Cryptography

Rosen Section 4.6

Tom Michoel

MNF130V2020 – Week 9

Cryptography

- **Cryptography** is the subject of transforming information so that it cannot be easily recovered without special knowledge.

Cryptography

- **Cryptography** is the subject of transforming information so that it cannot be easily recovered without special knowledge.
- **Encryption** is the process of making a message secret.

Cryptography

- **Cryptography** is the subject of transforming information so that it cannot be easily recovered without special knowledge.
- **Encryption** is the process of making a message secret.
- **Decryption** is the process of determining the original message from the encrypted message.

Cryptography

- **Cryptography** is the subject of transforming information so that it cannot be easily recovered without special knowledge.
- **Encryption** is the process of making a message secret.
- **Decryption** is the process of determining the original message from the encrypted message.
- Number theory is the basis of much of cryptography. Modern *ciphers* rely on *modular arithmetic* and the difficulty of the *prime factorization* problem.

Shift cyphers

- A **shift cypher** shifts each letter k letters forward in the alphabet, sending the last k to the first k.

Shift cyphers

- A **shift cypher** shifts each letter k letters forward in the alphabet, sending the last k to the first k.
- Mathematically it can be described as a function f on $\mathbb{Z}_{26}$ (where each letter corresponds to an integer between 0 and 25) which maps $n \in \mathbb{Z}_{26}$ to $f(n) = (n + k) \bmod 26$.

Shift cyphers

- A **shift cypher** shifts each letter k letters forward in the alphabet, sending the last k to the first k.
- Mathematically it can be described as a function f on $\mathbb{Z}_{26}$ (where each letter corresponds to an integer between 0 and 25) which maps $n \in \mathbb{Z}_{26}$ to $f(n) = (n + k) \bmod 26$.
- If we know the *key* k, we can decrypt using the *inverse function* $f^{-1}(n) = (n - k) \bmod 26$.

Shift cyphers

- A **shift cypher** shifts each letter k letters forward in the alphabet, sending the last k to the first k.
- Mathematically it can be described as a function f on $\mathbb{Z}_{26}$ (where each letter corresponds to an integer between 0 and 25) which maps $n \in \mathbb{Z}_{26}$ to $f(n) = (n + k) \bmod 26$.
- If we know the *key* k, we can decrypt using the *inverse function* $f^{-1}(n) = (n - k) \bmod 26$.

Example

Julius Caesar made messages secret using a shift cypher with $k = 3$. For instance, to encrypt the message "THE ANSWER IS NO", translate it to numbers, shift and translate back:

$$\begin{array}{rrrr} 20\,8\,5 & 1\,14\,19\,23\,5 & 9\,19 & 14\,15 \\ 23\,11\,8 & 4\,17\,22\,26\,8 & 12\,21 & 17\,18 \end{array}$$

to obtain "WKH DQVZH LU QR".

Public vs. private key cryptography

- In **private key cryptography**, once you know an *encryption* key, you can quickly find the *decription* key.

Public vs. private key cryptography

- In **private key cryptography**, once you know an *encryption* key, you can quickly find the *decription* key.
- In **public key cryptography**, knowing how to send encrypted messages does not help to decrypt them.

Public vs. private key cryptography

- In **private key cryptography**, once you know an *encryption* key, you can quickly find the *decription* key.
- In **public key cryptography**, knowing how to send encrypted messages does not help to decrypt them.
- The **RSA cryptosystem** is a widely used *public key* cryptosystem.

The RSA cryptosystem

- Each individual has a **public encryption key** (n, e), and **private decryption key** d, where:

The RSA cryptosystem

- Each individual has a **public encryption key** (n, e), and **private decryption key** d, where:
  - The **modulus** $n = pq$ is the product of two large primes p, q.

The RSA cryptosystem

- Each individual has a **public encryption key** (n, e), and **private decryption key** d, where:
  - The **modulus** $n = pq$ is the product of two large primes p, q.
  - The **exponent** e is relatively prime to $(p-1)(q-1)$, $\gcd(e, (p-1)(q-1)) = 1$

The RSA cryptosystem

- Each individual has a **public encryption key** (n, e), and **private decryption key** d, where:
  - The **modulus** $n = pq$ is the product of two large primes p, q.
  - The **exponent** e is relatively prime to $(p-1)(q-1)$, $\gcd(e, (p-1)(q-1)) = 1$
  - The **decryption key** d is an inverse of e modulo $(p-1)(q-1)$.
- p and q can be found with relative ease using probabilistic primality tests, but factoring n into p and q is computationally infeasible.

The RSA cryptosystem

- Each individual has a **public encryption key** (n, e), and **private decryption key** d, where:
  - The **modulus** $n = pq$ is the product of two large primes p, q.
  - The **exponent** e is relatively prime to $(p-1)(q-1)$, $\gcd(e, (p-1)(q-1)) = 1$
  - The **decryption key** d is an inverse of e modulo $(p-1)(q-1)$.
- p and q can be found with relative ease using probabilistic primality tests, but factoring n into p and q is computationally infeasible.
- Messages *to* an individual are encrypted using that person's *public* encryption key.

The RSA cryptosystem

- Each individual has a **public encryption key** (n, e), and **private decryption key** d, where:
  - The **modulus** $n = pq$ is the product of two large primes p, q.
  - The **exponent** e is relatively prime to $(p-1)(q-1)$, $\gcd(e, (p-1)(q-1)) = 1$
  - The **decryption key** d is an inverse of e modulo $(p-1)(q-1)$.
- p and q can be found with relative ease using probabilistic primality tests, but factoring n into p and q is computationally infeasible.
- Messages *to* an individual are encrypted using that person's *public* encryption key.
- The current standard is to have keys of 2048 bits (617 digits) or primes with about 300 digits each.

RSA in a nutshell

- A message m is encrypted using the encryption function

$$c = f(m) = m^e \bmod n$$

- An encrypted message c is decrypted using the decryption function

$$m = f^{-1}(c) = c^d \bmod n$$

RSA encryption

- A message M is translated into strings of digits using the map

$$f : \{A, B, C, \ldots, Z\} \to \{00, 01, 02, \ldots, 25\}$$

and divided in blocks $m_1, m_2, \ldots, m_k$ each of length $2N$, where $2N$ is the largest even integer such that $\underbrace{2525\ldots25}_{2N} \leq n$.

RSA encryption

- A message M is translated into strings of digits using the map

$$f : \{A, B, C, \dots, Z\} \to \{00, 01, 02, \dots, 25\}$$

  and divided in blocks $m_1, m_2, \dots, m_k$ each of length $2N$, where $2N$ is the largest even integer such that $\underbrace{2525\dots25}_{2N} \leq n$.

- Each block m_i is encrypted using the function

$$c = f(m) = m^e \bmod n$$

.

RSA decryption

- To decrypt an RSA encrypted messaged (recover m from c), we use a **private decryption key** d, an inverse of e modulo $(p-1)(q-1)$.

RSA decryption

- To decrypt an RSA encrypted messaged (recover m from c), we use a **private decryption key** d, an inverse of e modulo $(p-1)(q-1)$.
- The decryption function is

$$m = f^{-1}(c) = c^d \bmod n$$

RSA decryption

- To decrypt an RSA encrypted messaged (recover m from c), we use a **private decryption key** d, an inverse of e modulo $(p-1)(q-1)$.
- The decryption function is

$$m = f^{-1}(c) = c^d \bmod n$$

- Decryption recovers the original message, if and only if

$$m = \left(m^e \bmod n\right)^d \bmod n$$

RSA decryption

- To decrypt an RSA encrypted messaged (recover m from c), we use a **private decryption key** d, an inverse of e modulo $(p-1)(q-1)$.
- The decryption function is

$$m = f^{-1}(c) = c^d \bmod n$$

- Decryption recovers the original message, if and only if

$$m = \left(m^e \bmod n\right)^d \bmod n$$

- That this result is true is based on the following results:
  - Existence of a decryption key
  - Fermat's little theorem
  - The Chinese remainder theorem

Existence of a decryption key

- Recall that if integers a, m are relatively prime, $\gcd(a, m) = 1$, then there exists a unique inverse of a modulo m.

Existence of a decryption key

- Recall that if integers a, m are relatively prime, $\gcd(a, m) = 1$, then there exists a unique inverse of a modulo m.
- By assymption, the exponent e is relative prime to $(p-1)(q-1)$. Hence an inverse d of e modulo $(p-1)(q-1)$ exists.

Existence of a decryption key

- Recall that if integers a, m are relatively prime, $\gcd(a, m) = 1$, then there exists a unique inverse of a modulo m.
- By assymption, the exponent e is relative prime to $(p-1)(q-1)$. Hence an inverse d of e modulo $(p-1)(q-1)$ exists.
- Recall that this means

$$de \bmod (p-1)(q-1) = 1$$

and that there exists an integer k such that

$$de = 1 + k(p-1)(q-1)$$

Existence of a decryption key

- Recall that if integers a, m are relatively prime, $\gcd(a, m) = 1$, then there exists a unique inverse of a modulo m.
- By assymption, the exponent e is relative prime to $(p-1)(q-1)$. Hence an inverse d of e modulo $(p-1)(q-1)$ exists.
- Recall that this means

$$de \bmod (p-1)(q-1) = 1$$

  and that there exists an integer k such that

$$de = 1 + k(p-1)(q-1)$$

- Hence we have

$$m^{de} = m(m^k)^{(p-1)(q-1)}$$

  and

$$c^d \bmod n = (m^e)^d \bmod n = (m \bmod n)\big((m^k)^{(p-1)(q-1)} \bmod n\big) \bmod n$$

Fermat's little theorem

Theorem (Fermat's little theorem)

If p is prime and a is an integer not divisible by p then

$$a^{p-1} \equiv 1 \pmod{p}.$$

Furthermore, for every integer a we have

$$a^p \equiv a \pmod{p}.$$

Fermat's little theorem

Theorem (Fermat's little theorem)

If p is prime and a is an integer not divisible by p then

$$a^{p-1} \equiv 1 \pmod{p}.$$

Furthermore, for every integer a we have

$$a^p \equiv a \pmod{p}.$$

This theorem allows some modular exponentiations to be computed even more rapidly. Recall that we computed $3^{67} \bmod 7 = 3$ by expressing 67 in its binary expansion. In fact, by Fermat's little theorem

$$3^{67} \bmod 7 = 3^{1+11\cdot 6} \bmod 7 = (3 \bmod 7)\big((3^{11})^6 \bmod 7\big) = 3 \bmod 7 = 3$$

That 3^{11} is not divisible by 7 follows from the fact that the *unique* prime number factorization of 3^{11} is $3 \cdot 3 \cdot \cdots \cdot 3$ does not contain 7.

RSA Decryption

- By Fermat's little theorem, we have

$$(m^k)^{(p-1)(q-1)} \equiv 1 \pmod{p}$$
$$(m^k)^{(p-1)(q-1)} \equiv 1 \pmod{q}$$

RSA Decryption

- By Fermat's little theorem, we have

$$(m^k)^{(p-1)(q-1)} \equiv 1 \pmod{p}$$
$$(m^k)^{(p-1)(q-1)} \equiv 1 \pmod{q}$$

- Hence $(m^k)^{(p-1)(q-1)}$ is a solution to the **system of linear congruences**

$$x \equiv 1 \pmod{p}$$
$$x \equiv 1 \pmod{q}$$

with $\gcd(p, q) = 1$ (because p and q are primes).

RSA Decryption

- By Fermat's little theorem, we have

$$\begin{aligned}(m^k)^{(p-1)(q-1)} &\equiv 1 \pmod{p}\\ (m^k)^{(p-1)(q-1)} &\equiv 1 \pmod{q}\end{aligned}$$

- Hence $(m^k)^{(p-1)(q-1)}$ is a solution to the **system of linear congruences**

$$\begin{aligned}x &\equiv 1 \pmod{p}\\ x &\equiv 1 \pmod{q}\end{aligned}$$

  with $\gcd(p,q) = 1$ (because p and q are primes).

- By the Chinese remainder theorem, $x \equiv 1 \pmod{pq}$, and, using $pq = n$, we obtain

$$(m^k)^{(p-1)(q-1)} \equiv 1 \pmod{n}$$

- Hence

$$c^d \bmod n = (m \bmod n)\big((m^k)^{(p-1)(q-1)} \bmod n\big) \bmod n = m \bmod n = m$$

The last equality follows because the original message blocks m were constructed to be $\leq n$.

Secure communication with RSA

- To **send** an encrypted message to your friend, use **her** public encryption key. She will be able to decrypt it using her private decryption key.

Secure communication with RSA

- To **send** an encrypted message to your friend, use **her** public encryption key. She will be able to decrypt it using her private decryption key.
- To **receive** an encrypted message, share **your** public public encryption key. You will be able to decrypt it using your private decryption key.

Digitial signatures with RSA

Let your public key be (n, e) with private key d. To send a message M such that the recipient knows it came from you:

Digitial signatures with RSA

Let your public key be (n, e) with private key d. To send a message M such that the recipient knows it came from you:

- Divide M into integer blocks m_i such that $0 \leq m_i \leq n$ for all i.

Digitial signatures with RSA

Let your public key be (n, e) with private key d. To send a message M such that the recipient knows it came from you:

- Divide M into integer blocks m_i such that $0 \leq m_i \leq n$ for all i.
- Apply **your (private) decryption** function $c = f^{-1}(m) = m^d \bmod n$ to each block.

Digitial signatures with RSA

Let your public key be (n, e) with private key d. To send a message M such that the recipient knows it came from you:

- Divide M into integer blocks m_i such that $0 \leq m_i \leq n$ for all i.
- Apply **your (private) decryption** function $c = f^{-1}(m) = m^d \bmod n$ to each block.
- The recipient applies **your (public) encryption** function $f(c) = c^e \bmod n$ to the received message. By the same calculation as before

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

Digitial signatures with RSA

Let your public key be (n, e) with private key d. To send a message M such that the recipient knows it came from you:

- Divide M into integer blocks m_i such that $0 \leq m_i \leq n$ for all i.
- Apply **your (private) decryption** function $c = f^{-1}(m) = m^d \bmod n$ to each block.
- The recipient applies **your (public) encryption** function $f(c) = c^e \bmod n$ to the received message. By the same calculation as before

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

- Only if the message was really sent by you (that is, if it was modified by *your* private decryption function), will this result in a readable message.

Digitial signatures with RSA

Let your public key be (n, e) with private key d. To send a message M such that the recipient knows it came from you:

- Divide M into integer blocks m_i such that $0 \leq m_i \leq n$ for all i.
- Apply **your (private) decryption** function $c = f^{-1}(m) = m^d \bmod n$ to each block.
- The recipient applies **your (public) encryption** function $f(c) = c^e \bmod n$ to the received message. By the same calculation as before

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

- Only if the message was really sent by you (that is, if it was modified by *your* private decryption function), will this result in a readable message.
- The message itself is **not secure**: your public key is public and hence everyone will be able to reconstruct the original message.

Encrypted messages with digital signatures

Let (n_s, e_s) and d_s be the public and private keys of the **sender**.

Let (n_r, e_r) and d_r be the public and private keys of the **recipient**.

To send an encrypted message M with the sender's digital signature:

Encrypted messages with digital signatures

Let (n_s, e_s) and d_s be the public and private keys of the **sender**.

Let (n_r, e_r) and d_r be the public and private keys of the **recipient**.

To send an encrypted message M with the sender's digital signature:

- Divide M into integer blocks m_i such that $0 \leq m_i \leq n$ for all i.

Encrypted messages with digital signatures

Let (n_s, e_s) and d_s be the public and private keys of the **sender**.

Let (n_r, e_r) and d_r be the public and private keys of the **recipient**.

To send an encrypted message M with the sender's digital signature:

- Divide M into integer blocks m_i such that $0 \leq m_i \leq n$ for all i.
- The sender applies her *private* decryption function $c = m^{d_s} \bmod n_s$ to each block.

Encrypted messages with digital signatures

Let (n_s, e_s) and d_s be the public and private keys of the **sender**.

Let (n_r, e_r) and d_r be the public and private keys of the **recipient**.

To send an encrypted message M with the sender's digital signature:

- Divide M into integer blocks m_i such that $0 \leq m_i \leq n$ for all i.
- The sender applies her *private* decryption function $c = m^{d_s} \bmod n_s$ to each block.
- The sender applies the recipient's *public* encryption function $c' = c^{e_r} \bmod n_r$ to each block.

Encrypted messages with digital signatures

Let (n_s, e_s) and d_s be the public and private keys of the **sender**.

Let (n_r, e_r) and d_r be the public and private keys of the **recipient**.

To send an encrypted message M with the sender's digital signature:

- Divide M into integer blocks m_i such that $0 \leq m_i \leq n$ for all i.
- The sender applies her *private* decryption function $c = m^{d_s} \bmod n_s$ to each block.
- The sender applies the recipient's *public* encryption function $c' = c^{e_r} \bmod n_r$ to each block.
- The recipient first applies her *private* decryption and then the sender's *public* encryption function to reconstruct the message:

$$(c')^{d_r} \bmod n_r = c^{e_r d_r} \bmod n_r = c$$

$$c^{e_s} \bmod n_s = (m^{d_s e_s}) \bmod n = m$$

Encrypted messages with digital signatures

Let (n_s, e_s) and d_s be the public and private keys of the **sender**.

Let (n_r, e_r) and d_r be the public and private keys of the **recipient**.

To send an encrypted message M with the sender's digital signature:

- Divide M into integer blocks m_i such that $0 \leq m_i \leq n$ for all i.
- The sender applies her *private* decryption function $c = m^{d_s} \bmod n_s$ to each block.
- The sender applies the recipient's *public* encryption function $c' = c^{e_r} \bmod n_r$ to each block.
- The recipient first applies her *private* decryption and then the sender's *public* encryption function to reconstruct the message:

$$(c')^{d_r} \bmod n_r = c^{e_r d_r} \bmod n_r = c$$
$$c^{e_s} \bmod n_s = (m^{d_s e_s}) \bmod n = m$$

- Now only the recipient is able to reconstruct the message and she will still know it came from the sender.

RSA DIY

- On unix(-like) systems, keys are stored in `~/.ssh/`:
  - `id_rsa`: private key (base64 format represented with ASCII characters)[1]
  - `id_rsa.pub`: public key
  - `known_hosts`: public keys of trusted servers
- `ssh-keygen` is the standard utility to generate RSA key pairs on unix(-like) systems.[2]
- `OpenSSH` offers more utilities to play with RSA keys[3]:
  - Generate 2048 bit keys:
    `> openssl genrsa -des3 -out private.pem 2048`
  - Extract public key:
    `> openssl rsa -in private.pem -outform PEM -pubout -out public.pem`
  - Extract public key modulus and exponent:
    `> openssl rsa -pubin -in public.pem -text -noout`

---

[1] https://www.cs.sfu.ca/~ggbaker/zju/math/int-alg.html
[2] https://en.wikipedia.org/wiki/ssh-keygen
[3] https://en.wikipedia.org/wiki/OpenSSH