

Network Security (II)

RSA - Rivest, Shamir, and Adleman (cont.)

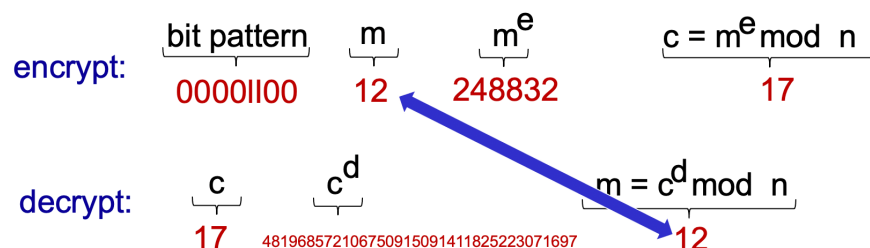
- RSA: getting ready
 - message: just a bit pattern
 - bit pattern can be uniquely represented by an integer number
 - thus, encrypting a message is equivalent to encrypting a number
 - Example:
 - $m = 1001\ 0001$. This message is uniquely represented by the decimal number 145.
 - to encrypt m , we encrypt the corresponding number, which gives a new number (the ciphertext).
- RSA: Creating public/private key pair
 1. choose two large prime numbers p , q . (e.g., 1024 bits each)
 2. compute $n = pq$, $z = (p-1)(q-1)$
 3. choose e (with $e < n$) that has no common factors with z (e , z are “relatively prime”).
 4. choose d such that $ed-1$ is exactly divisible by z . (in other words: $ed \bmod z = 1$).
 5. public key K_B^+ is (n, e) . private key K_B^- is (n, d) .
- RSA: encryption, decryption
 - given (n, e) and (n, d) as computed above
 - to encrypt message m ($m < n$), compute $c = m^e \bmod n$
 - to decrypt received bit pattern, c , compute $m = c^d \bmod n$

magic happens!

$$m = (\underbrace{m^e \bmod n}_c)^d \bmod n$$

from number theory
[Kaufman 1995]

- RSA example:
 - B choose $p = 5$, $q = 7$, Then $n = 35$, $z = 24$. $e = 5$ (so, e , z relatively prime).
 - $d = 29$ (so $ed - 1$ exactly divisible by z).
 - encrypting 8-bit messages.



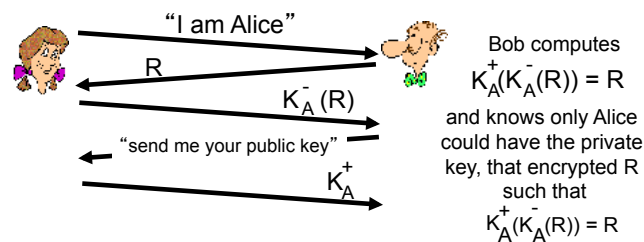
- Is RSA secure enough?
 - suppose you know B's public key (n, e) . How hard is it to determine d ?
 - essentially need to find factors of n without knowing the two factors p and q
 - fact: factoring a big number is hard
 - How do you break it?
 - if there is algorithms for quickly factoring a number [not known yet, but be Quantum Computer]
- RSA in practices: used for session keys
 - exponentiation in RSA is computationally intensive
 - DES is at least 100 times faster than RSA
 - use public/private key crypto to establish secure connection, then establish second key - symmetric session key - for encrypting data

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

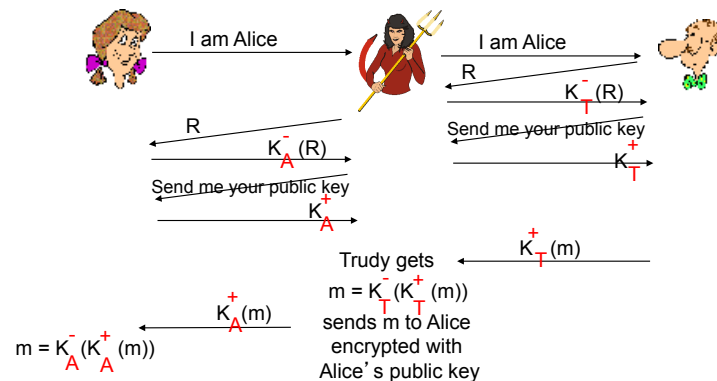
- Basic idea:
 - B verifies A by using public/private key cryptography
 - based on the idea that only possessor of private key could have encrypted something that decrypts using its public key



- 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

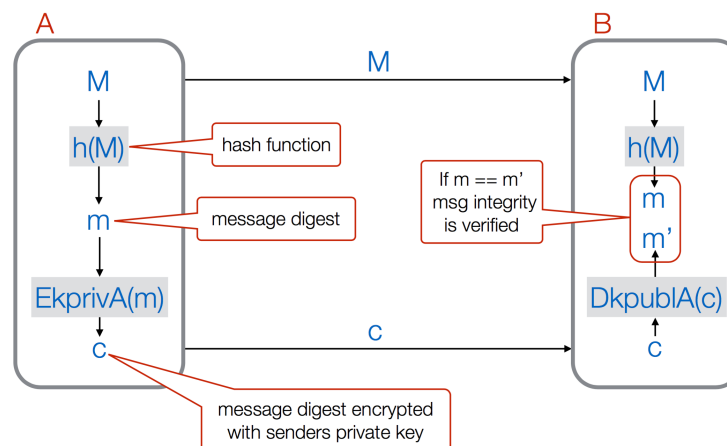
- solution: use of a **nonce**
 - number (R) used only **once-in-a-lifetime**
 - A use K_A^- to encrypt R sent by B ($K_A^-(R)$), B uses K_A^+ to decrypt and check R
 - Even if the intruder recorded ($K_A^-(R)$) and use it to pretend to be Alice later, B will know the number R is out-of-date after deception and know this Alice is a fake Alice.
- Still unaddressed: We need a trusted way to obtain someone's public key
 - security hole: difficult to detect, all three people receive all messages

man (or woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)



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, send encrypted (with senders private key) hash of message
 - if hash of the received message agrees with decrypted received hash, it is assured that the message was not altered in transit



- 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`
 - `openssl`: multi-platform, opensource SSL/TLS toolkit

Disclaimer: Notes adapted from the textbook and online resources.