Computer Networks

Chapter 08

prof. dr ir Maarten van Steen

Vrije Universiteit Amsterdam
Faculty of Science
Dept. Mathematics and Computer Science
Room R4.20. Tel: (020) 444 7784

steen@cs.vu.nl

Contents

Security in Computer Networks

Goal: we want to enable secure communication between two parties in a distributed system. This requires implementing three related **security functions**:

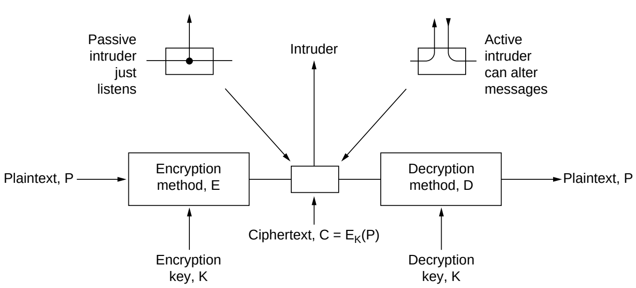
Authentication: (1) Ensuring that a message is genuine, has arrived exactly as it was sent, and came from the stated source; (2) Verifying the identity of an individual, such as a person at a remote terminal or the sender of a message.

Data integrity: The property that data has not been altered or destroyed in an unauthorized manner.

Confidentiality (secrecy): The property that information is not been made available or disclosed to unauthorized individuals, entities, or processes.

Implementing these functions generally requires the use of **cryptographic protocols**.

Cryptography



Cryptographic functions:

Secret key: Use a single key to (1) encrypt the plaintext and (2) decrypt the ciphertext. Requires that sender and receiver **share** the secret key.

Public key: Use different keys for encryption and decryption, of which one is **private**, and the other **public**.

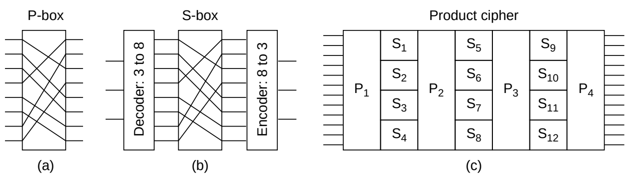
Hashing: Just use a hash function on the plaintext and send it off. There's no decryption at all, just verification.

Cryptology – Devising Ciphers

Substitution: Just replace some characters with others following a predefined mapping ⇒ implement by means of an **S-box**.

Transposition: Reshuffle the characters following a predefined pattern (e.g., a transposition table) ⇒ implement by means of an **P-box**.

Combine: Cascading a lot of S and P boxes does the trick.

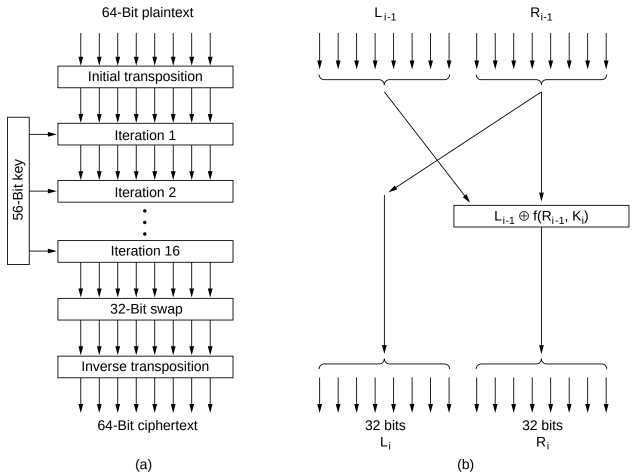


One-Time Pads

Simple idea: Choose a random bit string, as long as the plaintext, and simply XOR it to get the ciphertext. It can never be broken because the ciphertext has no information in it at all.

- They cannot be memorized
- The length of the transmitted data is limited by the key length
- Requires strict synchronization between sender and receiver: a single missed bit will screw up everything.

DES: Data Encryption Standard

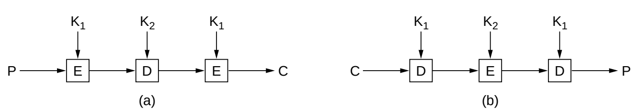


- Each iteration i uses a different key K_i. The complexity lies in the mangler function f.
- The keys K_i are derived from the initial 56-bit key.

The real problem is the 56-bit key: it's too easy to break.

Triple DES

Problem: The 56-bit key is way too short. The solution is to expand DES to a form with a 112-bit key.



Observation: By using an encrypt-decrypt-encrypt scheme, Triple DES is compatible with single DES by setting $K_1 = K_2$.

AES/Rijndael

Problem: DES is just too weak. The Advanced Encryption Standard is now gradually being used as the way to do symmetric encryption. AES is also known as Rijndael (winner of the AES contest).

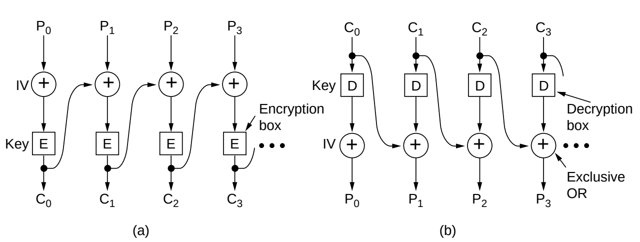
```
#define LENGTH 16                    /* # bytes in data block or key */
#define NROWS 4                      /* number of rows in state */
#define NCOLS 4                      /* number of columns in state */
#define ROUNDS 10                    /* number of iterations */
typedef unsigned char byte;          /* unsigned 8-bit integer */

rijndael(byte plaintext[LENGTH], byte ciphertext[LENGTH], byte key[LENGTH])
{
  int r;                             /* loop index */
  byte state[NROWS][NCOLS];          /* current state */
  struct {byte k[NROWS][NCOLS];} rk[ROUNDS + 1];      /* round keys */

  expand_key(key, rk);               /* construct the round keys */
  copy_plaintext_to_state(state, plaintext);   /* init current state */
  xor_roundkey_into_state(state, rk[0]);       /* XOR key into state */

  for (r = 1; r <= ROUNDS; r++) {
       substitute(state);                      /* apply S-box to each byte */
       rotate_rows(state);                     /* rotate row i by i bytes */
       if (r < ROUNDS) mix_columns(state);     /* mix function */
       xor_roundkey_into_state(state, rk[r]);  /* XOR key into state */
  }
  copy_state_to_ciphertext(ciphertext, state);     /* return result */
}
```


Cipher Block Chaining Mode

Problem: A 64-bit plaintext will always come out the same way. Great, now we can fool around a bit – just see if you can divide your input file into chunks of 64 bits *width*:

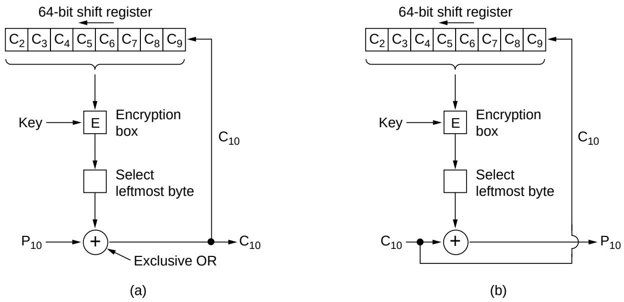
Name	Position	Bonus
Adams, Leslie	Clerk	$10
Black, Robin	Boss	$500,000
Collins, Kim	Manager	$100,000
Davis, Bobbie	Janitor	$5
Bytes 16	8	8

Solution: Use a predecessing, encrypted block to permute the current block before it's encrypted. Decryption works the other way around.



Cipher Feedback Mode

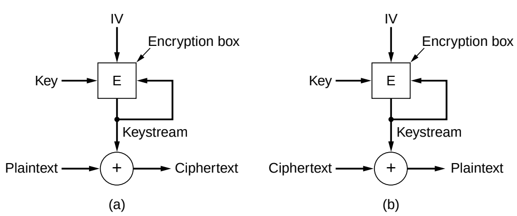
Issue: You don't always want to wait until a full 64-bit block has come in before starting to encrypt. You want to operate on bytes.



Idea: when a plaintext byte (P_{10}) needs to be encrypted, simply use a 64-bit ciphertext and select the leftmost byte to (1) send over the network, (2) push it in a shift register.

Stream Cipher Mode

Essence: Take an initialization vector (IV) and encrypt it, and use a key to an output block. The output block is encrypted to get another block, and so on. This gives an arbitrarily large sequence of output blocks that can be used to encrypt a stream:



Note: Never use the same *(keystream,IV)* pair or it will otherwise generate the same keystream.

RSA: Rivest, Shamit, Adleman

Really nifty: The whole idea is that you use a private and a public key. First find the right number for encryption:

1. Choose two large primes p and q
2. Compute $n = p \times q$ and $z = (p-1) \times (q-1)$
3. Choose a number d relatively prime to z
4. Find e such that $e \times d = 1 \bmod z$

Next step: Consider your plaintext as a bitstring; divide into blocks, where each block is considered to be a binary number $0 \leq P < n$.

Sending: encrypt each message P into C as: $C = P^e (\bmod n)$, and send it off. **Note:** you need e and n.

Receiving: decrypt an incoming message into $Q = C^d (\bmod n)$. Guess what: $Q = P^{e \cdot d} = P$. **Note:** you need d and n.

Note: if $\gcd(a,n) = 1$, then $a^{(p-1)(q-1)} \bmod n = 1$. Consequently, $P^{e \cdot d} \bmod n = P^{e \cdot d - 1} \times P \bmod n = P \bmod n$.

Digital Signatures

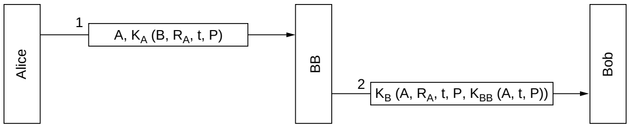
What we often really need is to authenticate a message, and assure its integrity:

1. Receiver can verify the claimed identity of the sender
2. The sender can later not deny that he/she sent the message
3. The receiver can not tamper the message itself.

The solution is to **digitally sign** the message. This means:

- have the sender put a signature that can be verified
- be sure that the signature cannot be faked, i.e. it should be uniquely associated with the message.

Symmetric-Key Signatures



Basic idea: Pretty simple – just use a *Big Brother* who passes the message, but signed, to the destination:

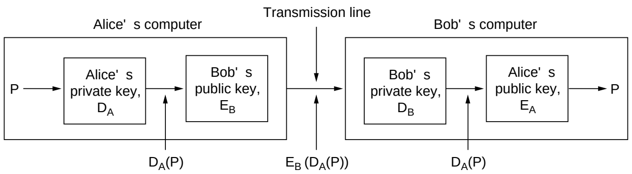
1. Alice sends $[A, K_A(B, R_A, t, P)$ to Big Brother.

2. Bib Brother signs $[A, t, P]$ and sends it along with the original message, encrypted with Bob's secret key: $[K_B(A, R_A, t, P, K_{BB}(A, t, P)]$.

Note: Using R_A and timestamps helps against replays.

Question: Why is signing by Big Brother necessary?

Public-Key Signatures



1. Alice encrypts her message P with her private key D_A: $P_A = D_A(P)$.
2. She then encrypts P_A with Bob's public key E_A: $E_A(P_A)$, and sends it off.
3. Bob decrypts the incoming message with his private key D_B. We know for sure that no one else has been able to read P_A during its transmission.
4. Bob decrypts the message with Alice's public key E_A, now knowing that it came fro Alice.

Note: we're assuming that $E_X(D_X(P)) = D_X(E_X(P))$

Message Digests

Idea: take an arbitrary length message, and compute a unique, fixed-length number from it. Also called **message digest**, or **one-way function**.

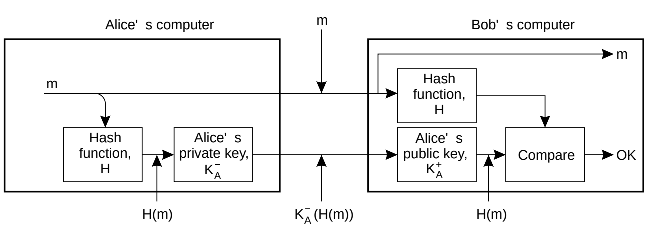
- Computing the hash $h(m)$ for any message m is relatively easy.
- Given a hash value $h(m)$, the only way of getting m is to enumerate over all possible messages. In other words, h^{-1} is almost impossible to find.
- It is computationally infeasible to find two messages m_1 and m_2 such that $h(m_1) = h(m_2)$.

Used for: password hashing (store hash values for comparison instead of cleartext passwords), message fingerprinting (add a message digest to the message to safeguard against changes), signatures (sign the message digest instead of the entire message).

Message Digests: Signatures

Basic idea: Don't mix authentication and secrecy. Instead, it should also be possible to send a message in the clear, but have it signed as well.

Solution: take a message digest, and sign that:



Public-Key Management

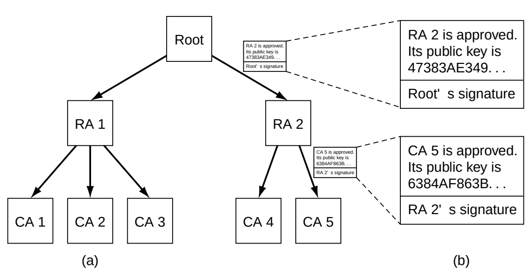
Problem: If two parties don't know each other, how can they get a hold of each other's public key and be *certain* that it's the right key?

Solution: Introduce a **trusted** third party that signs public keys by means of a **certificate**. The public key of this **certification authority** must be well known.

Field	Meaning
Version	Which version of X.509
Serial number	This number plus the CA's name uniquely identifi es the certifi cate
Signature algorithm	The algorithm used to sign the certifi cate
Issuer	X.500 name of the CA
Validity period	The starting and ending times of the validity period
Subject name	The entity whose key is being certifi ed
Public key	The subject's public key and the ID of the algorithm using it
Issuer ID	An optional ID uniquely identifying the certifi cate's issuer
Subject ID	An optional ID uniquely identifying the certifi cate's subject
Extensions	Many extensions have been defi ned
Signature	The certifi cate's signature (signed by the CA s private key)

Public-Key Infrastructures

Issue: We can't have just a single CA; we probably want several to distribute the work. The solution is simple: build a hierarchy (and cache certificates):



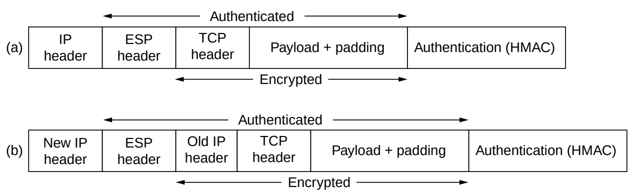
Question: What would be a good way to revoke certificates?

IPSec

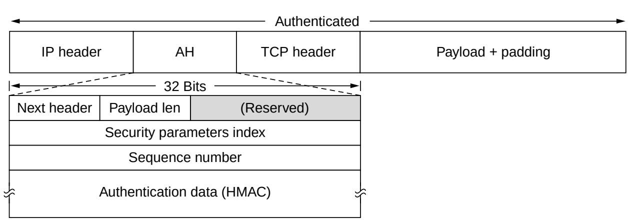
Issue: How can we incorporate *real* security in IP? The solution so far is IPSec by which IP packets can be sent securely over the Internet. Key establishment is still an open question (the original proposal is heavily flawed).

Transport mode: A separate IPSec header is inserted just after the normal IP header. It contains the information needed for secure transmission of the entire packet.

Tunnel mode: An entire IP packet is encapsulated in a new IPSec packet. Good for communication to/from a firewall that can leave the stations behind it unaware of IPSec.



IPSec Header



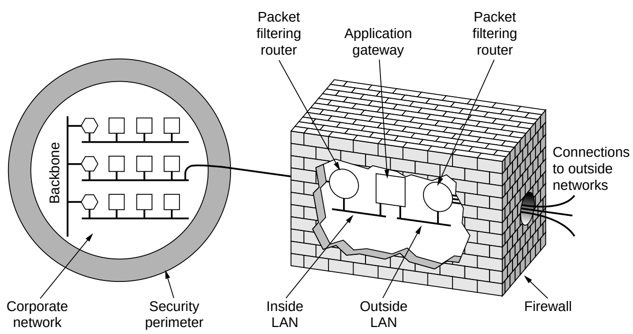
- **Index:** an identifier that associates this packet with a previous one. Essentially an index for the receiver to lookup the shared key they both use.
- **Sequence number:** *all* packets get a unique number (including retransmissions). Used for detecting replay attacks.
- **Authentication data:** contains the sender's digital signature.

Note: IPSec does not allow data to be encrypted; it is mainly used for integrity checking only.

Firewalls

Essence: Sometimes it's better to select service requests at the lowest level: network packets. Packets that do not fit certain requirements are simply removed.

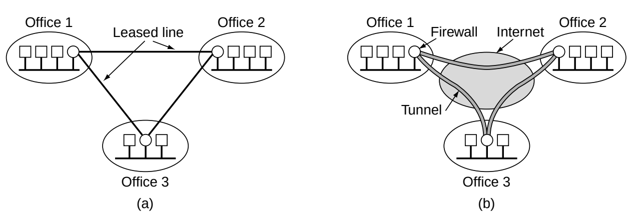
Solution: Protect your company by a firewall: it implements access control



Question: What do you think would be the biggest breach in firewalls?

Virtual Private Networks

Issue: Build your own private network that can span several different locations, for example, building IPSec tunnels between firewalls:



Wireless Security

802.11 (WEP): Based on keystreams. Unfortunately, the selection and use of the initialization vector is heavily flawed; the encryption algorithm has been broken; and it took only two hours for two students to build the software to eavesdrop on an industry 802.11 network.

Bluetooth: Far more sophisticated and applied to different layers of the protocol stack. See Tanenbaum for further details.

Authentication

Note: The whole business of security is that we can ensure authorized access to resources. In practice, this means that we pay a lot of attention to authentication first. Confidentiality, i.e. privacy, is practically less important.

Question: this is not entirely true – what's the big exception here?

Note: A stronger version of authentication is **nonrepudiation**: it is not possible for someone to **deny** that they sent a message.

Question: How can we safeguard against repudiation?

Authentication versus Integrity

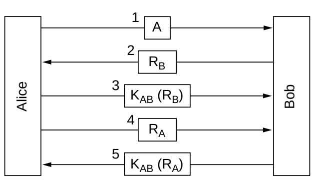
Note: Authentication and data integrity rely on each other: Consider an active attack by an enemy X on the communication from A to B.

Authentication without integrity: A's message is authenticated, and intercepted by X, who tampers with its content, but leaves the authentication part as is. B will conclude the message came from A – it came from X, so authentication fails.

Integrity without authentication: X intercepts a message from A, and then makes B believe that the content was really sent by X. The data has now been "changed" in an unauthorized manner, so integrity is violated. In other words: integrity is meaningless if you don't know the source of information.

Question: What can we say about confidentiality versus authentication and integrity?

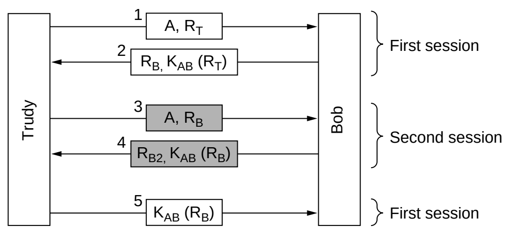
Authentication Protocols Secret Keys



1. Alice sends ID to Bob
2. Bob sends challenge R_B (i.e. a random number) to Alice
3. Alice encrypts R_B with shared key K_{AB}. Now Bob knows he's talking to Alice
4. Alice send challenge R_A to Bob
5. Bob encrypts R_A with K_{AB}. Now Alice knows she's talking to Bob

Note: We can "improve" the protocol by combining steps 1 & 4, and 2 & 5. This costs only the correctness.

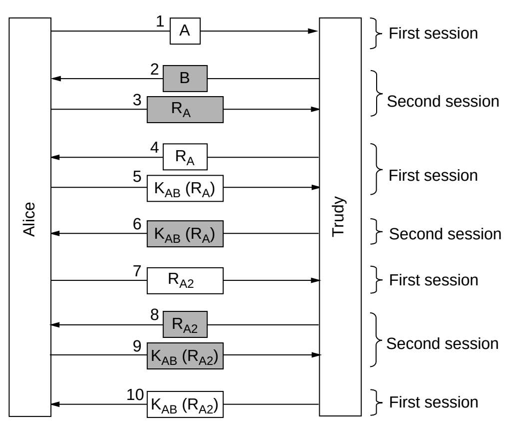
Authentication Protocols Reflection Attack (1/2)



1. Trudy claims she is Alice, and sends challenge R_T
2. Bob sends back a challenge R_B and the encrypted R_T
3. Trudy starts a second session, claiming she is Alice, but uses challenge R_B
4. Bob sends back a challenge, plus $\{R_B\}_{K_{AB}}$.
5. Trudy sends back $\{R_B\}_{K_{AB}}$ for the first session to prove she is Alice

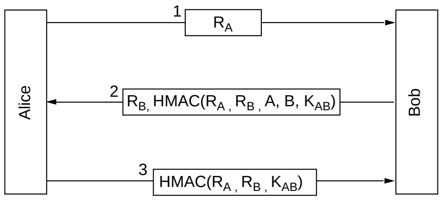
Authentication Protocols Reflection Attack (2/2)

It's even worse: Assume that Alice is a general-purpose computer:



Observation: Trudy can succesfully start two different sessions.

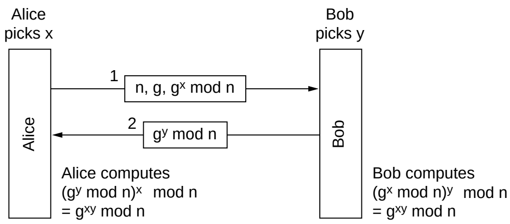
Authentication Protocols
Hashing Traffic



Important: The hash (HMAC) is computed using the knowledge shared by Alice and Bob. Essentially, Alice verifies that Bob is at the other end by computing the hash herself.

Establishing a Shared Key: Diffie-Hellman

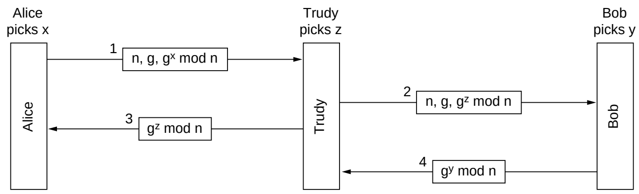
- Alice and Bob have to agree on two large prime numbers, n and g. Both numbers may be public.
- Alice chooses large number x, and keeps it to herself. Bob does the same, say y.



1. Alice sends $(n, g, g^x \bmod n)$ to Bob
2. Bob sends $(g^y \bmod n)$ to Alice
3. Alice computes $K_{AB} = (g^y \bmod n)^x = g^{xy} \bmod n$
4. Bob computes $K_{AB} = (g^x \bmod n)^y = g^{xy} \bmod n$

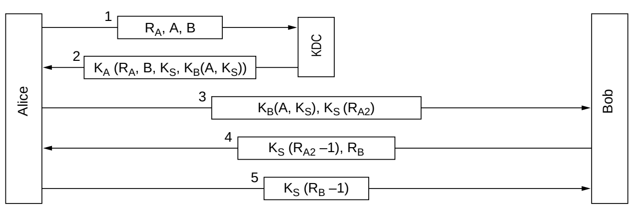
Bucket-Brigade Attack

Problem: Diffie Hellman works fine, but there is no way that Bob knows for sure he's getting information from Alice. Here comes Trudy again:



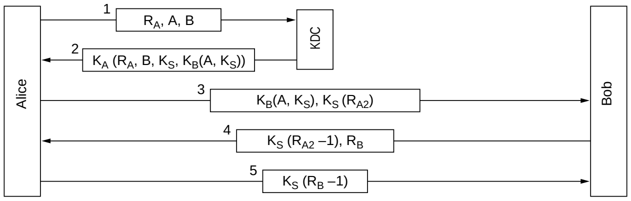
Needham-Schroeder (simplified)

Idea: There is a KDC which shares a key with a number of **principals**. A principal can request a session key to be used for a secure channel with another principal (also known to the KDC).



1. Alice asks KDC a session key for channel to Bob, with challenge R_A. KDC generates K_S and **ticket** $T_B = K_B(A, K_S)$.
2. KDC sends $K_A(R_A, B, K_S, T_B)$

Needham-Schroeder (cont'd)



3. Alice just sends the ticket, as well as a challenge $K_S(R_{A2})$. Bob retrieves the session key from the ticket.
4. Bob sends proof back and challenges Alice: $[K_S(R_{A2}-1), B]$.
5. Alice returns proof of the challenge: $K_S(R_B-1)$.

Needham-Schroeder (cont'd)

Q1: Why does the KDC put Bob into its reply message, and Alice into the ticket?

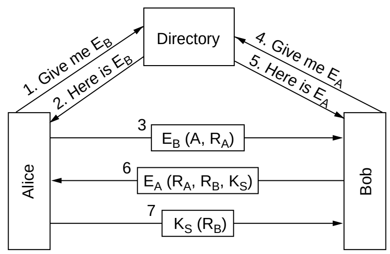
Q2: The ticket sent back to Alice by the KDC is encrypted with Alice's key. Is this necessary?

Security flaw: Suppose Trudy finds out Alice's key ⇒ she can use that key anytime to impersonate Alice, even if Alice changes her private key at the KDC.

Reasoning: Once Trudy finds out Alice's key, she can use it to decrypt a (possibly old) ticket for a session with Bob, and convince Bob to talk to her using the old session key.

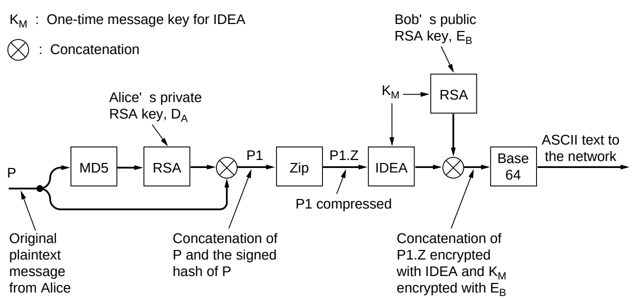
Solution: Have Alice get an encrypted number from Bob first, and put that number in the ticket provided by the KDC ⇒ we're now ensuring that every session is known at the KDC.

Authentication Protocols Public Key



1. Alice sends a challenge R_A to Bob, encrypted with Bob's public key E_B.
2. Bob decrypts the message, proves he's Bob (by sending R_A back), and sends a challenge R_B to Alice, along with a session key K_S. Everything's encrypted with Alice's public key E_A.
3. Alice proves she's Alice by sending back the decrypted challenge, but now encrypted with the session key K_S.

Pretty Good Privacy (1/2)



1. Calculate hash (MD5) of message, and encrypt that hash with Alice's private key $\Rightarrow$ you've got Alice's signature.

2. Append signature to text, and compress it to P1.Z.

3. Encrypt P1.Z with IDEA, and send along key K_M, after encrypting it with Bob's public key $\Rightarrow$ Bob can get K_M for decryption.

PGP (2/2)

Some observations:

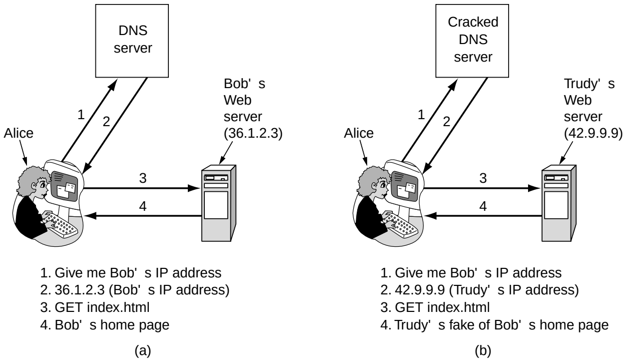
- Expensive RSA is used only to encrypt two 128-bit messages. IDEA, which is much more efficient, is used for the hard stuff.
- Public keys are stored locally and can be retrieved in different ways. For that reason, there is a value indicating the strength of the trust the holder has in that key. Don't use low-trusted keys for high-security messages.
- A user can maintain several private-public key pairs. This allows easy switching to another key pair when one is suspected to have been compromised.

Web Security

- Secure naming
- Secure sockets
- Mobile code security

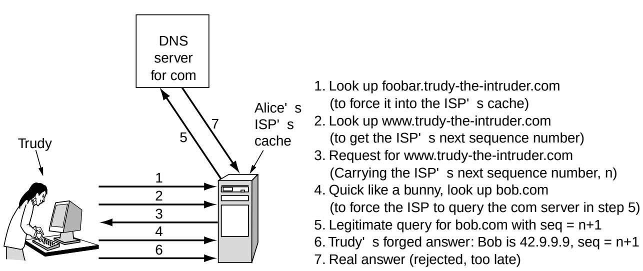
Secure Naming (1/2)

Essence: Break into DNS and replace the name-to-address mapping of a DNS name.



Secure Naming (2/2)

Observation: DNS uses UDP which prevents checking whether replies are actually from the host you queried. Trudy can now "easily" **spoof** DNS:



Assumption: The DNS cache is initially empty (or cached entry has expired).

Secure DNS

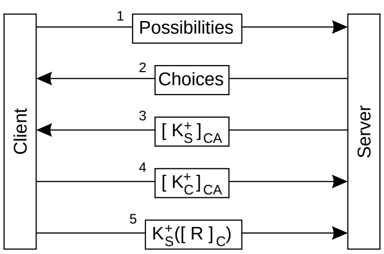
Solution: Normally, entries are filled by having one server send its local database (zone) to the requester. Simply let the originator sign what it sends.

Domain name	Time to live	Class	Type	Value
bob.com.	86400	IN	A	36.1.2.3
bob.com.	86400	IN	KEY	3682793A7B73F731029CE2737D...
bob.com.	86400	IN	SIG	86947503A8B848F5272E53930C...

When you need to know an IP address, you get the relevant resource records back associated with, for example, *bob.com*. Assume you know the public key of *com*. You now have *bob.com*'s public key, which allows to check *www.bob.com*.

Secure Sockets Layer

Essence: SSL is a secure-message layer just on top of the transport layer. It consists of two separate phases: (1) establishing a secure connection, and (2) using it.



Note: $[K_X^+]_Y$ denotes the public key of X, signed by Y. R is a random number generated and signed by the client for authentication.

Question: How can authentication work here?

Mobile Code Security

Java Applets: Interpreted code that can be run in a **sandbox**, by which every instruction is inspected before being executed

ActiveX: Whether or not an ActiveX control is run depends on who (if anything) signed the code and if that entity is trusted. If trusted, the control can do anything a normal program can do.