[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy
Scribed by: Manas Jitendrakumar Ingle (202151086)

Winter 2023-2024
Week 9

1 Public Key Cryptography

Public-key cryptography, or asymmetric cryptography, is the field of cryptographic systems that use pairs of related keys. Each key pair consists of a public key and a corresponding private key. Key pairs are generated with cryptographic algorithms based on mathematical problems termed one-way functions.

1.1 Diffie and Hellman Key Exchange Algorithm

Suppose we are having a symmetric key encryption setup, that is, Alice and Bob have same key K.

Alice can encrypt a message and send it to Bob and Bob can decrypt the ciphertext. The problem here is that the secret key has to be same with Alice and Bob. Otherwise, the decryption will never give the correct plaintext. The problem lies in sharing the secret key. Before 1976, there was only one mechanism to share this secret key; which is, you have to meet the other party secretly. In 1976, Diffie and Hellman proposed a Key Exchange mechanism and with their results, the domain of Public Key Cryptography began. It was published in IEEE Transactions on Information Theory.

Let us first recall the concept of Group and a cyclic group before going further in the Diffie and Hellman Key Exchange Algorithm. A set G along with an binary operation, (G, *) is called to be Group if it satisfies the following properties:

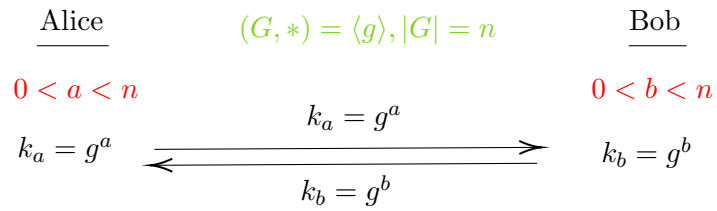
1. The operation * is closed under G, that is, if $a, b \in G$, then $a * b \in G$.
2. $*$ is associative on G, that is, $a * (b * c) = (a * b) * c \ \forall \ a, b, c \in G$
3. There is an element $e \in G$ called the Identity Element, such that $a * e = a = e * a \ \forall \ a \in G$.
4. For each $a \in G$, there exists an element $a^{-1} \in G$, called the inverse of a, such that $a * a^{-1} = e = a^{-1} * a \ \forall \ a \in G$.

(G, *) is a cyclic group if every element in G can be generated using only one element $g \in G$, that is,

$$\forall a \in G \ \exists \ g \in G, \text{ such that } a = g^i, \text{ where } i \in Z.$$

The element g is called the generator of G, and is denoted as $G = \langle g \rangle$.

Now, according to Diffie and Hellman Key Exchange Algorithm, Alice and Bob will agree on one cyclic group (G, *) over a public communication. It means that via a public communication, they will agree that they both will use the cyclic group $(G, *)$ whose generator is g.



The part that is written with green colour in the figure above is public where as the part written in red is secret to the person on whose side it is written. Alice selects a number a such that $0 < a < n$ and keeps it secret to herself. Similarly, Bob selects a number b such that $0 < b < n$ and keeps it secret to himself. Alice will compute $k_a = g^a$ and send it to Bob and Bob will compute $k_a = g^a$ and send it to Alice over the public channel. That is, Alice is making k_a public and Bob is making k_b public. Now, Alice and Bob have the following data with them.

Alice	Bob
a	b
g	g
$k_b = g^b$	$k_a = g^a$

Now, since, g^a and g^b belongs to same cyclic group. Given g^b, if we have a, then we can compute $(g^b)^a$. Therefore, Alice will compute $(g^b)^a$ and Bob will compute $(g^a)^b$. Therefore, Alice now has g^{ba} and Bob has g^{ab}. Since, a and b are integers, therefore,

$$\begin{aligned} & a \cdot b = b \cdot a \\ \implies & g^{ba} = g^{ab} \end{aligned}$$

Hence, Alice and Bob have same element g^{ab}. They can use this element to be their secret key and start the communication using the symmetric key encryption algorithms. The same key generated by Alice and Bob, that is, g^{ab} is called the Shared Secret Key.

We can see that Alice and Bob are exchanging some data over the public channel and they are able to establish a secret key. Since, a and b are only known to Alice and Bob respectively, that is, they are not shared publicly, hence, they are known as secret keys. However, $k_a = g^a$ and $k_b = g^b$ are shared by Alice and Bob to each other over a public channel. Hence, they are made public and are known as public keys. Both the parties have two keys, one public key and one secret key. Note that there may be some technique to compute $a(b)$ from $g^a(g^b)$, but since Alice (Bob) is not sharing $a(b)$, we are assuming that it is secret key for Alice(Bob).

Now, if we don't know the secret key of Alice (a), even if we know g^a and g^b, we will not be able to compute g^{ab}, which is the key used for communication between Alice and Bob. That means, without knowing the secret key of Alice or Bob, we will not get the shared secret key used for communication between Alice or Bob.

Now, we have g^x as public key and x as secret key. If we have a very good cyclic group (G, *), based on the properties of the group, finding x from g^x is computationally difficult. It is possible theoretically, but the amount of time it requires is exponential. This hard problem is known as Discrete Log Problem. Since, finding x from g^x for a good group is computationally hard and we are using this group for establishing the shared secret key, then the key establishment mechanism will be secure. Therefore, the security of Diffie-Hellman Key Exchange Algorithm relies on the fact that Discrete Log Problem is hard for certain groups.

One obvious way to compute x from g^x, as we know G, g and g^x, is to compute g^i for $1 \leq i \leq n-1$ and return i if we get $g^i = g^x$.

```
for i = 2 to N − 1 do
    if g^i == g^x then
        t = i;
        break;
    end
end
```

Algorithm 1: Brute Force Algorithm to find x from g^x

The complexity here will be equal to the size of the set G, i.e $|G|$. Let us say our group contains $2^{512} - 1$ elements, then getting x using this mechanism is impossible. The loop in this case will never terminate in practical time. Therefore, for the Discrete Log Problem to be hard on a group (G, *), the group must follow certain properties. These properties are mention below:

1. Size of the set G, i.e $|G|$, should be very large.
2. The group operation $*$ must be chosen carefully. Even if the group size is very large, you can still study the properties of the group operation and can find x from g^x very easily for certain group operators. For example, consider the cyclic group $G = (Z_p, +_p)$, p is a very large prime number and $G = \langle g \rangle$. If we try to compute g^i, it will be equal to:

$$g^i = g +_p g +_p ..... +_p g$$
$$g^i = i \cdot g$$

So, if we have generator of G, i.e g, g^i can be easily computed. Now, if we know g^i, finding i will be very easy and we will not have to search for i exhaustively. Multiplying both the sides with g^{-1} will give us:

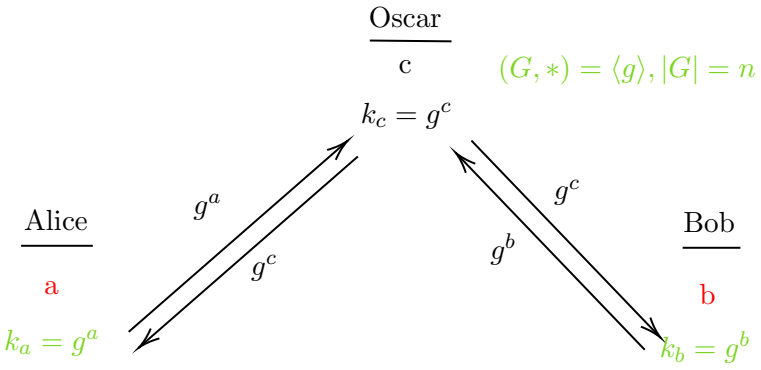
$$i = g^{-1} \cdot g^i \bmod \text{p}$$

We know g^i and we can compute g^{-1} in polynomial time using the Extended Euclidean Algorithm, which will definitely hold true because gcd(i, p) = 1 as p is prime. Hence, $+_p$ is not a good group operation.

1.2 Man in the Middle Attack on Diffie-Hellman Key Exchange Algorithm

Man in the Middle means there is a person who is listening to your communication and he has control over the data. Suppose you are writing a letter to your friend. You write the letter, put it in an envelope and visit the nearby post office to send it to your friend. Now, the responsibility of the post office is to transfer the letter to your friend. If the middle man, i.e. the post office, is corrupted, there will be problems and your message can be read or can be changed by the post office. In Whatsapp, you exchange certain data with its server, the server, in turn, send it to the intended person. If the Whatsapp server is corrupted, your data can be leaked.

Let's try to understand the Man in the Middle Attack on the Diffie-Hellman Key Exchange Algorithm.



Here, Oscar can capture the communication between Alice and Bob. Suppose if Alice has sent some message to Bob. Oscar can capture this message and can send a different message to Bob. Suppose Alice has sent g^a to Bob. Oscar will compute g^c as g and G are public. Oscar will intercept the g^a sent by Alice to Bob and send g^c to Bob. Since, Bob doesn't have any mechanism for authenticating that the message is coming from Alice, Bob will believe that Alice has sent her public key to him. However, this is not the case, Bob has actually recieved g^c (not g^a).
Bob will share his public key g^b to Alice. Again, Oscar will intercept this message and send g^c to Alice. Alice, again, doesn't have any mechanism to check if it is coming from Bob or not, so she will accept it.
With the data that Alice has, she can compute:

$$(g^c)^a = g^{ac}$$

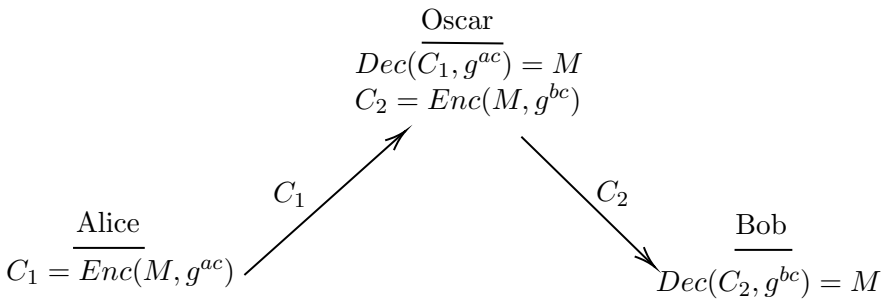
With the data that Bob has, he can compute:

$$(g^c)^b = g^{bc}$$

and with the data that Oscar have, he can compute:

$$(g^a)^c = g^{ac}$$
$$(g^b)^c = g^{bc}$$

Now, Oscar has two shared secret keys, one is same with Alice and the other is same with Bob. Alice and Bob will begin the communication as they are unaware about the middle man. Suppose Alice has sent a message to Bob.



Alice will encrypt the message using g^{ac} and send it. Oscar will receive this message and decrypt it using g^{ac}. Oscar will get the original message sent by Alice. Now, Oscar will encrypt this message using g^{bc} and send it to Bob. Bob on receiving the message will decrypt it using g^{bc} and will receive the original message sent by Alice. Both Alice and Bob will be unaware of the fact that Oscar is receiving all the communication between them. This is known as Man in the Middle Attack on Diffie-Hellman Key Exchange Algorithm.

In Whatsapp or Telegram, Diffie-Hellman Key Exchange is used. This exchange is performed by the middle person, i.e the Whatsapp or Telegram. If they want they can do anything with the data. However, there are certain mechanisms which they implement to provide security.

If we wish to compute the shared secret key g^{ab}, we have to compute g^a and g^b where $|G|$ is large and also a and b should be large. In fact, if you will use any pseudo-random selector from 1 to (n-1), it is highly guaranteed that it will be in the middle of 1 and (n-1). Hence, if $|G| = 2^{512}$, then a and b will be in the order of 256 bits. The problem here is how to compute g^a and g^b efficiently. It is not possible to compute g^a and g^b by running a loop and multiplying by g in each iteration. This is practically impossible. Therefore, to compute g^a and g^b, we use the Square and Multiply Algorithm. According to the algorithm, suppose we want to calculate x^c. The binary representation of c is $c_{l-1} \dots c_1, c_0$. Then,

$c = \Sigma_{i=0}^{l-1} c_i \cdot 2^i$

$x^c = x^{\Sigma_{i=0}^{l-1} c_i \cdot 2^i} = \Pi_{i=0}^{l-1} x^{c_i \cdot 2^i}$

$x^c = x^{c_0 \cdot 2^0} \dots x^{c_{l-1} \cdot 2^{l-1}}$

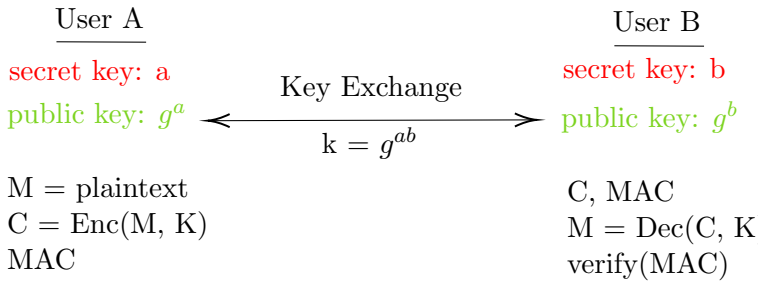
These are just $log(c)$ multiplications, hence, we can compute x^c in logarithmic time of c.

Input: x and c
Z = 1;
for $i = l - 1$ *to* $N0$ **do**
  $Z = Z^2$;
  if $c_i == 1$ **then**
    Z = Z * x;
  end
end
return Z;

Algorithm 2: Square and Multiply Algorithm to find x^c

Let us take and example and calculate 3^5. Therefore, x = 3 and c = 5 = $(101)_2$. Initially Z = 1, i = 2 (since l = 3). First iteration of for loop, $Z = Z^2 = 1$, since $c_2 = 1$, $Z = Z*x = 3$. Second iteration of for loop, Z = 3, i = 1. $Z = Z^2 = 9$, since $c_1 = 0$, Z will not be multiplied by x. Last iteration of for loop, Z = 9, i = 0. $Z = Z^2 = 81$, since $c_0 = 1$, $Z = Z*x = 81*3 = 243$. Z = 243 = 3^5 is returned.

Using the Square and Multiply Algorithm g^a and g^b can be computed efficiently in $log(a)$ and $log(b)$ time complexity respectively.



Now, if we have a Man in the Middle Attack, then we can have a communication setup where the MAC can also be incorporated. The MAC authenticates the source of the message as well as the correctness of the message. So, now we can authenticate the source, hence, the Man in the Middle Attack will not be possible now.

2 RSA (Rivest Shamir Adleman) Encryption

It is the first public key encryption algorithm. Before beginning any discussion on RSA encryption, let's first recall a few concepts.

- The Euler's Totient Function $\phi(n)$ denotes the number of integers less than n that are co-prime to n, i.e number of x such that gcd(x, n) = 1 where $1 \leq x \leq n-1$
- Let there be a set $S = \{x \bmod m\}$ such that $|S| = m$.

  $$S = \{r_1, r_2, \ldots, r_m\}$$

  All the elements in the set S are unique (usually they are from 0 to m-1). Assume an integer a such that $gcd(a, m) = 1$. Let's say that there is another set S_1 such that,

  $$S_1 = \{ar_1 \bmod m, ar_2 \bmod m, \ldots, ar_m \bmod m\}$$

  Since, $\{r_1, r_2, \ldots, r_m\}$ are different elements and gcd(a, m) = 1, it can be concluded that $\{ar_1 \bmod m, ar_2 \bmod m, \ldots, ar_m \bmod m\}$ will also be m unique elements. This can be proved using contradiction. Suppose, if $ar_i = ar_j$ for $r_i \neq r_j$. Therefore,

  $$ar_i \equiv ar_j \bmod m$$

  Since, gcd(a, m) = 1, therfore, $1 = ab + ms$(from Bezout's Identity). Therefore, there exists an integer b such that $ab \equiv 1$ mod m. The value of b is known as multiplicative inverse of a and it can be found using Extended Euclidean Algorithm. Therefore, on multiplying the above equation by b on both sides gives us,

$$b \cdot a \cdot r_i \equiv b \cdot a \cdot r_j \text{ mod } m$$

$$r_i \equiv r_j \text{ mod m } (\because ab \equiv 1 \text{ mod m})$$

Hence, it is a contradiction to our initial assumption that $r_i \neq r_j$. Hence, elements in the set S_1 will be unique iff gcd(a, m) = 1.

2.1 Euler's Theorem

If gcd(a, m) = 1, then $a^{\phi(m)} \equiv 1$ (mod m).
Let us assume, that we have a set S, such that

$$\text{S} = \{ \text{ x} \mid \text{gcd(x, m)} = 1\}$$
$$\text{S} = \{s_1, s_2, s_3, s_4, ..., s_{\phi(m)}\}$$

Let us consider gcd(a, m) = 1 and create another set S_1 such that

$$S_1 = \{ \ as_1, as_2, as_3..., as_{\phi(m)} \ \}$$

if $as_i \equiv as_j$ (mod m)
$\Rightarrow s_i \equiv s_j$(mod m)
Given that gcd(a, m) = 1 and $b.a \equiv 1$ (mod m)

$$|S| = \phi(\text{m})$$
$$|S_1| = \phi(\text{m})$$

Since a is co-prime with m and s_i is also co-prime with m, then there must be some correspondence between elements of S and S_1.

$$s_i \equiv as_j \text{ (mod m)}$$

Let us now take product on both sides

$$\prod_{i=1}^{\phi(m)} s_i \equiv \prod_{j=1}^{\phi(m)} as_j (mod\ m)$$

$$\Rightarrow \prod_{i=1}^{\phi(m)} s_i \equiv a^{\phi(m)} \prod_{j=1}^{\phi(m)} s_j (mod\ m)$$

Since gcd(s_i, m) = 1, each s_i will have multiplicative inverse under mod m. So, after simplifying we get,

$$a^{\phi(m)} \equiv 1 (mod\ m)$$

2.2 Fermat's Theorem

If P is a prime number and P does not divide a(means that P is co-prime to a), then

$$a^{P-1} \equiv 1(mod\ P)$$

Using Fermat's theorem,

$$\Rightarrow a^P \equiv a(mod\ P)$$

Note:
If P—a(P divides a), then

$$\begin{gathered} \text{a} \equiv 0 \bmod \text{P} \\ \Rightarrow a^P \equiv 0 mod\ P \\ \Rightarrow a^P \equiv a mod\ P \end{gathered}$$

But the Fermat's theorem will not hold when P$\|a$.

2.3 RSA Cryptosystem

Few facts:

- g(a, m) = 1, then $a^{\phi(m)} \equiv 1(mod\ m)$
- $a^{P-1} \equiv 1(mod\ P)$

Now let us understand the components of RSA

1. n = pq, where p, q are primes
2. Plaintext space = $_n$
   Ciphertext space = $_n$
3. Key space = {K = (n,p,q,e,d) | $ed \equiv\ (mod\phi(n))$}
4. Encryption :

$$\begin{gathered} \text{E(x, K)} = \text{c} \\ \text{c} = \text{E(x, K)} = x^e (\text{mod n}) \end{gathered}$$

5. Decryption :

$$\begin{gathered} \text{Dec(c, K)} = \text{x} \\ \text{c} = \text{Dec(c, K)} = c^d (\text{mod n}) \end{gathered}$$

We know that e and d are related as :

$$\text{ed} \equiv 1 \bmod \phi(\text{n})$$
$$\Rightarrow \text{ed - } 1 = \text{t}.\phi(\text{n})$$
$$\Rightarrow 1 = \text{ed} + t_1.\phi(\text{n})$$
$$1 = \gcd(\text{e, Q(n)}) = \text{ed} + t_1.\phi(\text{n})$$

Encryption:

$$\text{c} = x^e (\text{mod n})$$

Decryption:

$$\text{x} = c^d (\text{mod n})$$
$$c^d = (x^e)^d \bmod \text{n}$$
$$c^d = x^{ed} \bmod \text{n}$$
Now using ed = 1 + t.ϕ(n) from above
$$c^d = x^{1+t.\phi(n)} \bmod \text{n}$$
$$c^d = \text{x}.x^{t.\phi(n)} \bmod \text{n}$$
Since p, q are primes and n = pq, then ϕ(n) = (p-1)(q-1)
$$c^d = \text{x}.x^{t[(p-1)(q-1)]} \bmod \text{n}$$
Finally,
$$c^d = \text{x}.x^{t[(p-1)(q-1)]} \bmod (\text{pq})$$

Now let us simplify the part $x^{t[(p-1)(q-1)]}$ mod (pq), where x $\in$
We check $x^{t[(p-1)(q-1)]}$ mod p
$\equiv {x^{p-1}}^{t(q-1)}$ mod p
$\equiv 1(\text{mod p})$ [As $x^{p-1} \equiv 1 mod\ p$]
Now we check $x^{t[(p-1)(q-1)]}$ mod q
$\equiv {x^{q-1}}^{t(p-1)}$ mod q
$\equiv 1(\text{mod p})$ [As $x^{q-1} \equiv 1 mod\ q$]
We finally have,

$$x^{t[(p-1)(q-1)]} \equiv 1(\text{mod p})$$
$$x^{t[(p-1)(q-1)]} \equiv 1(\text{mod q})$$
$$\Rightarrow x^{t[(p-1)(q-1)]} \equiv 1(\text{mod pq})$$

Substituting the above result,

$$c^d = \text{x}.x^{t[(p-1)(q-1)]} \bmod (\text{pq})$$
$$c^d = \text{x}.1 \bmod (\text{pq})$$
$$c^d = \text{x} \bmod (\text{pq})$$

Hence, our decryption is successful.

Now let us consider a scenario where Alice is trying to communicate with Bob. Here, two keys play the main role-one is public key and the other is secret key.
Here, Bob is encrypting the message and sending it to Alice and Alice has both the keys. Public key is known to Bob but the secret key is not known to Bob

Alice

n = pq and p, q are large prime numbers
ed $\equiv$ 1 (mod ϕ(n))
She chooses e
Public key of Alice = (n, e)
She can generate d using extended euclidean algorithm
Secret key of Alice = (p, q, d)
Bob
Now Bob selects a message x from $_n$
x
He knows n, e for Alice, so he can encrypt

y = $x^e(modn)$
Now the message y is sent to Alice, she can decrypt it with her secret key as :

$$x = y^d \text{ (mod n)}$$

Now, if we were given n, how can we find p and q in polynomial time?
We can run a loop from 2 to $\sqrt{n}$, and everytime we calculate n % i, if it is zero, it means we have found a factor, then we check if the factor is prime. If yes, then we divide n by p, to get q. Getting the prime factors of a number n is computationally hard problem when n is large.
Note: If we are able to compute p,q from n, then we will be able to compute ϕ(n). And then we already have e, so we will be able to find d by extended euclidean algorithm and security of RSA will be broken. So, RSA is based on the fact that factorization problem is hard.

2.3.1 RSA Problem

We have public key (n, e) and c. If from this we can find x (c = x^e), we will be able to break security of RSA.
We have an algorithm to solve the RSA problem, i.e., it can find the decryption without the factorization. Is this true? **Note:** If we can break the RSA, there is no guarantee that we can find the factors. But if we have the factors, we can always Break the RSA. But the reverse is not always true. So RSA is secure under two assumptions:

- factorization is hard
- decryption is hard

Note : Public key encryption is generally heavy because of x^e and c^d operation. The exponential operations are heavy. So they are usually avoided.