Security:
Public Key Cryptography

CS 352, Lecture 19

http://www.cs.rutgers.edu/~sn624/352-S19

Srinivas Narayana

(heavily adapted from slides by Prof. Badri Nath and the textbook authors)

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Review: Security

- Key properties: Confidentiality, integrity, authenticity
- Cryptography: prevents adversaries from reading our data
- Terminology: Encryption, decryption, plain text, cipher text, keys, ciphers
- Symmetric key cryptography: shared secret among communicating parties
- Key building blocks: substitution and permutation
- Stream and block ciphers
- Block ciphers that use substitution: use a mathematical function instead of a lookup table

Encryption using symmetric keys

- Same key for encryption and decryption
- Efficient to implement: Often the same or very similar algorithm for encryption and decryption
- Achieves confidentiality
- No integrity: message vulnerable to tampering
- No authentication by itself
- Vulnerable to replay attacks
  - Bad guy can steal the encrypted message and later present it on behalf of a legitimate user
- How to agree on keys?

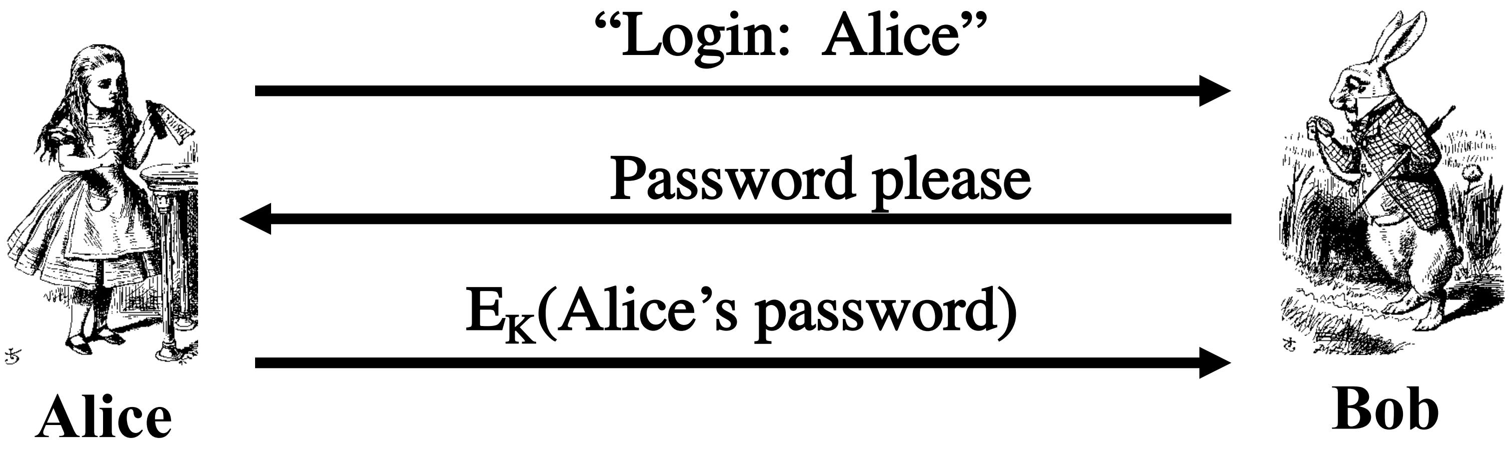
Problems with Block Ciphers

Hence, also problems with symmetric key cryptography

An example: Login system

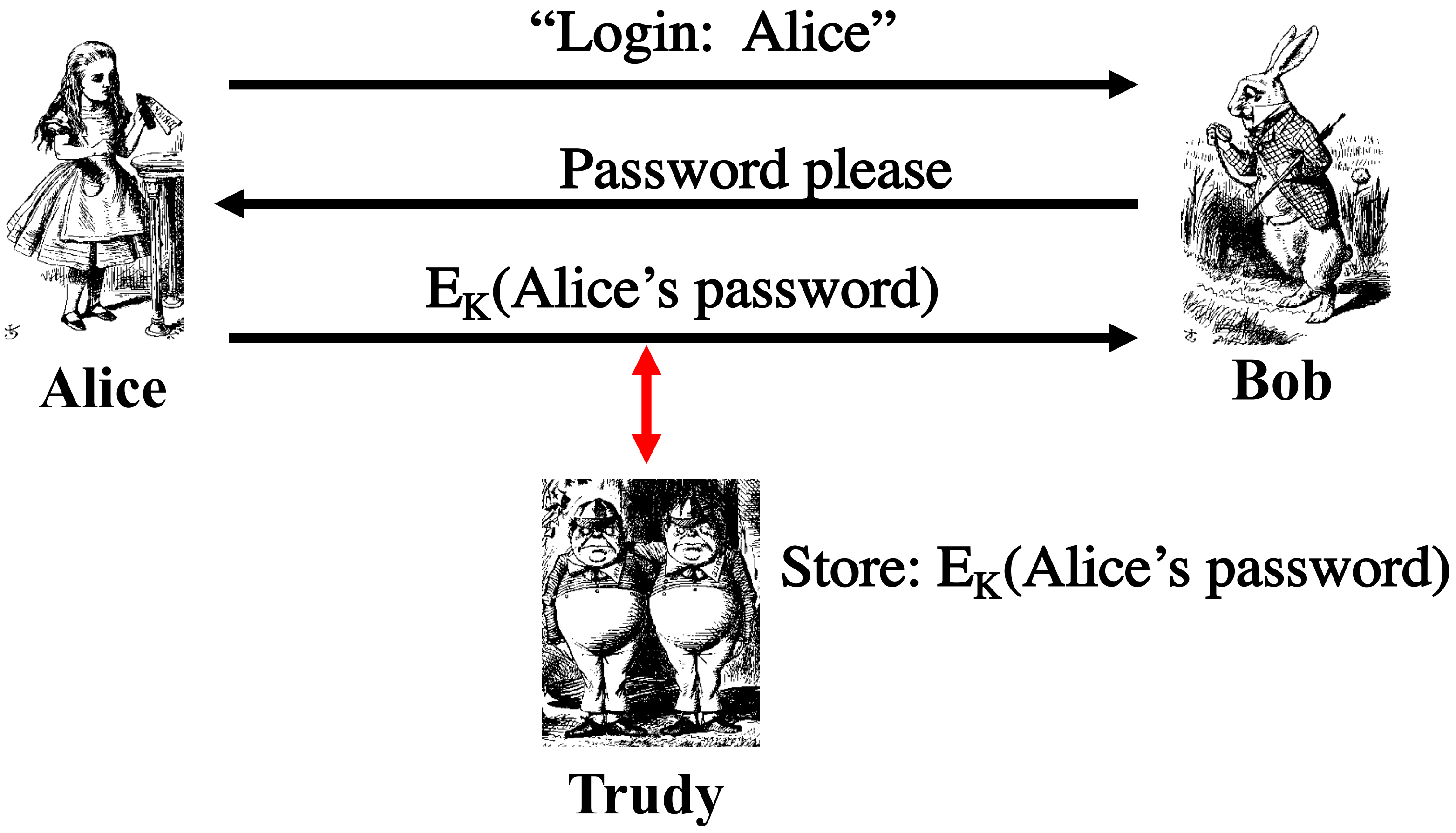
- Bob runs a login server to provide access to protected resources
- Alice must present a password to login
- Exchange of password implemented using symmetric key cryptography on top of block ciphers

Replay attack

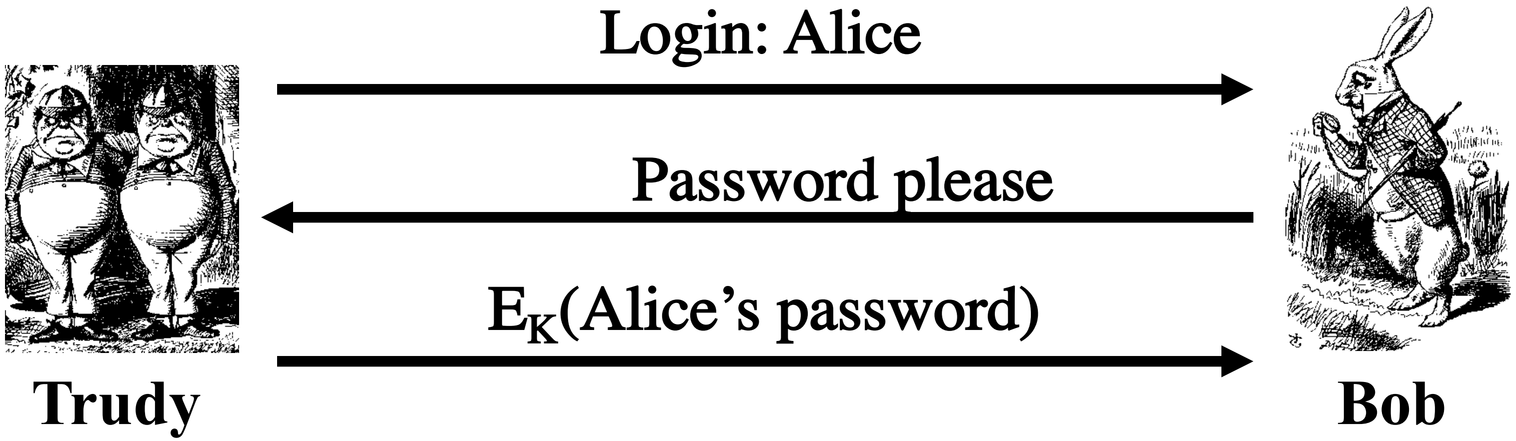


- Alice's password is encrypted
  - From both Bob and attackers
  - If Bob is trusted, he can decrypt using the shared secret key
- But subject to replay attack

Replay attack

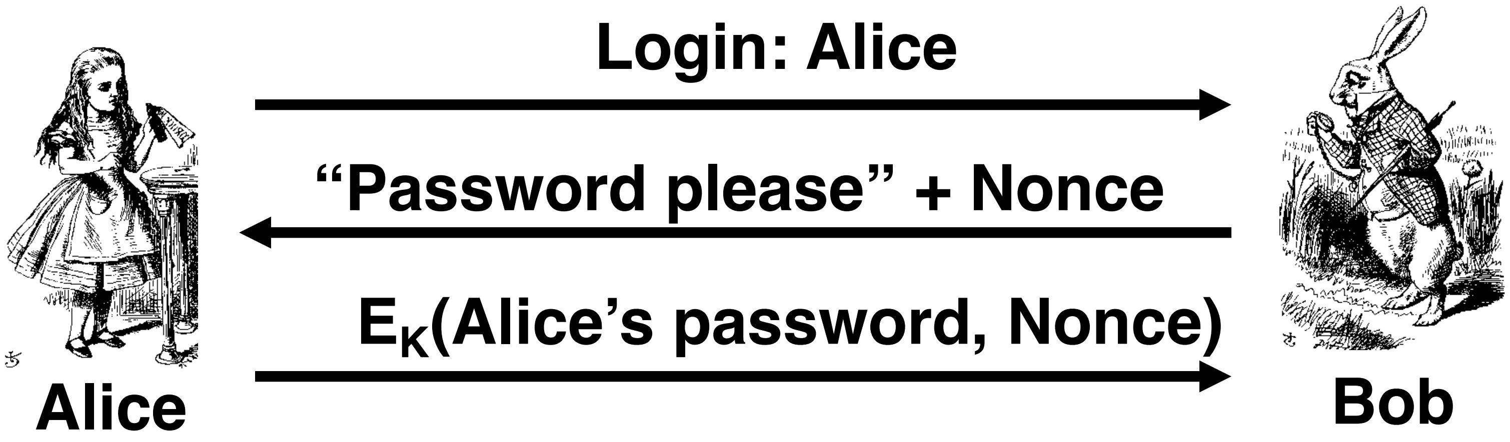


Replay attack



- This is a replay attack
- How can we prevent a replay?
- By adding a NONCE value; Number used once only
  - Use a temporary random number

Challenge-Response



- Nonce is a challenge that is changed every time
- The encrypted message is the response
- Critically, the ciphertext depends on the nonce

How do nonces help?

- What if Trudy steals the ciphertext?
  - Nonce changed every time ➔ ciphertext is fresh for each login
  - Even if Trudy steals the authenticating ciphertext, she can’t reuse it

- Does the nonce need to be confidential?

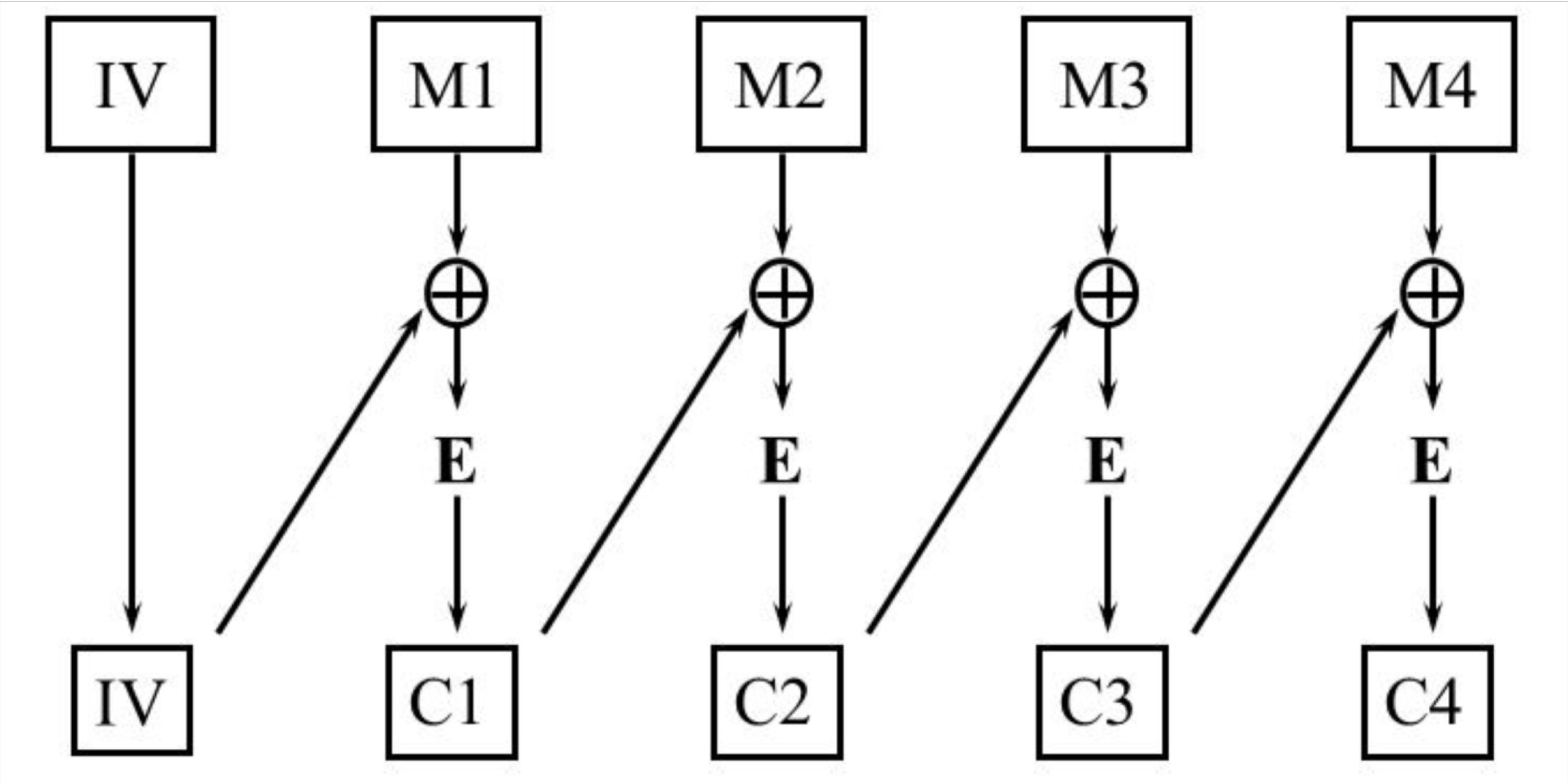
General problems with repeated ciphertext

- Block ciphers take chunks of info (ex: 64-bit) to other chunks
  - Previous example: Repeated passwords can be replayed
- But more generally, easy to guess parts of the payload with repeated plaintext
- Example: “HTTP/1.1” often occurs on HTTP messages
  - Trudy could guess which ciphertext payloads contain that plaintext
  - Then use those parts of a message to guess other parts of the payload
  - … and so on

Can we use nonces for all messages?

- Yes!
  - Remember, nonces can be sent as plain text
- Example: Use ciphertext $E_k(\text{message} \oplus \text{nonce})$ to respond to the challenge
  - Here, $\oplus$ is the bitwise XOR operation
- But very inefficient:
  - For the example above, send double # bits for every message
- Use a method to generate nonces automatically
- Cipher block chaining: use the previous ciphertext as a nonce for the next plain text block
- First block “randomized” using Initialization Vector (IV)

Cipher block chaining: Encryption



Exercise: how would decryption work?

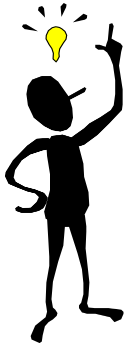
How to agree on a shared secret key?

- In reality: two parties may meet in person or communicate “out of band” to exchange shared key
- But communicating parties may never meet in person
  - Example: An online retailer and customer
  - It’s very common not to meet someone you talk to over a network
- What if the shared secret is stolen?
  - Must exchange keys securely again
- Is there a way to communicate securely without worrying about secure key exchange?

Next topic: Public key cryptography

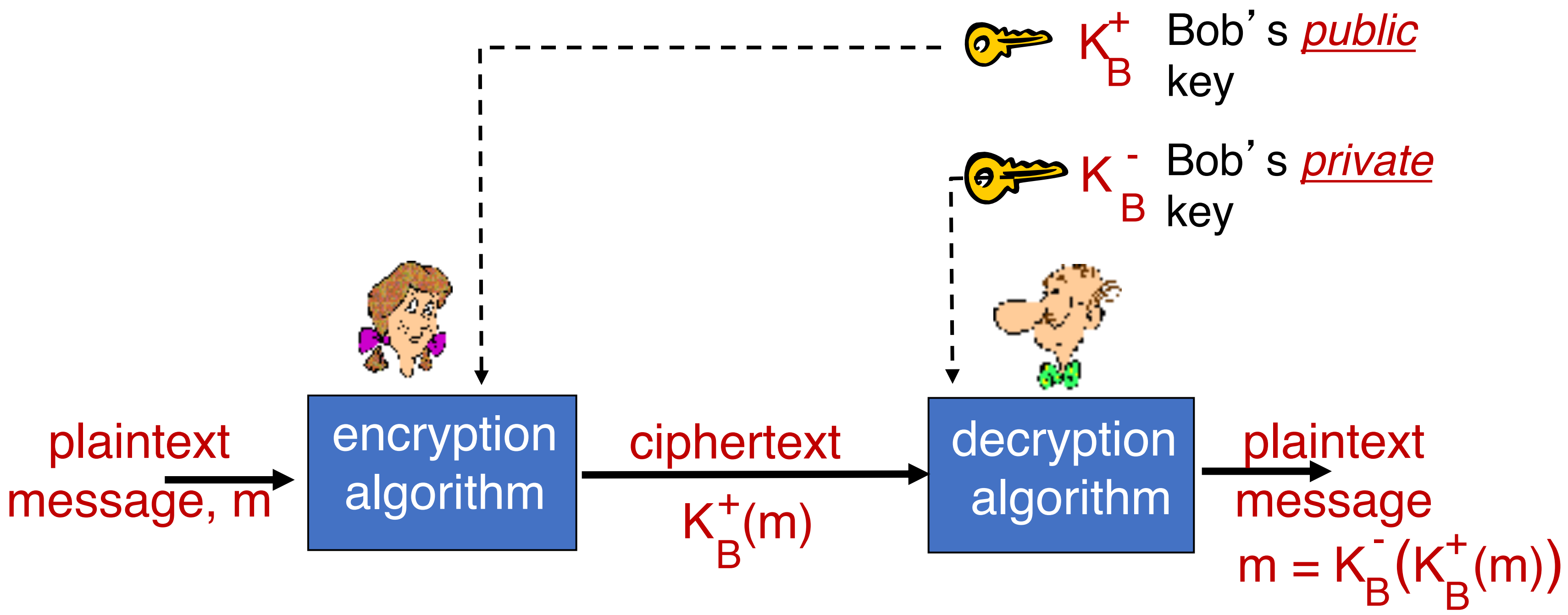
Public key cryptography

Public Key Cryptography



- A radically different approach [Diffie-Hellman76, RSA78]
- sender, receiver do *not* share secret key
- *public* encryption key known to *all*
- *private* decryption key known only to the receiver

Public key cryptography



Public Key Cryptography

An Example

Sally

Two keys:

$K_{pub,sally}$

$K_{priv,sally}$

Jeff

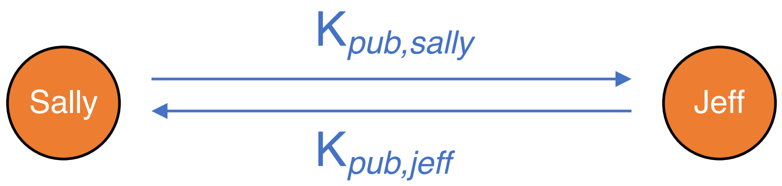
Two keys:

$K_{pub,jeff}$

$K_{priv,jeff}$

Public Key Cryptography

An Example



Two keys:
$K_{pub,sally}$
$K_{priv,sally}$

Two keys:
$K_{pub,jeff}$
$K_{priv,jeff}$

Sally and Jeff exchange *public* keys

(As we'll see later, this may happen through a trusted third authority)

Public Key Cryptography

An Example

$C = E_{K_{pub,sally}}(P)$

Two keys:
$K_{pub,sally}$
$K_{priv,sally}$

Two keys:
$K_{pub,jeff}$
$K_{priv,jeff}$

If Jeff wants to send an encrypted plaintext message P to Sally, he uses Sally's public key to encrypt the message to form ciphertext C

Public Key Cryptography

An Example

$P = D_{K_{priv,sally}}(C)$

Two keys:
$K_{pub,sally}$
$K_{priv,sally}$

Two keys:
$K_{pub,jeff}$
$K_{priv,jeff}$

Sally uses her *private* key to decrypt the message C from Jeff. Only Sally can decrypt messages that are encrypted using her public key. A message to Sally cannot be decrypted using Sally's public key.

Public key ciphers

RSA

Public key encryption algorithms

requirements:

1. need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that

$$K_B^-(K_B^+(m)) = m$$

2. given public key K_B^+, it should be impossible to compute private key K_B^-

RSA: Rivest, Shamir, Adelson algorithm

Prerequisite: modular arithmetic

- x mod n = remainder of x when divide by n
- facts:

  [(a mod n) + (b mod n)] mod n = (a+b) mod n

  [(a mod n) - (b mod n)] mod n = (a-b) mod n

  [(a mod n) * (b mod n)] mod n = (a*b) mod n
- thus

  $(a \bmod n)^d \bmod n = a^d \bmod n$
- example: x=14, n=10, d=2:

  $(x \bmod n)^d \bmod n = 4^2 \bmod 10 = 6$

  $x^d = 14^2 = 196 \quad x^d \bmod 10 = 6$

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= 10010001 . 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 step 1: 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 mod $z = 1$).
5. *public* key is $\underbrace{(n,e)}_{K_B^+}$. *private* key is $\underbrace{(n,d)}_{K_B^-}$.

RSA step 2: encryption and decryption

0. given (n,e) and (n,d) as computed above

1. to encrypt message m ($<n$), compute

   $c = m^e \bmod n$

2. to decrypt received bit pattern, c, compute

   $m = c^d \bmod n$

magic happens! $m = (\underbrace{m^e \bmod n}_{c})^d \bmod n$

RSA example:

Bob chooses *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.

encrypt: $\underbrace{\text{bit pattern}}_{0000\text{l}000} \quad \underbrace{m}_{12} \quad \underbrace{m^e}_{24832} \quad \underbrace{c = m^e \bmod n}_{17}$

decrypt: $\underbrace{c}_{17} \quad \underbrace{c^d}_{481968572106750915091411825223071697} \quad \underbrace{m = c^d \bmod n}_{12}$

Why does RSA work?

- must show that $c^d \bmod n = m$ where $c = m^e \bmod n$
- fact: for any x and y: $x^y \bmod n = x^{(y \bmod z)} \bmod n$
  - where $n = pq$ and $z = (p-1)(q-1)$
- thus,

$$
\begin{aligned}
c^d \bmod n &= (m^e \bmod n)^d \bmod n \\
&= m^{ed} \bmod n \\
&= m^{(ed \bmod z)} \bmod n \\
&= m^1 \bmod n \\
&= m
\end{aligned}
$$

RSA: another important property

The following property will be very useful later (for authentication and integrity):

$$\underbrace{K_B^-(K_B^+(m))}_{\text{use public key first, followed by private key}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{use private key first, followed by public key}}$$

use public key first, followed by private key

use private key first, followed by public key

result is the same!

Why $K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$?

follows directly from modular arithmetic:

$$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$$
$$= m^{de} \bmod n$$
$$= (m^d \bmod n)^e \bmod n$$

Why is RSA secure?

- Suppose you know Bob'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
- Turns out that no one knows efficient algorithms to factor big numbers, especially a product of two large primes

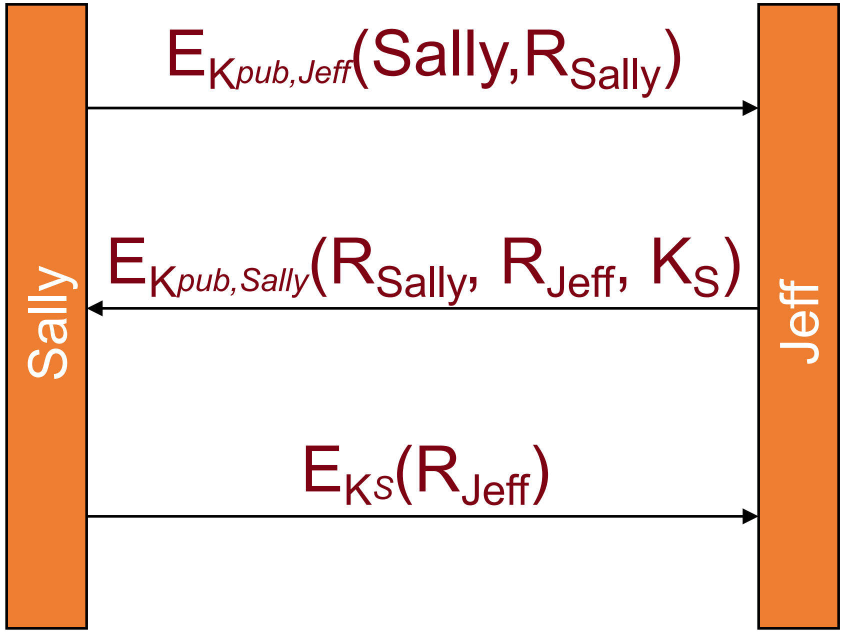
RSA in practice: session keys

- exponentiation in RSA is computationally intensive
- DES is at least 100 times faster than RSA
- use public key crypto to establish secure connection, then establish second key – symmetric session key – for encrypting data

session key, K_S

- Bob and Alice use RSA to exchange a symmetric key K_S
- once both have K_S, they use symmetric key cryptography

RSA in practice: session keys



Public Key Cryptography: Summary

- Why public key cryptography is so powerful:
  - No need to exchange secret keys securely
  - Only the receiver of encrypted information holds the secret key
  - Public keys are exactly that: public!
- Examples of public key algorithms:
  - Merkle-Helman knapsack
  - Rivest-Shamir-Adleman (RSA)
  - Pretty Good Privacy (PGP)

Cryptography: the big picture

- Algorithms underlying secure communication over the Internet
    - Pervades almost everything we use
    - Example: HTTPS? (we'll see more about that soon…)
- Specific algorithms like AES and RSA are widely implemented on host and server systems

- So far: mainly confidential communication

- Next lectures: We'll see how cryptography is a building block for integrity and authenticity of communication as well