Public-Key Cryptography

Introduction

A drawback of private key cryptography is that it requires prior communication of the secret key k between the communicating parties over a secure channel.

The idea behind public-key cryptosystem is to break a key into two parts, a public key and a private key, in which the public key can be used for encryption while the private key is used for decryption.

The public key can be made public for everyone to see.

We will consider public-key schemes for key distribution, secrecy and message integrity.

We begin with a discussion of key distribution.

Key Distribution

Introduction

In private-key cryptography, two users share a secret key to establish a "secure" communications channel.

One question that needs to addressed is: How do the users share the secret key?

A requirement is that the secret key needs to be shared using a secure channel.

In some settings, this can be done. For example: close physical proximity to one another, trusted courier, etc.

In many settings, this is a difficult or expensive problem.

Introduction

In a more general setting, consider an organization with $N \geq 2$ employees where each pair of employees may need to communicate securely.

If each pair of the N employees need a secret key to communicate, then a total of $\binom{N}{2}$ keys are needed and each employee may need to store and manage a lot of keys.

Solving this problem can be quite complex if N is large and new keys are needed for each communications session.

Scenarios where a lot of secret keys are needed occurs in practice all the time : For example, customer sending credit-card data to a merchant, emailing a colleague, doing online banking, etc.

Problems

There are three problems related to the use of private-key cryptography with respect to the secure sharing of keys.

- key distribution - how keys are distributed for secure communications.
- key management - the storage and management of large numbers of secret keys.
- usage of cryptography in open systems.

Key Distribution Center (KDC)

We now present a partial solution to these problems.

We will consider these problems in the setting of a large organization where entities of the organization needs secure communication.

One approach is to deploy a central, trusted entity known as a key distribution center (KDC) that will help all the employees share secret keys.

When a new employee joins, the KDC can share a (long-term) key with that employee so the employee can communicate securely with the KDC.

Suppose Alice and Bob are employees and has secret keys k_{Alice}, k_{Bob} respectively, for communicating with the KDC.

Key Distribution Center (KDC)

At some later point, if Alice wishes to communicate securely with Bob, Alice can send a secure message to the KDC stating she wishes to communicate with Bob.

Then the KDC generates a session key k, sends $\text{Enc}_{k_{Alice}}(k)$ to Alice and $\text{Enc}_{k_{Bob}}(k)$ to Bob.

Since Alice and Bob now share the secret session key k, they can use it to communicate securely with each other.

This is all done within the confines of private-key cryptography.

A slightly different approach involving a KDC for obtaining a session key for Alice and Bob to communicate securely is given in the following slide.

Key Distribution Center (KDC)

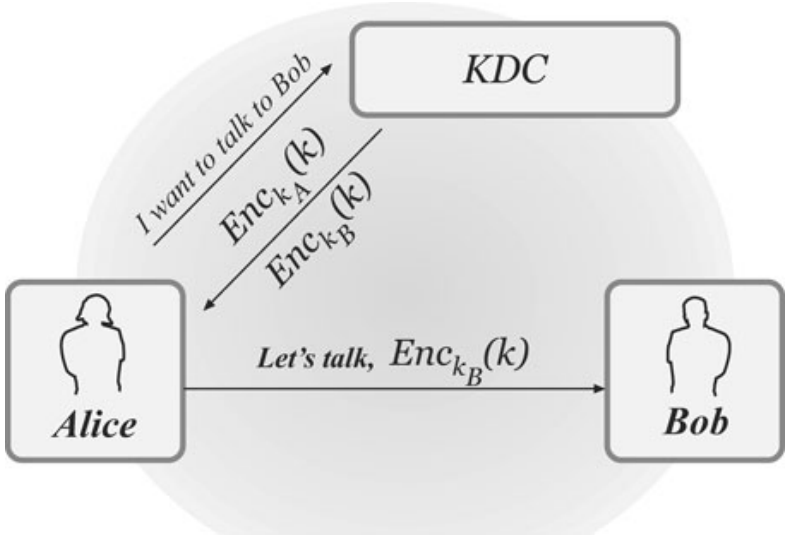

Figure: General framework for KDC Protocols

Here, the KDC only communicates with Alice who initiated the request for a session key for Alice and Bob to communicate securely.

The KDC send both $\text{Enc}_{k_{Alice}}(k)$ and $\text{Enc}_{k_{Bob}}(k)$ to Alice who then passes on $\text{Enc}_{k_{Bob}}(k)$ to Bob.

Key Distribution Center (KDC)

Advantages of using a KDC:

- Each employee needs to store only one key - the key it uses to communicate securely with the KDC. Note that the KDC needs to store many long-term keys, one for each employee.
- Each time a new employee joins the organization, it only needs to set up the secret key to communicate with the KDC. This has no effect on other employees.

Key Distribution Center (KDC)

Issues with using KDCs:

- Requires a private/secure channel for employee to obtain the key used to communicate with KDC.
- If attacker is successful in penetrating the KDC, this leads to a complete break in the system.
- The KDC is a single point of failure.

Kerberos is a well-known protocol by implementing a KDC.

The main point is that the distribution and management of keys are important problems but classical cryptography does not adequately address these problems.

We now consider an alternative approach to centralized key management and distribution.

A New Approach

In 1976, W. Diffie and M.E. Hellman published an article title New Directions in Cryptography.

This article provided a technique for achieving secure communications without ever communicating over a private channel and started the public-key cryptography revolution.

Main Ideas:

- there are problems that exhibit asymmetry in the sense that it is easy to compute but hard to invert.
- Make use of asymmetry to enable to parties to agree on a shared secret key using a public discussion. This is known as key exchange.

Key Exchange

Suppose two parties, Alice and Bob, runs a probabilistic protocol Π to generate a shared, secret key..

Π can be thought of as the set of instructions for Alice and Bob in the protocol.

They both start with the input 1^n (where n is the security parameter) and they run Π using (independent) random bits

At the end of the protocol, Alice and Bob output keys $k_A, k_B \in \{0,1\}^n$ respectively.

We required the keys generated satisfy $k_A = k_B$ and therefore we let the key $k = k_A = k_B$ be generated in some "honest" execution of Π.

Since Π is a randomized protocol, the key, in general, will be different each time Π is executed.

Before providing the actual key-exchange protocol (the Diffie-Hellman protocol) we need to introduce some the concept of (algebraic) group theory.

As you will see later on, public-key cryptography relies on numerous ideas from number theory and abstract algebra.

Some Group Theory

Groups

We begin with a definition of a group.

Definition 19.1 (group).

A group $(G, \circ)$ consists of a set G along with a binary operation $\circ(\cdot, \cdot)$ (where instead of writing $\circ(g, h)$, we write $g \circ h$) such that the following conditions hold:

- *(closure) : For all $g, h \in G, g \circ h \in G$.*
- *(associativity): For all $g_1, g_2, g_3 \in G, (g_1 \circ g_2) \circ g_3 = g_1 \circ (g_2 \circ g_3)$.*
- *(Existence of an identity) : There exists $e \in G$ (known as an identity) such that for all $g \in G, e \circ g = g \circ e = g$.*
- *(Existence of inverses) : For each $g \in G$, there exists an element $h \in G$ such that $g \circ h = h \circ g = e$. We say that h is an inverse of g*

Groups

A group G can have finite or infinite number of elements. We say G is a finite group if it has a finite number of elements. Otherwise, we say G is an infinite group.

If G is a finite group, then $|G|$, the number of elements in G, is called the order of the group.

Group elements do not necessarily commute with each other. That is, there may exist $g, h \in G$such that $g \circ h \neq h \circ g$.

Suppose $(G, \circ)$ is a group and suppose for every $g, h \in G$ the statement $g \circ h = h \circ g$ hold. Then we say G is an Abelian group.

When there is no ambiguity about the operation $\circ$, we simply call the set G a group.

Groups

It turns out that the identity of a group is unique and for any $g \in G$, its inverse is unique.

Claim 19.2.

Let $(G, \circ)$ be a group. Then it has a unique identity and each element in G has a unique inverse.

Groups

Instead of writing $g \circ h$, we usually write

- $g + h$ (this is known as additive notation)
- $g \cdot h$ or gh (this is known as multiplicative notation).

We'll use $(G, \circ)$ to denote that G is a multiplicative group and $(G, +)$ to denote that G is an additive group.

For a group G, we say it is a mulitplicative group if the operation is given using multiplication notation and is an additive group if the operation is given using addition notation.

It's important to remember that this is just a notation and doesn't necessarily mean addition or multiplication in the literal sense. It depends on the set G along with what the actual group operation is.

Groups

In the group $(G, +)$, we denote the group identity by 0 and in $(G, \cdot)$ we denote the group identity by 1.

Given a $g \in G$, the inverse in additive notation is given by $-g$ and the inverse in multiplicative notation is given by g^{-1}.

Most of the time, we'll use multiplicative group notation. That is, the group is given as $(G, \cdot)$.

We now give some examples of groups.

Examples of Groups

Example: Consider the integers $\mathbb{Z}$ under the addition operation. That is, the group $(\mathbb{Z}, +)$. This is a Abelian group with infinite number of elements.

Example: Consider the integers $\mathbb{Z}$ under the multiplication operation. This is not a group. Why not?

Example: Let n be a positive integer let $\mathbb{Z}_n = \{0, 1, ..., n-1\}$. Then $\mathbb{Z}_n$ under addition modulo p is a group. We denote this additive group by $(\mathbb{Z}_n, +)$.

Example: If N is a positive number and if $\mathbb{Z}_N^* = \{x \in \{1, ..., N-1\} | gcd(x, N) = 1\}$. Then $\mathbb{Z}_N^*$ under multiplication modulo N is a group. We denote this multiplicative group by $(\mathbb{Z}_N^*, \cdot)$.

In $\mathbb{Z}_7^*$, notice that $3 \cdot 3 = 2$ since $9 \equiv 2 \pmod 7$.

Examples of Groups

Note that when p is a prime number, $\mathbb{Z}_p^* = \{1, 2, ..., p-2, p-1\}$.

Claim 19.3.

Let n be a positive integer. Then $(\mathbb{Z}_n, +)$ is a finite Abelian group.

All the examples on the previous slide were examples of Abelian groups.

There are many examples of groups that are not Abelian.

Example: Let $GL(2, \mathbb{R})$ be the set of all invertible, real-valued 2 by 2 matrices. This is a group under the operation of matrix multiplication but it is not an Abelian group since matrix multiplication does not commute in general. You can easily find two invertible matrices that do not commute under multiplication.

Subgroups

If $(G, \circ)$ is a group and $H \subseteq G$. Then H is a subgroup of G if $(H, \circ)$ is a group.

Example: The set $2\mathbb{Z}$, consisting of the even integers, is a subgroup of $(\mathbb{Z}, +)$.

Example: The set of odd integers is NOT a subgroup of $(\mathbb{Z}, +)$

Exercise: Let H be the subset of $GL(2, \mathbb{R})$ that have determinant 1. Is H a subgroup of $GL(2, \mathbb{R}, \cdot)$. Here the operation is matrix multiplication.

Group Exponentiation

Let G be a multiplicative group. That is, the group operation of G is specified multiplicatively.

For a positive integer m, define $g^m = \underbrace{g \cdots g}_{m \text{ times}}$ and $g^{-m} = (g^m)^{-1}$.

Define $g^0 = 1$, the multiplicative identity.

It's easy to see that $g^{-m} = (g^{-1})^m$.

For integers m, n we have $g^m \cdot g^n = g^{m+n}, (g^m)^n = g^{mn}, g^1 = g$.

Exponentiation can be defined using additive notation also, where $mg = \underbrace{g + \cdots + g}_{m \text{ times}}$.

Group Exponentiation

Example: Consider the multiplicative group $(\mathbb{Z}_7^*, \cdot)$. We see that $3^0 = 1, 3^1 = 3, 3^2 = 2, 3^3 = 6, 3^4 = 4, 3^5 = 5, 3^6 = 1$. For any group element, we can write it as some power of 3.

If $(G, \cdot)$ is a group and there exists an element $g \in G$ such that for each $h \in G$ there is an m such that $g^m = h$, then we say that G is a cyclic group and the element g is a generator of G.

If p is a prime, then the multiplicative group $\mathbb{Z}_p^*$ is a cyclic group.

If a group is a cyclic group, then it is an Abelian group but the converse is not necessarily true.

An example of an Abelian group that is not cyclic is $G = \mathbb{Z}_6 \times \mathbb{Z}_2$ under addition operation. Eg. $(3, 0) + (4, 1) = (1, 1)$.

Claim 19.4.

If $(G, \cdot)$ is a cyclic group, then it is an Abelian group.

Group Exponentiation

Fact: If G is a finite (multiplicative) group with $|G| = m$, then for any $g \in G$ it holds that $g^m = 1$.

The previous statement implies that for any $g \in G$ and any integer x, we have $g^x = g^{x \pmod m}$.

Example: Suppose we wanted to compute 3^{800} in $(\mathbb{Z}_{17}, \cdot)$. Note that order of $\mathbb{Z}_{17}$ is 16.

Since $800 = 0 + 50(16)$, then $3^{800} = 3^{0+50(16)} = 3^0 \pmod{17}$

This allows us to (more) efficiently do large exponent calculations.

Exercise: Compute $3^{1000} \pmod{100}$. You need to be careful when computing the order of the group $\mathbb{Z}_{100}^*$ since 100 is not prime. The order of this group is the number of elements < 100 whose gcd with 100 is 1.

Finding Generators in $\mathbb{Z}_p^*$

The following result is often very useful.

Theorem 19.5.

Suppose p is a prime and consider the group $(\mathbb{Z}_p^*, \cdot)$.

- *Select some $g \in \mathbb{Z}_p^*$ and compute $g^{(p-1)/q}$ for each prime divisor q of $p-1$.*
- *If $g^{(p-1)/q} \not\equiv 1 \pmod{p}$ for each q, then g is a generator of $\mathbb{Z}_p^*$.*

Often, you'll hear the term primitive root of p. A primitive root of p is equivalent to a generator of $\mathbb{Z}_p^*$.

Finding Generators in $\mathbb{Z}_p^*$

Example: Suppose $p = 19$. Select $g = 2$. Note that $p - 1 = 18 = 2 \times 3^2$ and therefore $q = 2, 3$. We see that

$$2^{(19-1)/2} = 2^9 \equiv 18 \not\equiv 1 \pmod{19},$$

$$2^{(19-1)/3} = 2^6 \equiv 7 \not\equiv 1 \pmod{19}.$$

Therefore, 2 is generator of $\mathbb{Z}_{19}^*$.

Note you can check this by computing $2^0, 2^1, \ldots 2^{18}$ and see if it's all the elements in $\mathbb{Z}_{19}^*$.

Exercise: Can you find another generator of $\mathbb{Z}_{19}^*$?

The Discrete-Logarithm Problem

Suppose G is a (multiplicative) cyclic group of order q with generator g.

Then the elements of G can be stated as $G = \{g^0, g^1, ..., g^{q-1}\}$.

Therefore, for every $h \in G$, there exists a unique $x \in \mathbb{Z}_q$ such that $h = g^x$.

We call x the discrete logarithm of h and write $x = \log_g h$.

The discrete-logarithm problem (DLP) is the problem where, given a (multiplicative) cyclic group G of order q with generator g and $h \in G$, to compute $\log_g h$.

The group $(\mathbb{Z}_p^*, \cdot)$ is typically the group used in the discrete-logarithm problem, where p is a large prime.

The Discrete-Logarithm Problem

This is one of the most-studied problems in computational number theory.

It's clear that the discrete-logarithm problem can be solved in $O(q)$ time and $O(1)$ space using an exhaustive search.

Typically, q is very large (say > 150 digits) and therefore an exhaustive search is not computational feasible.

Various algorithms have been developed for developed for the discrete-logarithm problem, but no polynomial-time algorithm is known.

When talking about polynomial-time, it is with respect to the number of bits needed to encode q (that is, polynomial in $\log_2 q$).

Diffie-Hellman Problem (DHP)

Suppose the following are given :

- p - a prime that determines the group $\mathbb{Z}_p^*$,
- g - a generator of $\mathbb{Z}_p^*$,
- g^a - the value of g raised to a fixed (but unknown) power a,
- g^b - the value of g raised to a fixed (but unknown) power b.

The Diffie-Hellman Problem (DHP) is the problem of computing g^{ab} in Z_p^*.

Note that if an adversary can solve an instance of DLP, the (s)he can solve the DHP.

To see this, the values of a, b are simply the disrete log of g^a, g^b respectively and since we can solve the DLP, we can compute a, b. Then it is trivial to compute g^{ab}.

Diffie-Hellman Key-Exchange

Diffie-Hellman Key-Exchange Protocol

In the Diffie-Hellman protocol, let $\mathscr{G}$ be a probabilistic polynomial-time algorithm that on input 1^n, outputs

- a cyclic group G,
- its order q (with the number of bits needed to represent q to be denoted by n) and
- a generator $g \in G$.

We now give the Diffie-Hellman key-exchange protocol.

Diffie-Hellman Key-Exchange Protocol

Construction 20.1 (Diffie-Hellan Key-Exchage Protocol).

1. *Alice runs* $\mathcal{G}(1^n)$ *to obtain* (G, q, g)*, a multiplicative group* G *with order* q *and generator* g*.*
2. *Alice chooses a uniform(ly random)* $x \in \mathbb{Z}_q$ *and compute* $h_A = g^x$*.*
3. *Alice sends* (G, q, g, h_A) *to Bob.*
4. *Bob receives* (G, q, g, h_A)*, chooses a uniform(ly random)* $y \in \mathbb{Z}_q$ *and compute* $h_B = g^y$*. Bob sends* h_B *to Alice and outputs the key* $k_B = (h_A)^y$*.*
5. *Alice receives* h_B *and outputs the key* $k_A = (h_B)^x$*.*

Diffie-Hellman Key-Exchange Protocol

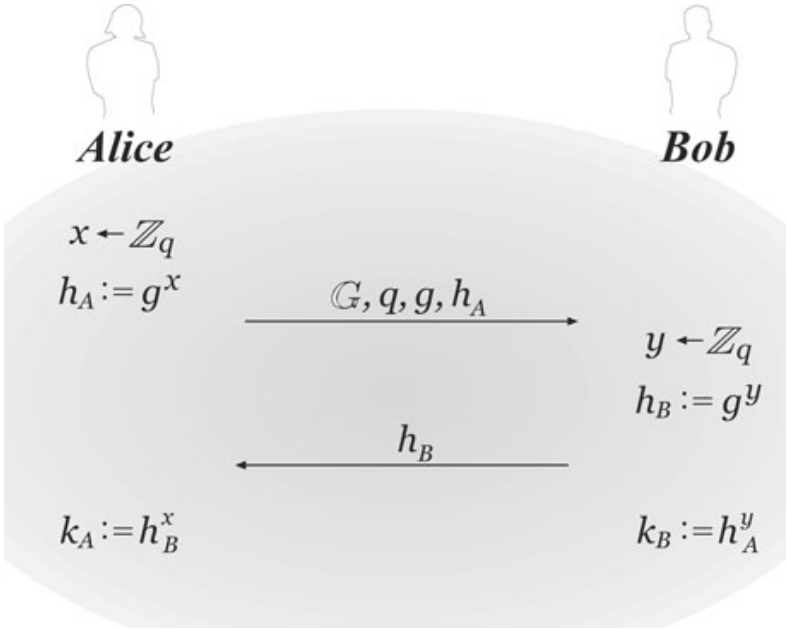

Figure: Diffie-Hellman key-exchange protocol

Diffie-Hellman Key-Exchange Protocol

Any adversary does not know x, y, but knows everything else.

In the protocol, Alice sends (G, q, g) to Bob. In practice, (G, q, g) is standardized and known to both Bob and Alice before the protocol begins.

The key k_B that is outputted by Bob is

$$k_B = (h_A)^y = (g^x)^y = g^{xy}.$$

The key k_A that is outputted by Alice is

$$k_A = (h_B)^x = (g^y)^x = g^{xy}.$$

Therefore Alice and Bob outputs the same key.

Diffie-Hellman Key-Exchange Protocol-Example

Example: Suppose $G = (\mathbb{Z}_{11}^*, \cdot)$ with order $q = 10$ and generator $g = 7$.

Alice choose the number $x = 3$ (randomly) and computes $h_A = g^x = 7^3 = 343 = 2 \pmod{11}$.

Alice sends $h_A = 2$ to Bob.

Bob choose the number $y = 6$ (randomly) and compute $h_B = g^y = 7^6 = 117,649 = 4 \pmod{11}$.

Bob sends $h_B = 4$ to Alice.

Alice takes Bob's result and computes the key $h_B^x = 4^3 = 9 \pmod{11}$. Similarly, Bob takes Alice's result and computes the key $h_A^y = 2^6 = 9 \pmod{11}$. The shared (secret) key is $k = 9$.

□

Security

The adversary's objective is to find the key $k = k_A = k_B$.

This can be accomplished if the discrete logs of h_A, h_B can be computed.

The importance of x, y being random cannot be overstated.

The order of the group needs to be very big. Otherwise, a brute force attack can be successfully used to compute discrete logs.

Security

But even if the discrete-logarithm problem is computational difficulty to solve, it doesn't imply the Diffie-Hellman protocol is computationally difficult to break as there may be other approaches that can determine the key without the knowledge of x, y.

For instance, the Diffie-Hellman protocol is vulnerable to a man-in-the-middle attack, which we'll describe in the next slides.

The protocol as stated here is not used in practice.

However, it is important because it demonstrates how asymmetric problems can be used to develop an approach for key distribution without the need for a private channel.

Man-in-the-Middle Attack

Consider the following (active) attack by adversary Malice on the Diffie-Hellman key-exchange protocol given earlier.

- Malice intercepts g^x from Alice and g^y from Bob.
  - Malice selects e with $1 < e < q$ and sends g^e to both Alice and Bob.
  - Alice now thinks that g^e is g^y and Bob thinks g^e is g^x.
- Alice computes what she thinks is $(g^y)^x$ but in fact she is really computing g^{ex}.
- Bob computes what he thinks is $(g^x)^y$ but in fact he is really computing g^{ey}.
- Malice computes $(g^x)^e$ (which is what Alice believes is the key) and $(g^y)^e$ (which is what Bob believes is the key).

Man-in-the-Middle Attack

When Alice sends an encrypted message (encrypted using the key g^{ex}) to Bob

- Malice can intercept it, decrypt it using key g^{ex} and re-encypt it with key g^{ey} and send it to Bob.
- Bob would decrypt the received ciphertext using key g^{ey}.

Similarly, Malice can read all messages sent by Bob to Alice.

This is an example of a man-in-the-middle attack.

This attack can be thwarted by ensuring keys are entity-authenticated (that is, verified as belonging to the correct person).

This can be done using digital signatures (to be discussed later).

Elementary Number Theory and RSA Cryptosystem

Introduction

We will introduce the fundamentals behind public-key encryption schemes and consider several well-known schemes.

These schemes are based on the "hardness" of solving certain number-theoretic problems.

In order to understand and appreciate these schemes, we need a bit of number theory.

We begin with a review of some basic number theory.

Euclidean Algorithm

Recall the Euclidean Algorithm for finding the greatest common denominator (gcd) of two positive integers a, b.

Algorithm 3: *EuclideanAlgorithm*

Input : a, b with $a > b$
Output: $\gcd(a, b)$

1 $r_0 \leftarrow a, r_1 \leftarrow b$
2 Compute q_1, r_2 such that $r_0 = q_1 r_1 + r_2$
3 $i \leftarrow 1$;
4 **while** $r_{i+1} > 0$ **do**
5 $\quad i \leftarrow i + 1$
6 $\quad$ Compute q_i, r_{i+1} such that $r_{i-1} = q_i r_i + r_{i+1}$;
7 **end**
8 **return** r_i

Euclidean Algorithm

Example: Compute $\gcd(216, 60)$.

$$216 = 3 \cdot 60 + 36$$
$$60 = 1 \cdot 36 + 24$$
$$36 = 1 \cdot 24 + 12$$
$$24 = 2 \cdot 12 + 0$$

The gcd of 216 and 60 is equal to 12

Euclidean Algorithm

Example: Compute $\gcd(55, 26)$.

$$55 = 2 \cdot 26 + 3$$
$$26 = 8 \cdot 3 + 2$$
$$3 = 1 \cdot 2 + 1$$
$$2 = 2 \cdot 1 + 0$$

The gcd of 55 and 26 is equal to 1

Extended Euclidean Algorithm

Given a positive integers b, n with $b < n$, how can we determine whether b has a multiplicative inverse modulo n.

That is, given positive integers b, n with $b < n$, we wish to determine if there exists some $a \in \mathbb{Z}_n$ such that $ab \equiv 1 \pmod{n}$.

In addition, if such an inverse exists, how can we find it and can we do it efficently?

This will an important component for the encryption and decryption process of the RSA cryptosystem.

This problem can be solved using the extended Euclidean algorithm.

Extended Euclidean Algorithm

Given values n, b as defined above, set $r_0 = n, r_1 = b$ and apply the Euclidean algorithm to get the values $q_1, q_2, ..., q_m$.

Now set up the the following recurrence relation:

$$t_0 = 0$$
$$t_1 = 1$$
$$t_j = t_{j-2} - q_{j-1} t_{j-1} \pmod{r_0} \quad \text{if } j \geq 2$$

Then the last equation computed by the Euclidean algorithm is

$$r_{m-1} = q_m r_m + r_{m+1}$$

where $r_{m+1} = 0$.

Extended Euclidean Algorithm

Claim 21.1.

Suppose n, b are positive integers such that $n > b$. If $gcd(n, b) = gcd(r_0, r_1) = 1$, then the multiplicative inverse of r_1 modulo r_0 exists and is given by t_m, where r_0, r_1, t_m are defined above.

Note that if $gcd(b, n) > 1$, then b^{-1} does not exist in the group $(\mathbb{Z}_n, \cdot)$.

The values t_j, where $j = 2, 3, ..., m$ can be computed using what is known as the extended Euclidean algorithm.

It is an extended version of the Euclidean Algorithm since is the Euclidean algorithm with some extra information computed (the t_j's).

Extended Euclidean Algorithm

Example: Compute 26^{-1} in $(\mathbb{Z}_{55}^*, \cdot)$.

Recall the Euclidean algorithm.

$$55 = 2 \cdot 26 + 3$$
$$26 = 8 \cdot 3 + 2$$
$$3 = 1 \cdot 2 + 1$$
$$2 = 2 \cdot 1 + 0$$

The gcd of 55 and 26 is equal to 1

Note: $r_0 = 55, r_1 = 26, q_1 = 2, q_2 = 8, q_3 = 1, q_4 = 2$ and $m = 4$

j	0	1	2	3	4
q_i	-	2	8	1	2
t_i	0	1	-2	17	-19

Since $\gcd(26, 55) = 1$ and $t_m = t_4 = -19$, the claim implies $(-19)(26) \equiv 1 \pmod{55}$.

Note that $-19 \equiv 36 \pmod{55}$. Therefore 26^{-1} modulo 55 is 36.

Extended Euclidean Algorithm

Example: Compute 28^{-1} in $(\mathbb{Z}_{201}^*, \cdot)$.

Recall the Euclidean algorithm.

$$201 = 7 \cdot 28 + 5$$
$$28 = 5 \cdot 5 + 3$$
$$5 = 1 \cdot 3 + 2$$
$$3 = 1 \cdot 2 + 1$$
$$2 = 2 \cdot 1 + 0$$

The gcd of 201 and 28 is equal to 1

Note: $r_0 = 201, r_1 = 28, q_1 = 7, q_2 = 5, q_3 = 1, q_4 = 1, q_5 = 2$ and $m = 5$

j	0	1	2	3	4	5
q_i	-	7	5	1	1	2
t_i	0	1	-7	36	-43	79

Since $\gcd(201, 28) = 1$ and $t_m = t_5 = 79$, the claim implies $(79)(28) \equiv 1 \pmod{201}$.

We conclude that 28^{-1} modulo 201 is 79.

Euler ϕ (Totient) Function

Suppose $a \geq 1$ and $m \geq 2$ are positive integers.

We say that a and m are relatively prime if $\gcd(a, m) = 1$.

The number of integers in $\mathbb{Z}_m$ that are relatively prime to m is denoted by $\phi(m)$.

That is, $\phi(m) = |\{x : \gcd(x, m) = 1, 1 \leq x < m\}|$.

The function ϕ is known as the Euler phi-function or the Euler totient function.

A well-known result is the following: Suppose $m = \prod_{i=1}^{n} p_i^{e_i}$, where the p_i's are distinct primes and $e_i > 0$ for $i = 1...n$. Then

$$\phi(m) = \prod_{i=1}^{n} (p_i^{e_i} - p_i^{e_i - 1}).$$

Euler ϕ (Totient) Function

Example: Suppose $m = pq$ where p, q are distinct primes.

Then $\phi(m) = (p-1)(q-1)$.

We'll make use of this example in the RSA cryptosystem.

□

Suppose $a \in \mathbb{Z}_m$, where m is not necessarily a prime number.

When does a^{-1} exists in the multiplicative group $(\mathbb{Z}_m, \cdot)$?

It turns out that a^{-1} exists if and only if $\gcd(a, m) = 1$.

Exercise : Let $a \in \mathbb{Z}_m$. Show that the equation $ab \equiv 1 \pmod{m}$ has a solution if and only if $\gcd(a, m) = 1$.

An Important Result

We state a couple important results that will be useful later on.

Theorem 5 (Fermat's Little Theorem).

Let p be a prime number and let a be an integer. Then

$$a^{p-1} \equiv \begin{cases} 1 \pmod{p} & \text{if } p \text{ does not divide } a, \\ 0 \pmod{p} & \text{if } p \text{ divides } a. \end{cases}$$

Another useful result in number theory is the following:

Theorem 6.

If p, q are distinct primes and $x \equiv a \pmod{p}, x \equiv a \pmod{q}$, then $x \equiv a \pmod{pq}$.

Exercise: Prove these two results.

Public-Key Encryption Schemes

A public-key encryption scheme is an encryption scheme where the key that is used by the encryption algorithm is different from the key that is used by the decryption algorithm.

The encyption key is made public while the decryption key is kept private.

The security of the scheme is typically based on the hardness of solving mathematical problem.

The concept of public-key encryption schemes was introduced by Diffie and Hellman in 1976.

The first public-key encryption scheme was introduced in 1977 by Rivest, Shamir and Adelman. This is the well-known RSA cryptosystem, which we'll consider now.

RSA Cryptosystem

The basic RSA cryptosystem is given as follows:

Construction 21.2 (RSA).

Let $n = pq$ where p, q are distinct odd primes. Let $\mathscr{M} = \mathscr{C} = \mathbb{Z}_n^*$. Define the keyspace $\mathscr{K}$ as

$$\mathscr{K} = \{(n, p, q, d, e) : n = pq, de \equiv 1 \pmod{\phi(n)}.$$

For a key $K = (n, p, q, d, e)$, define

$$\text{Enc}_K(x) = x^e \pmod{n}$$

and

$$\text{Dec}_K(y) = y^d \pmod{n}$$

The value e is often referred to as the encryption exponent)

RSA Cryptosystem

In the RSA cryptosystem, the values n and e are public while p, q, d are secret.

Note that since $n = pq, \phi(n) = (p-1)(q-1)$.

Suppose Bob wishes to send a secret message m to Alice. Here is how to use the RSA cryptosystem to do this.

1. Alice generates two large, distinct primes p, q and computes $n = pq$.
2. Alice selects e such that $\gcd(e, (p-1)(q-1)) = 1$.
3. Alice computes d satisfying $de \equiv 1 \pmod{(p-1)(q-1)}$. This is Alice's private key.
4. Alice publicly announces her public key consisting of n and e.
5. Bob computes $c = m^e \pmod{n}$ and sends it to Alice.
6. Alices takes c and computes $c^d \pmod{n}$ to recover m.

RSA Example

Example: Alice generates two distinct primes $p = 13, q = 31$ and computes $n = pq = 403$. Then
$\phi(n) = (p-1)(q-1) = 12 * 30 = 360$.

Alice chooses the encryption exponent $e = 17$. Note $gcd(17, 360) = 1$ and therefore 17 is a valid choice for e.

Using the extended Euclidean algorithm, Alice obtains $d \equiv 17^{-1} \pmod{360}$ to be 233. One can check that $17(233) \equiv 1 \pmod{360}$.

Alice publishes her public key $(e = 17, n = 403)$. Her secret key is $d = 233$. Note also that only Alice knows the values p, q.

RSA Example

The number 403 can be encoded using $\lceil \log_2 403 \rceil = 9$ bits and therefore the cryptosystem can work with messages of length $9 - 1 = 8$ bits.

Suppose Bob wishes to send the character "?" to Alice using ASCII encoding (and therefore "?" encodes to 63).

Bob computes $63^{17} \pmod{403}$ to get 280 and sends ciphertext 280 to Alice. The value $e = 17, n = 403$ form Alice's public key.

Upon receiving this, Alice computes $280^{233} \pmod{403}$ to get plaintext message 63. Recall that $d = 233$ is Alice's decryption exponent, which only she knows.

□

Exercise: Work out the details of the calculations in the above example.

Correctness of RSA

We now want to show that the cryptosystem is correct.

First off, we know that d exists because $\gcd(e, \phi(n)) = 1$ and we can compute it using the Extended Euclidean algorithm.

We'll now show that for any message $x \in \mathbb{Z}_n$, $(x^e)^d \equiv x \pmod{n}$.

Theorem 7.

In the RSA cryptosystem, the following statement holds for every $x \in \mathbb{Z}_n$:

$$(x^e)^d \equiv x \pmod{n}.$$

Correctness of RSA

Proof: Let $x \in \mathbb{Z}_n$. We may assume $x \neq 0$ since result is trivially true for $x = 0$. Our goal is to prove that

$$(x^e)^d \equiv x \pmod{n}.$$

We note that since $de \equiv 1 \pmod{(p-1)(q-1)}$, then there exists $t \in \mathbb{Z}$ such that $de = t(p-1)(q-1) + 1$.

We consider two possible cases:

Case 1: Suppose $\gcd(x, p) = 1$. Then by Fermat's Little Theorem, we have

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

Then $x^{(p-1)(q-1)t} \equiv 1 \pmod{p}$ and multiplying both sides by x, we get

$$x^{1+(p-1)(q-1)t} \equiv x \pmod{p}.$$

Correctness of RSA

Case 2: Suppose $\gcd(p, x) = p$. That is, x is a multiple of p.

Then the last congruence holds for this case since each side is congruent to 0 (mod p). That is

$$x^{1+(p-1)(q-1)t} \equiv x \pmod{p}$$

holds for this case also.

Therefore, for all values of x, we have

$$x^{de} \equiv x \pmod{p}.$$

Correctness of RSA

Using a similar argument, we can show that (in both cases).

$$x^{de} \equiv x \pmod{q}.$$

By Theorem 6, these two congruences along with the fact p, q are distinct primes imply that

$$x^{de} \equiv x \pmod{pq}$$

as desired. ■

The proof shows that in the RSA cryptosystem, the message x can be any value in $\mathbb{Z}_n$ (that is, x need not be strictly a member of $\mathbb{Z}_n^*$).

Security of RSA

The security of RSA relies on the assumption that factoring the value of n is computational hard. Recommended that the modulus n should be at least 2048 bits in length.

Given the public key (n, e), the problem of computing the decryption exponent d is computational equivalent to the problem of factoring n.

A small encryption exponent e should not be used if sending the same message to multiple receivers. Small exponents are also an issue when m is small.

Notice that the encryption algorithm is deterministic and therefore cannot be CPA-secure.

This can be remedied by padding the message with a random binary-string of a fixed length.

Encoding Elements of $\mathbb{Z}_n$

Let l denote the number of bits needed to encode n.

Binary strings of length $l - 1$ may be view as elements of $\mathbb{Z}_n$.

Binary strings of varying length may be padded to bring them up to correct length.