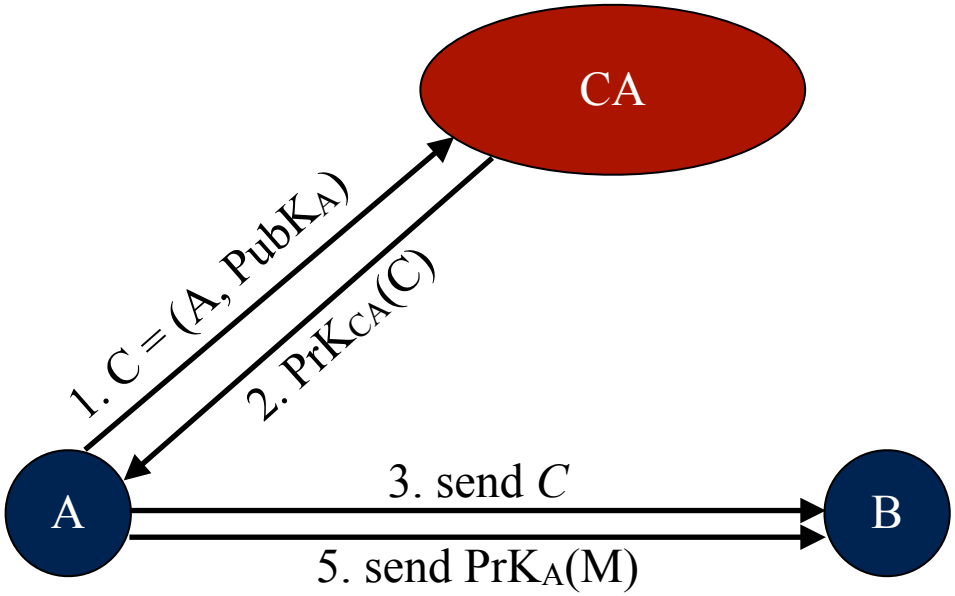
Mutual Authentication Using Public Keys

- K_X^+: public key of X
- $K_X^+(M)$: encryption of M that can only be read by B
- R_X: *challenge* used to authenticate sender
- **Q:** Why exchange symmetric keys if we have public keys?
- **Q:** What if there was no challenge?



Certification Authority

- Having keys for all clients does not scale
- Solution: distribute public keys while avoiding impersonation
  - Remember Trudy's man-in-the-middle attack with AP 5.0
- Certification Authority acts as a trusted third party
  - Everyone knows its public key
  - Issues (and revokes) *certificates:* (server ID, public key) pairs



1. A sends *certificate C*
2. CA signs it and returns it
3. A sends signed certificate to B
4. B uses PubK$_{CA}$ to validate signature and concludes that PubK$_A$ comes from A
5. A sends encrypted M
6. B uses PubK$_A$ to decrypt

Security in Enterprises

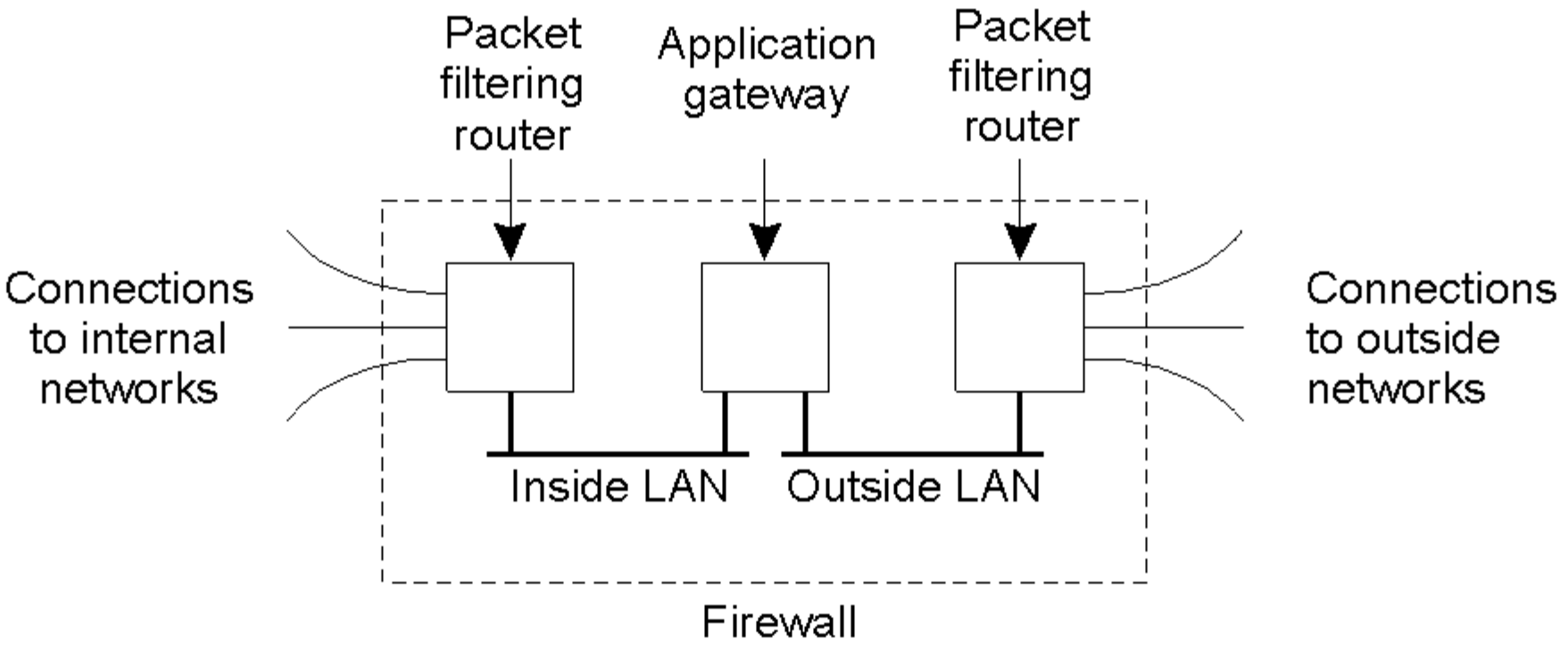
- Multi-layered approach to security in modern enterprises
  - Security functionality spread across multiple entities
- Firewalls (policies + ports)
- Deep Packet inspection
- Virus and email scanners
- VLANs
- Network radius servers
- Securing WiFi
- VPNs
- Securing services using SSL, cerificates, kerberos

Security in Internet Services

- Websites
  - SSL + authentication + captchas
- Challenge-response authentication
  - paypal
- Two factor authentication
  - Gmail: password + mobile phone
- One-time passwords
  - Hotmail one-time password
- Online merchant payments: paypal, amazon payments, google checkouts

Protection Against Intruders: Firewalls

- A common implementation of a firewall.



Firewalls

Firewall: network components (host/router+software) sitting between inside ("us") and outside ("them)

Packet filtering firewalls: drop packets on basis of message *header* (i.e., IP address, port)

Application gateways: application-specific code inspecting message *content*

- E.g., email or telnet gateways
-

Secure Email

- Requirements:
  - Secrecy
  - Sender authentication
  - Message integrity
  - Receiver authentication
- Secrecy (nobody can read the message)
  - Public keys to encrypt messages?
    - Inefficient for long messages
  - Use symmetric keys
    - Alice generates a symmetric key K
    - Encrypt message M with K
    - Encrypt K with $PubK_B$
    - Send (K(M), $PubK_B$(K))
    - Bob decrypts using PrK_B, gets K, decrypts K(M)

Secure Email

- Authentication and Integrity (msg comes from Alice)
  - Alice applies hash function H to M (H can be MD5 or SHA)
  - Creates a digital signature $PrK_A(H(M))$
  - Send $(M, PrK_A(H(M)))$ to Bob
- Putting it all together: *authenticate* then *encrypt*
  - Compute $H(M)$, $PrK_A(H(M))$
  - $M' = (M, PrK_A(H(M)))$
  - Generate symmetric key K, compute $K(M')$
  - Encrypt K as $PubK_B(K)$
  - Send $(K(M'), PubK_B(K))$
- Used in PGP (Pretty Good Privacy)

Secure Sockets Layer (SSL)

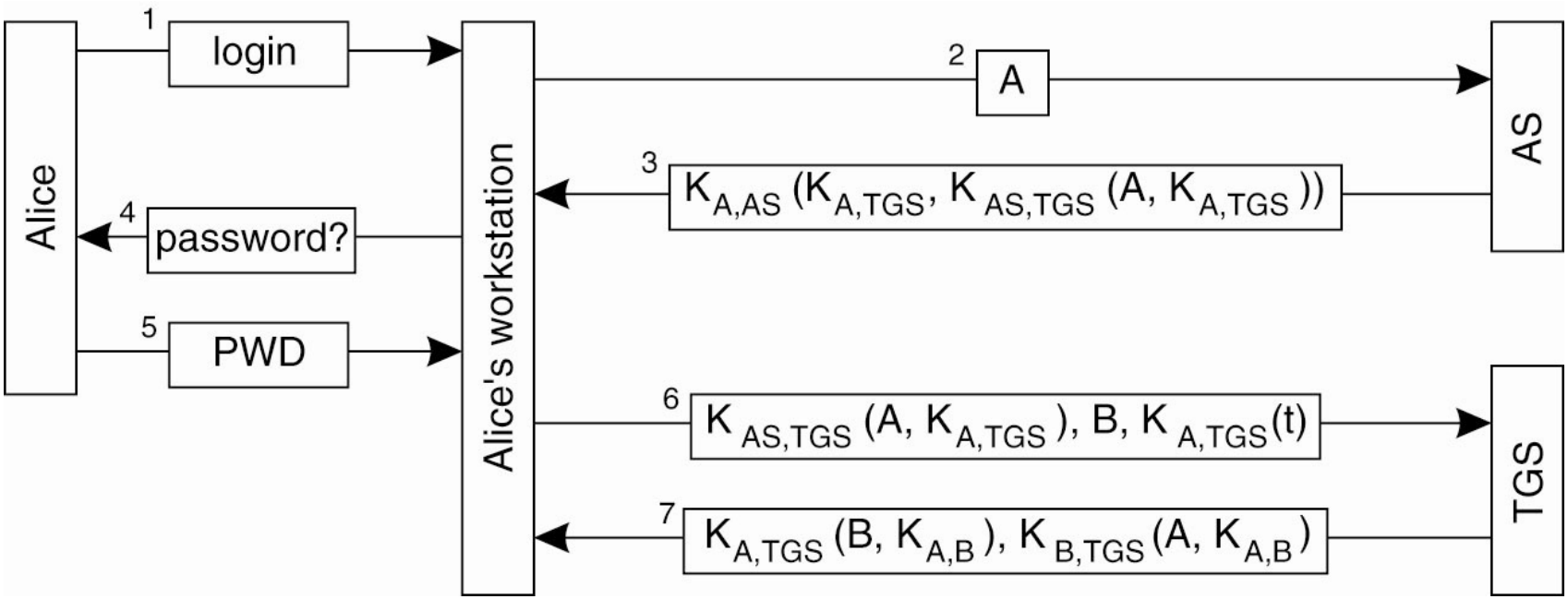
- SSL: Developed by Netscape
  - Provides data encryption and authentication between web server and client
  - SSL lies above the transport layer
  - Useful for Internet Commerce, secure mail access (IMAP)
  - Features:
    - SSL server/client authentication (using certificates)
    - Encrypted SSL session

Secure Socket Layer (https)

- Opening session between browser B and server S
  - B -> S: SSL version and preferences @ B
  - S->B: SSL version, preferences, and certificate @ S
  - B: uses its list of CAs and public keys to decrypt $PubK_S$
  - B->S: generate K, encrypt K with with $PubK_S$
  - B->S: "future messages will be encrypted", and K(m)
  - S->B: "future messages will be encrypted", and K(m)
  - SSL session begins…

Example: Kerberos (1)

- Assist clients in setting up secure channel with a server
- Auth Server (AS) provides login service
- Ticket granting service (TGS) sets up secure channel
  - Tickets are used to convince the server of the authenticity of the client
    - Single signon: no need to auth to other servers separately



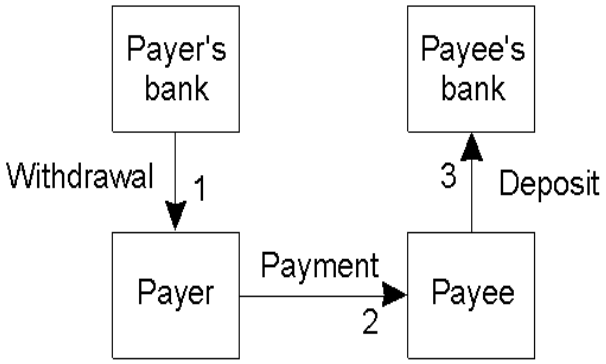
Electronic Payment Systems

- Payment systems based on direct payment between customer and merchant.

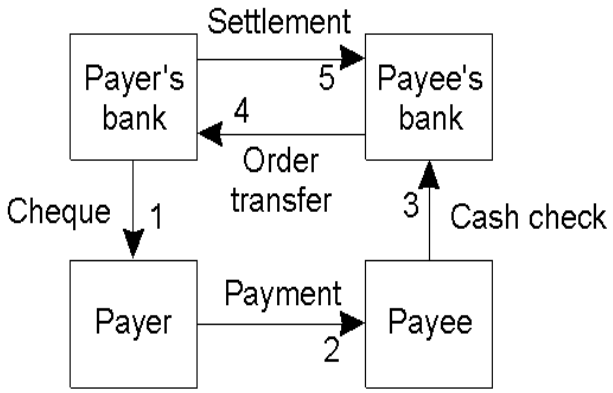
a) Paying in cash.

b) Using a check.

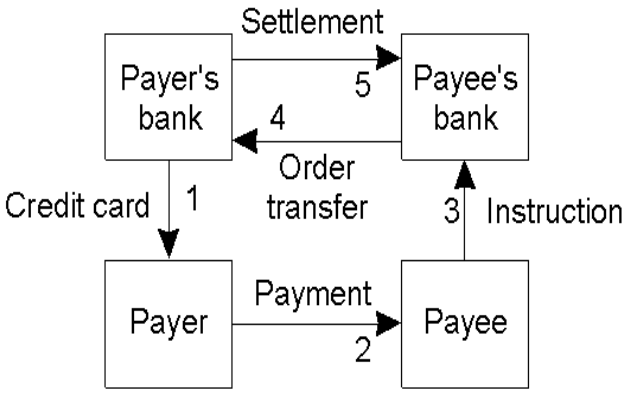
c) Using a credit card.



(a)



(b)



(c)

BitCoin

- Digital currency: P2P electronic cash, Decentralized
  - Open source crypto protocol
  - Satoshi Nakamoto

- New coins made by bitcoin servers
  - Expend resources to generate a coin
  - Decreasing rate of coin generation
- Uses digital signatures to pay to "public keys"

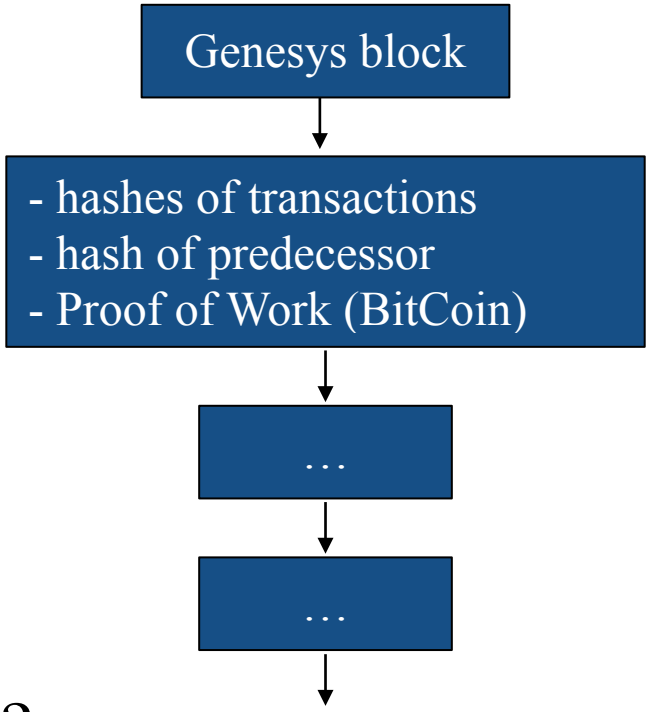
- Bitcoin blockchain: distributed transaction ledger

Blockchain: Distributed Ledger

- Blockchain: distributed public ledger of transactions
  - Lists all financial transactions, distributed DB
  - Generic protocol for transactions based on public key cryptography
- **Applications**: stock register, land transactions, marriage records, logistics, smart contracts
- **Sign** a transaction with private key and insert in the ledger
- Every block contains multiple transactions
- Massively duplicated; shared using **P2P** file transfer protocol
- Updated by special nodes "miners" to append blocks
  - Creating new block is expensive: rewarded with coins
- All nodes perform validation and clearing

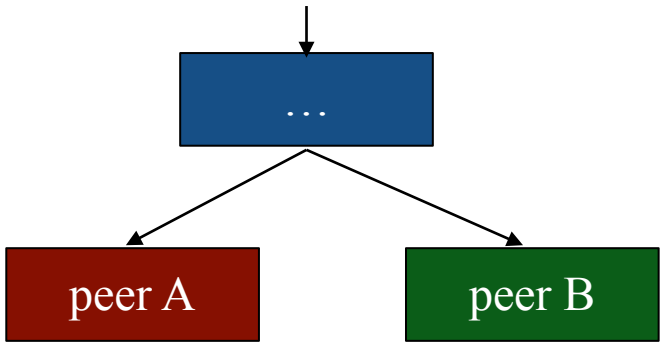
Blockchain

- Used to authenticate and order transactions
  - Chain of blocks authenticating transactions
  - Replicated at all peers
  - Everyone can verify
- Open P2P, no access control
  - Everyone can install client, replicate chain, append to it
- **Q:** When is a transaction committed?



Forks and Proof of Work

- Fork: different peers can append concurrently
  - Either on purpose of because they are not aware of each other
- Different heuristics to merge a fork
  - Longest branch wins -> orphan blocks ignored, double-spending
- Bitcoin's Proof of Work is a way to limit forks
  - Slows down the generation of blocks to limit chances of forks
  - Also provides incentives to extend the chain (block = coin)
- **Q:** when is a transaction committed?
- **Q:** looks familiar?



Permissioned Blockchains

- Bitcoin is eventually consistent: it does not execute consensus
  - Would be unfeasible in an open P2P system
- Permissioned blockhains
  - Access control to limit membership of the blockchain
  - Run Byzantine-fault-tolerant consensus among members
- Based on PBFT protocol, which has similarities to Paxos
  - 3f+1 replicas, always safe up to f failure, live with synchrony
  - Leader-based
  - View change protocol analogous to read phase of Paxos