CMPSCI 677 Operating Systems **Spring 2017**

Lecture 25: April 30

Lecturer: Prashant Shenoy *Scribe:* ***Daniel Saunders***

25.1 Distributed Security

25.1.1 Authentication using public keys

AP 4.0 uses symmetric keys for authenticantion. **Question**: can we use public keys? *symmetry*: DA(EA(n)) = EA(DA(n)).

AP 5.0:

A to B: msg = "I am A"
B to A: once in a lifetime value n
A to B: msg = DA(n)
B computes: If EA(DA(n)) = n
then A is verified
else A is fradulent

25.1.2 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
	nonce n	
	DA(n)	
	send me EA	
	EA	

Bob sends data using ET, and Trudy decrypts and forwards it using EA (Trudy *transparently* intercepts every message).

25.1.3 Digital signatures using public keys

Goals of digital signatures:

- Sender cannot repudiate message never sent ("I never sent that").

- Receiver cannot fake a received message.

Suppose A wants B to "sign" a message M.

B send DB(M) to A
A computes if EB(DM(A)) = M
then B has signed M

Question: Can B plausibly deny having sent M?

25.1.4 Message digests

Encypting and decrypting entire messages using digital signatures is computationally expensive. Routers routinely exchange data, which do not need encryption, but do require authentication and to verify that data hasn't changed.

A message digest is a compact summary of a message, like a checksum. A has function H converts a variable length string to a fixed length hash value. The user will then digitally sign $H(M)$, and send both M and $DA(H(M))$. The receiver can verify who sent the message and that it has been changed.

Important property of H: Given a digest x, it is infeasible to find a message y for which $H(y) = x$. Also, it is infeasible to find any two messages such that $H(x) = H(y)$ (hash collision). If the hash function H satisfies this property, then this scheme is much more efficient than digital signatures with public keys: we need only encrypt the fixed-length hash value $H(M)$, typically much shorter than M.

25.1.5 Hash functions: MD5

MD5 takes an arbitrarily-sized objects, splits it into smaller chunks, and hash each of the chunks. This method is recursed until we have a fixed-sized hash value.

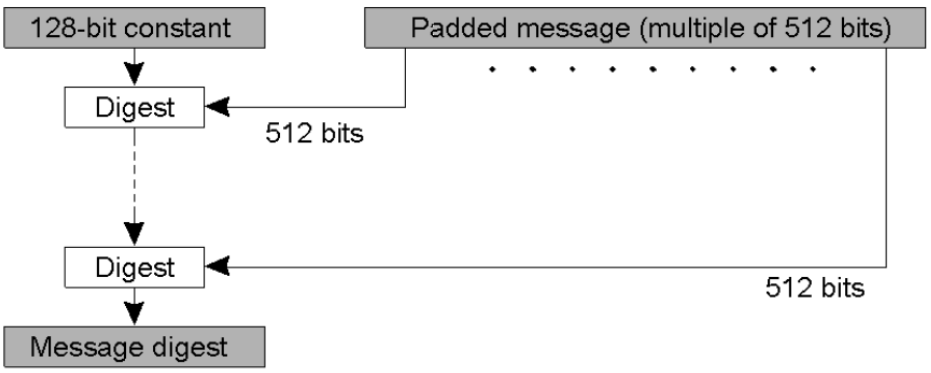

Figure 25.1: The structure of MD5.

25.1.6 Hash functions: SHA

MD5 is not secure anymore. Secure Hash Algorithms (SHA) hash functions:

- 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.

Let's say you have a hashed value h, and you want to know the original message m such that $H(m) = h$. How can we find out m? We could carry out a dictionary attack, in which we try all m' in some "dictionary" and compute $H(m')$. If any such m' produces $H(m') = h$, then $m' = m$. Clearly, the longer the message, the more time required by this brute-force attack.

25.1.7 Symmetric key exchange: trusted server

Problem: How do distributed entities agree on a key?

This is the problem of key distribution. If keys are comprised, than anyone may use your key to decrypt your messages. We need a secure method of key exchange; it must be just as strong as your encryption algorithm itself.

Assume: Each entity has its own single key, which only it and a trusted server know.

Server:

- Will generate a one-time session key (symmetric) that A and B use to encrypt all communication.
- Will use A and B's single keys to communicate session key to A and B.

25.1.8 Key exchange: key distribution center

Trusted third party as the key distribution center. Alice sends a message to the KDC saying "I'm Alice, and I want to communicate with Bob; please generate a key". The KDC generates a random session key, encrypts it with Alice's public key, and sends it back to Alice. The KDC will send the same key to Bob, encryped with Bob's public key.

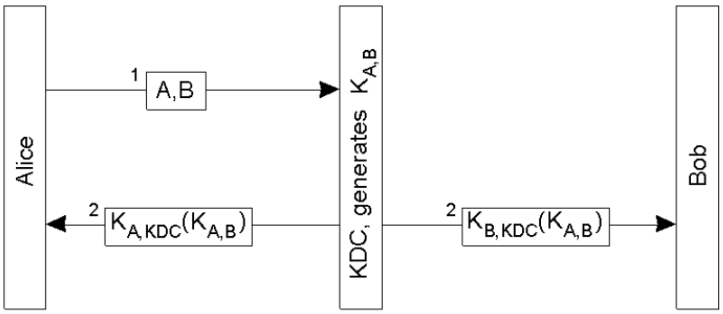

Figure 25.2: The principles of using a KDC.

25.1.9 Authentication using a key distribution center

Pictured below: Same technique as before, but the KDC does not actually send a message to Bob. Messages are sent from Alice to Bob.

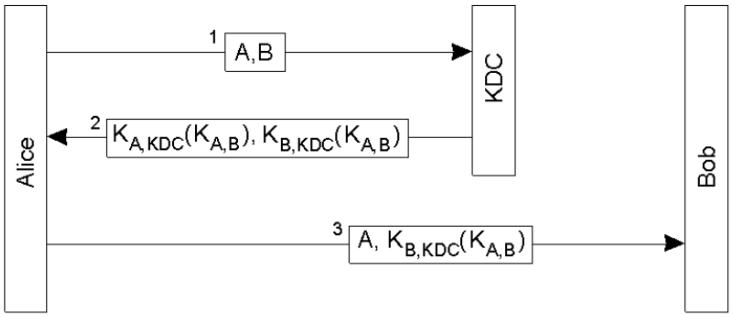

Figure 25.3: Using a ticket and letting Alice set up a connection to Bob.

Pictured below: The Needham-Schroeder authentication protocol uses a session key as well as a nonce (once-in-a-lifetime integer) for Alice to talk to the KDC. A second nonce is generated by Alice to begin communicating with Bob. Bob will generate a nonce as well and send it and the nonce from Alice back to Alice. The nonces are used as an extra validation step.

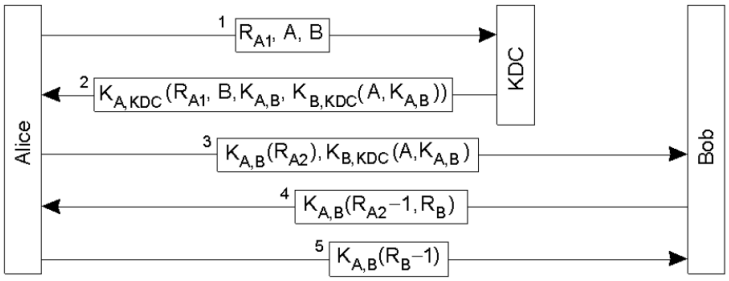

Figure 25.4: The Needham-Schroeder authentication protocol.

25.1.10 Public key exchange

There is no KDC in this protocol; public and private keys are used only. This protocol assumes we have a secure way of getting someone's public key before communicating with them. Alice sends a nonce to Bob, which is encrypted by Alice with Bob's public key. Bob decrypts it, sends it back, sends a new nonce and session key back (all of this is encrypted with Alice's public key). Alice decrypts this message, and sends back Bob's generated nonce for validation purposes.

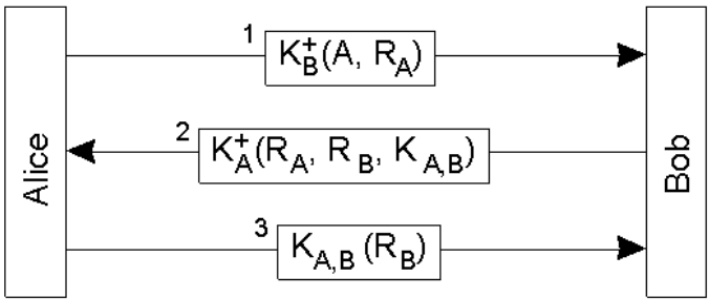

Figure 25.5: Mutual authentication in a public-key cryptosystem.

Public key retrieval is subject to man-in-the-middle attacks if public keys aren't shared securely. We can use a trusted third party in order to distributed public keys securely. This third party will issue "certificates", public keys of servers which are signed by the trusted third party. All servers have the trusted server's encryption key. Suppose that A wants to send B a message using B's "public" key; A can accomplish this by using certificates from the trusted third party. Certificates may be revoked.

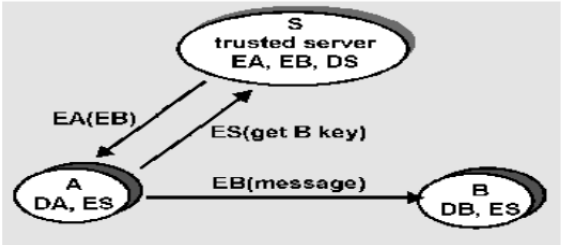

Figure 25.6: Public key exchange: trusted server.

25.1.11 Security in enterprises

- In the multi-layered approach to security, Security functionality is spread across multiple components.
- Firewalls
- Deep packet inspection
- Virus and email scanning
- VLANS
- Network radius servers
- Securing WiFi
- VPNs
- Securing services using SSL, certificates, kerberos

Distributed systems security or network security are very complex problems. There are many layers of security protections that are deployed, and things get complicated fast. Companies typically get hacked because there is a small issue with one or more of the security items listed above.

25.1.12 Security in internet services

How can we secure internet services?

- Websites
  - Ensure all connections between browser and server are encrypted.
  - Authenticate user (username and password checks).
  - Use techniques like CAPTCHAs to prevent scripts from launching attacks.
- Challenge-response authentication
  - Push a one-time key to a user's phone to enter for further authentication.
- Two-factor authentication
  - Password and mobile phone (gmail).
- One-time passwords
- Online merchant payments: paypal, amazon payments, google checkouts

25.1.13 Firewalls

A firewall is a machine that sits on the boundary of the internal (e.g., home WiFi) and external networks (e.g., the Internet). All traffic between the internal and external networks must pass through the firewall, which has a pre-configured set of packet-filtering rules. If a packet matches one of the firewall's rules, it is allowed to pass; otherwise, it is dropped.

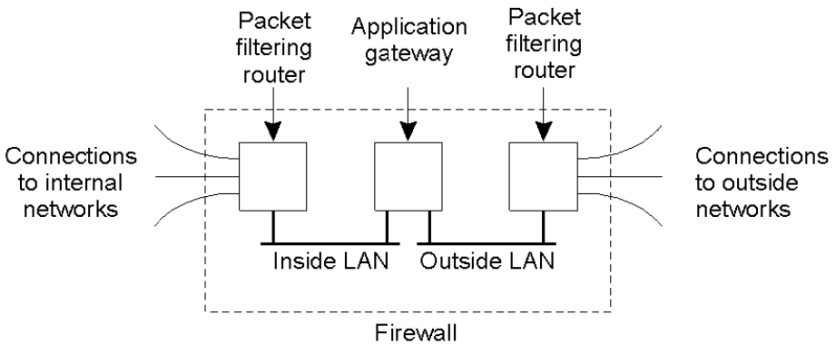

Figure 25.7: A common implementation of a firewall.

25.1.14 Secure email

- Requirements
  - Secrecy

  - Sender authentication
  - Message integrity
  - Receiver authentication

- Secrecy
  - Can use public keys to encrypt messages (this is inefficient for long messages because public keys are very long).
  - Use shorter, message-specific symmetric keys
    * Alice generates symmetric key K
    * Encrypt message M with K
    * Encrypt K with E_B
    * Send $K(M), E_B(K)$
    * Bob decrypts using his private key, gets K, and decrypts $K(M)$

- Authentication and Integrity (without secrecy)
  - Alice applies has function H to M (H can be MD5 or SHA)
  - Creates a digital signature $D_A(H(M))$
  - Send $M, D_A(H(M))$ to Bob

- Putting it all together
  - Compute $H(M), D_A(H(M))$
  - $M' = \{M, D_A(H(M))\}$
  - Generate symmetric key K, compute $K(M')$
  - Encrypt K as $E_B(K)$
  - Send $K(M'), E_B(K)$

- Used in PGP (pretty good privacy)

25.1.15 Secure sockets layer (SSL)

SSL (developed by NetScape) provides data encryption and authentication between web servers and clients. It lies above the transport layer. It is used for internet commerce, secure mail access (IMAP), and more. Features include: SLL server authentication, encrypted SSL sessions, and SSL client authentication.

It uses the HTTPS protocol instead of HTTP. The browsers sends the first message to the server saying it can support some version of SSL, and the server responds with its supported version of SSL, as well as a certificate (server's RSA public key encrypted by a trusted third party's (certification authority) private key). The browser generates a session key K, and encrypts the key with E_S, the public key of the server sent in the certificate. The browser sends "future messages will be encrypted" and $K(M)$ to the server, and the server responds with the same. The SSL session then begins.

25.2 Electronic payment systems

Payment systems based on direct payments between customer and merchant.

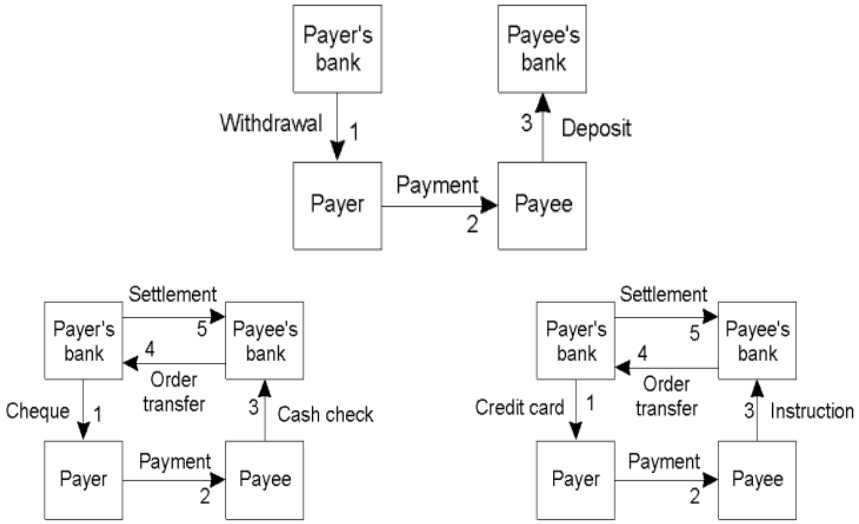

Figure 25.8: (a) Paying in cash; (b) Using a check; (c) Using a credit card.

25.2.1 E-cash

We want to use a digital form of cash which has anonymity properties. Users may generate "coins", an integer token, which are "blinded" (hide the sequence number of currency to the bank; this prevents the bank from tracking how they are spent), and request the bank to "sign" it. The signed coin is unblinded, and subsequently used as a currency, which are check for validity (valid coins have not already been spent) with the bank before transactions can take place.

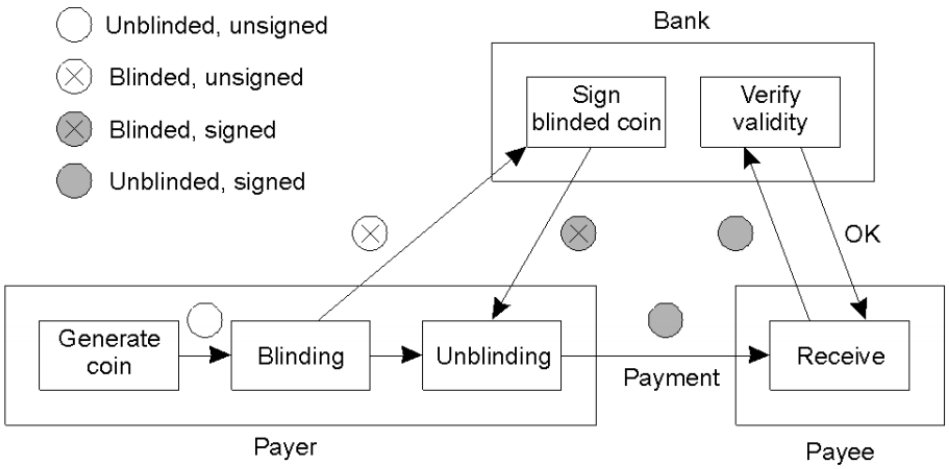

Figure 25.9: Illustration of E-cash.

25.2.2 Bitcoin

Bitcoin is a digital currency which uses peer-to-peer (P2P) transactions so as to avoid using a central bank or organization as the middle man (decentralized).

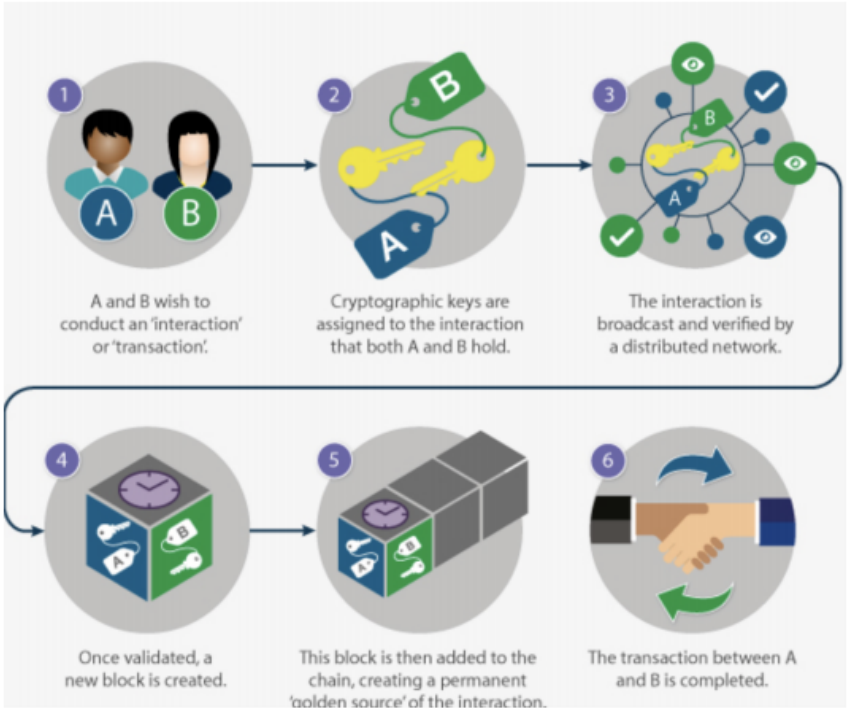

Figure 25.10: How Bitcoin works.

25.2.3 Blockchain: distributed ledger

Blockchain is a distributed database (ledger) which records every transactions that goes through the system. It uses a generic protocol for transactions based on public key cryptography. All the currency exchanged using the system and how much currency is owned by each person is publicly available. Transactions are signed by private keys and added into the ledger. The blockchain is massively replicated and shared using the P2P file sharing protocol. Special nodes called "miners" update the blockchain by appending blocks of multiple transactions. All nodes perform validation of transactions. Miner nodes perform "settlement" using a distributed consensus protocol.

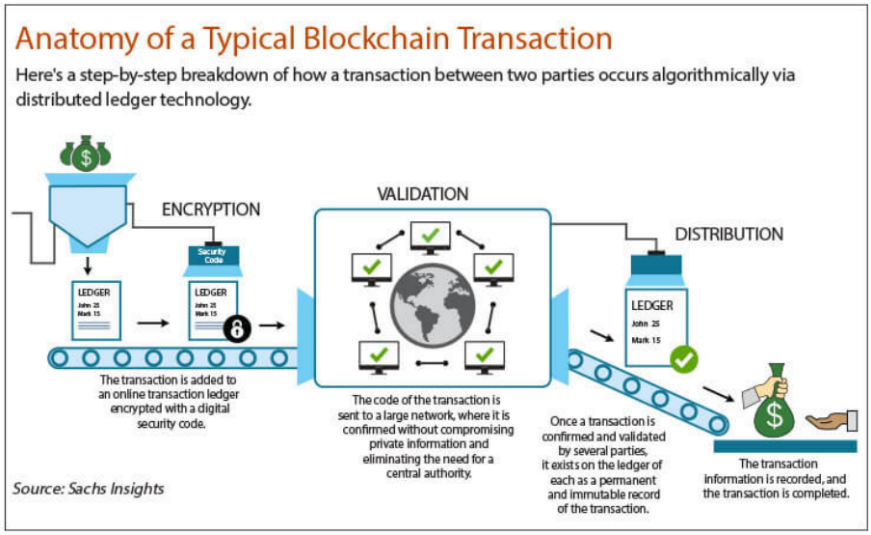

Figure 25.11: How blockchain works.