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)
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

“Login: Alice”

Password please

$E_K$(Alice’s password)

Store: $E_K$(Alice’s password)

Alice

Bob

Trudy
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 $\Rightarrow$ 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

plaintext message, $m$

Encryption algorithm

$ciphertext = K_B^+(m)$

Decryption algorithm

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

Bob’s public key

$K_B^+$

Bob’s private key

$K_B^-$
Public Key Cryptography

An Example

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

Two keys:
- $K_{pub, jeff}$
- $K_{priv, jeff}$
Public Key Cryptography

An Example

Sally and Jeff exchange *public* keys

(As we’ll see later, this may happen through a trusted third authority)
Public Key Cryptography

An Example

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 $C$.

An Example

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 $C$.
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

• $x \mod n = \text{remainder of } x \text{ when divide by } n$
• facts:
  $[(a \mod n) + (b \mod n)] \mod n = (a+b) \mod n$
  $[(a \mod n) - (b \mod n)] \mod n = (a-b) \mod n$
  $[(a \mod n) \times (b \mod n)] \mod n = (a\times b) \mod n$
• thus
  $(a \mod n)^d \mod n = a^d \mod n$
• example: $x=14$, $n=10$, $d=2$:
  $(x \mod n)^d \mod n = 4^2 \mod 10 = 6$
  $x^d = 14^2 = 196 \quad x^d \mod 10 = 6$
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 \((n, e)\). private key is \((n, d)\).
**RSA step 2: encryption and decryption**

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

1. to encrypt message \(m < n\), compute
   \[ c = m^e \mod n \]

2. to decrypt received bit pattern, \(c\), compute
   \[ m = c^d \mod n \]

---

**magic happens!**

\[ m = (m^e \mod n)^d \mod n \]
RSA example:


- $e=5$ (so $e$, $z$ relatively prime).
- $d=29$ (so $ed-1$ exactly divisible by $z$).

encrypting 8-bit messages.

**Encrypt:**

<table>
<thead>
<tr>
<th>bit pattern</th>
<th>m</th>
<th>$m^e$</th>
<th>$c = m^e \mod n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001000</td>
<td>12</td>
<td>24832</td>
<td>17</td>
</tr>
</tbody>
</table>

**Decrypt:**

$$c = 17$$

$$c^d = 481968572106750915091411825223071697$$

$$m = c^d \mod n = 12$$
Why does RSA work?

• must show that $c^d \mod n = m$ where $c = m^e \mod n$

• fact: for any $x$ and $y$: $x^y \mod n = x^{(y \mod z)} \mod n$
  • where $n = pq$ and $z = (p-1)(q-1)$

• thus,
  $c^d \mod n = (m^e \mod n)^d \mod n$
  $= m^{ed} \mod n$
  $= m^{(ed \mod z)} \mod n$
  $= m^1 \mod n$
  $= m$
RSA: another important property

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

\[ K^{-}_B(K^+_B(m)) = m = K^+_B(K^{-}_B(m)) \]

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:

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

$$= m^{de} \mod n$$

$$= (m^d \mod n)^e \mod n$$
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

\textit{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

\[ E_{K_{pub, Jeff}}(Sally, R_{Sally}) \]

\[ E_{K_{pub, Sally}}(R_{Sally}, R_{Jeff}, K_{S}) \]

\[ E_{K_{S}}(R_{Jeff}) \]
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