[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2022-2023
Scribed by: Bhargavi Kamble (202051048) — Lecture (Week 1)

Cryptography: Where we develop algorithms to provide security.
Crypto analysis: Where we try to break the security of the designed algorithm.
Cryptology: Where data communication and storage is secured using **Cryptography** and **Crypto analysis**. In cryptology we first design an algorithm and then we analyze it. We've to check that the security the algorithm is claiming is true or also find vulnerabilities, and in order to do that we perform crypto analysis. Without crypto analysis you can't make a secure scheme and also you can't determine the security of the scheme.
Once the algorithm is analyzed it can be approved by **NIST**.
NIST: Standardizes cryptography algorithms.
Example: If you want to store your ATM pins such that they are secured you can store them in the form of cipher text as mentioned below. Here we are using addition function to change the original number to a different value.

Sr. no:	**Plain text**	**Function and Secret key**	**Cipher text**
1	PIN1	+ X	Y1
2	PIN2	+ X	Y2
3	PIN3	+ X	Y3

Encryption: The process of converting readable data to unreadable is called as encryption. Encryption uses encryption function and secret key to convert plain text to cipher. Cipher texts are still readable but it doesn't make sense like plain text (The context of the original meaning can't be identified).

$$\mathbf{E(P,K) = C}$$

(here P is the plain text K is the secret key E is the encryption function and C is the cipher text.)

Decryption: The process of converting unreadable data to readable is called as decryption. Decryption uses decryption function and secret key to the cipher to original text.

$$\mathbf{D(C,K) = P}$$

The Cryptology can be divided into two types:
1. **Symmetric key cryptology**: It consists of only single key. The same key can be used for encryption as well as decryption.
2. **Public key cryptology**: It consists of two keys, one is public and other is the secret. The public key is known to everyone whereas secret key is only known to you.

$$\mathbf{E(P,Ek) = C}$$
$$\mathbf{D(C,Dk) = P}$$

One algorithm cannot provide all security services. Different algorithms are used for different purposes. Using all the algorithms we can define a protocol that protocol can cover all the security services.

Cryptography provides:

1. **Confidentiality**: Should not be seen by undesired people. 2. **Integrity**: Information should not be altered. 3. **Authentication**: Information should be from the correct source. 4. **Non-repudiation**: A mechanism to prove to prove that the sender really sent that message.
Text book definitions:
1. **Confidentiality**: Confidentiality is the protection of transmitted data from passive attacks. With respect to the content of a data transmission, several levels of protection can be identified.
Confidentiality = Secrecy
1. Plain text = Original message.
2. Encryption Algorithm = Function.
3. Cipher text = Unreadable form of the plain text.
4. Decryption algorithm = Function.
5. Encryption key = key.
6. Decryption key = key.
2. **Integrity**:A connection-oriented integrity service, one that deals with a stream of messages, assures that messages are received as sent, with no duplication, insertion, modification, reordering, or replays. The destruction of data is also covered under this service. Thus, the connection-oriented integrity service addresses both message stream modification and denial of service.
3. **Authentication**: The authentication service is concerned with assuring that a communication is authentic.
4. **Non-repudiation**:Non-repudiation prevents either sender or receiver from denying a transmitted message. Thus, when a message is sent, the receiver can prove that the alleged sender in fact sent the message. Similarly, when a message is received, the sender can prove that the alleged receiver in fact received the message.

Ceasar Cipher

This Cipher is named after Julius Ceasar. The result is obtained for shifting the letters of the message by an agreed number. Here agreed number = 3.
The letters are numbered as A=0, B=1, C=2.... and so on.
Consider the word "INTERNET", this is our plain text.
Here we'll shift all the letters to the right side by 3 places (secret key/agreed number) therefore, the resultant word will be "LQWHUQHW"(Cipher text).
To decrypt the word we can subtract and move the letters to left side by 3 places which will again give us the original word "INTERNET".

Function:

f: A $\to B$ it is a relation between the elements of A and B with the property that if a,b $\in$ A and a=b then f(a)=f(b).

One to one function:

If f(a)=f(b) then a = b.

Onto function:

f: A $\to B$ then $\forall$ b $\in$ B *exists* a $\in$ A such that f(a) = b.

Bijective function:

f: A $\rightarrow B$ is bijective function if and only if f is one to one and onto.

Permutation:

Let π be a permutation on a set S then $\pi : S \rightarrow$ S is a bijection from S to S. π: $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 4 \end{pmatrix}$

One way function:

f: X $\rightarrow Y$ is called a one way if x $\in$ X it is easy to compute f(x) but given f(x) it is difficult to calculate x.
e.g: Given p and q are large prime numbers. We have a computation N = p * q which is easy. Given N find p, q S.T N = p * q, this operation is hard.

Substitution box:

Here basically mapping is done. S : A $\rightarrow B$ with $|B| <= |A|$. For example, S: $\{1, 2, 3, 4\} \rightarrow \{1, 2, 3\}$. Therefore, here S(1) = 1, S(2) = 2, S(3)=3, but S(4) = 1... and so on.

Transposition Cipher:

We have M = $m_1.m_2.m_3..m_t$ as plain text of length t. e is the permutation on t elements (secret key). We'll perform the encryption in the following way:
Encryption:

$$\mathbf{C}= m_{e1}m_{e2}m_{e_3}...m_{et} = c_1c_2c_3...c_t$$

Here C is the resultant Cipher text.

Decryption:

$$\mathbf{M}= c_{e^{-1}(1)}c_{e^{-1}(2)}..c_{e^{-1}(t)}$$

Here e is the symmetric key.
Example: Let plain text be CAESAR = $m_1m_2...m_6$
Here secret key e = $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 4 & 1 & 3 & 5 & 2 \end{pmatrix}$ by performing the encryption we'll get the resultant cipher as "RSCEAA".
Here for decryption we'll take d = e^{-1}
Therefore, e^{-1} will be $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 6 & 4 & 2 & 5 & 1 \end{pmatrix}$ and therefore, the plain text will be "CEASAR".

Substitution Cipher:

The only difference between permutation cipher and substitution cipher is that in permutation cipher the size of the domain and co-domain is same but in substitution it can be different.
Let M = $m_1.m_2.m_3..m_l$, A = {a,b,c,...z}, $m_i \in A$. e is the substitution from A to A.
Encryption can be done as C = $e(m_1)e(m_2)e(m_3)....e(m_l)$
Example: Let e(A) = Z, e(B) = D, e(C) = A. Therefore, plain text = ABC and cipher text = ZDA.

Affine Cipher:

It's a good design but not secure cipher.We'll map all the alphabets from 0 to 25. A basic requirement of any encryption algorithm is that it be one-to-one. . A = 0, B = 1, C = 2, ... Z = 25. $A \rightarrow Z_{26}$, k is the secret key = (a,b) $\in Z_{26}$ X Z_{26}
Encryption function:
e(x,K) = (a.x + b) mod 26 = c
Decryption function:
d(c,k)=((c-a).a^{-1})mod26
a * a^{-1}= 1 mod 26
Steps:
Z_{26} = {1,2,3...25}
Secret key:
k = (a,b) $\in$ Z$_{26}$
gcd(a,26)=1
Encryption: C = E(x,k) = (a.x + b) mod 26
Decryption: x = D(c,k) = (c-b).a^{-1} mode 26

Playfair Cipher:

Multiple-letter encryption cipher is the Playfair, which treats digrams in the plaintext as single units and translates these units into ciphers. o normalize the plot, the number of occurrences of each letter in the ciphertext was again divided by the number of occurrences of e in the plaintext. The resulting plot therefore shows the extent to which the frequency distribution of letters, which makes it trivial to solve substitution ciphers, is masked by encryption. If the frequency distribution information were totally concealed in the encryption process, the ciphertext plot of frequencies would be flat, and cryptanalysis using ciphertext only would be effectively impossible.
Example: Secret key = Playfair example

P	L	A	Y	F
I	R	E	X	M*
B*	C	D*	G	H
K	N	O*	Q	S
T	U	Y	W	Z

Plain text = HIDE
HI = BM
DE = OD
CIPHER = BMOD

[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2022-2023
Scribed by: Bhargavi Kamble (202051048) — Lecture (Week 2)

Playfair Cipher:

Multiple-letter encryption cipher is the Playfair, which treats digrams in the plaintext as single units and translates these units into ciphers. o normalize the plot, the number of occurrences of each letter in the ciphertext was again divided by the number of occurrences of e in the plaintext. The resulting plot therefore shows the extent to which the frequency distribution of letters, which makes it trivial to solve substitution ciphers, is masked by encryption. If the frequency distribution information were totally concealed in the encryption process, the cipher text plot of frequencies would be flat, and cryptanalysis using cipher text only would be effectively impossible.
Example: Secret key = Playfair example
I=J

P	L	A	Y	F
I	R	E	X	M*
B*	C	D*	G	H
K	N	O*	Q	S
T	U	Y	W	Z

Plain text = HIDE
HI = BM
DE = OD
CIPHER = BMOD

If the length of the message is of odd length we can add extra X to make it even.
For e.g: we have the text ”TEA” we can take it as ”TEAX” to make it of even length.
If there are repeated characters in the plain text there too we can add a extra X between two repeated letters.
For e.g: We’ve the text ”BALL” the message can be taken as ”BALXLX”.
1.If both the letters are the same (or only one letter is left) add an ’X’ after the first letter. Encrypt the new pair and continue.
2. If the letters appear on the same row if your table replace them with the letter to their immediate right respectively.
3. If the letters appear on the same column of your table, then replace them with the letters immediately below respectively.
4. If the letters are not in the same row or column then replace them with the letters on the same row respectively but at the other part of corners of the rectangle defined by the original pair. The order is important - the first letter of the encrypted pair is one that lies on the same row as the first letter of the plain text pair.

Hill Cipher:

We have an matrix of n*n dimensions A = $(a_{i,j})_{n*n}$. Here, A is the secret key and plain text M = $(m_1 m_2 m_3 .... m_n)$. In order to get the encrypted text we multiply our plain text with the invertible matrix i.e. the secret key.
Encryption:

$$\mathbf{C = A*M}$$

(here A is the invertible matrix and M is the given plain text.)

Decryption:

$$\mathbf{M} = A^{-1} * \mathbf{C}$$

Kerckhoff's Rule:

Design has to be public. You can't hide the algorithm, it has to be known to everyone. If we hide the algorithm many people can't use it and there is possibility that our algorithm has some weakness.

Shannon's notion of perfect secrecy:

The cipher text is visible to everyone the main motivation is that it should not be able to decrypt by anyone. Perfect secrecy says that we'll be able to get the cipher text and even after getting it one should not be able to understand it or able to decrypt it if we achieve it our algorithm has achieved perfect secrecy.If your algorithm is perfectly secure the cipher text will not reveal **any** information. If we have the message the message encoded in 1 bit. So the probability that the message is 0 is 0.9 our algorithm should be built in such a way it should not have any extra advantage over that 0.9 of probability.

1. E = Encryption algorithm.
2. M = Message.
3. C = Cipher text.

E will be providing perfect secrecy if and only if the cipher text does not reveal any information regarding the plaintext/message.
E will be providing perfect secrecy iff the cipher text does not reveal any information regarding the plain text/message.

$$\textbf{Pr [M=m} \mid \textbf{C =c] = Pr [M=m]}$$
$$\textbf{Pr [message}\mid \textbf{ciphertext] = Pr [message]}$$

The probability of the message given the cipher text should remain exactly same as the probability of the message. Even if the cipher text is known it should not give any extra advantage.

OTP (One Time Padding):

It's not same as the one time password. It provides perfect secrecy but we don't use it practically.

The symmetric key cipher is of two types:

1.Block cipher.
2.Stream Cipher.

Block cipher:

As the name suggest it has blocks. In block cipher given any message it will be divided into various blocks of fixed length. If we have 1 TB message and I've an algorithm that can encrypt 56 bits then we'll divide the message into 56 blocks. The encryption will be done in block wise manner, we'll divide the message into various blocks of fixed size and then do encryption on each block.
eg: We have the message M it can be divided as:
M = $m_1||m_2||m_3||..||m_l$
C = Enc(m_0,k)||Enc(m_1,k) ||...||Enc(m_l,k)
C = $C_0||C_1||C_2...||C_l$

Stream Cipher:

Here the message will be encrypted bit wise. We have a algorithm which will be generating some bit from z0 to z1 and we'll xor it with the message to get the cipher.
M = $m_0....m_l$ here $m_0..m_l \quad \in 1, 2$
Encryption will be done bit wise.
Here C will be given by: C = $(m_0 \bigoplus z_0, ...m_l \bigoplus z_l)$
Therefore, C = c0...cl will be our cipher text.
For decryption we'll have the same $z_i's.$ Decryption of the ciphertext can happen in a manner similar to how the plaintext encryption occurs. This time, instead of the data and keystream being XOR-ed, the ciphertext and the keystream are XOR-ed.
In stream cipher we can encrypt a very long cipher which is not the case of block cipher. We can prefer block cipher for small messages.

Product Cipher:

A product cipher combines two or more transformations in a manner intending that the resulting cipher is more secure than the individual transformation. It is better in terms of security and as well as efficiency.

Substitution Permutation Network:

It is a product cipher based on substitution box and permutation box. We have a substitution box which has been mapped from n bit to m bit.
S = $\{0, 1\}^n \rightarrow \{0, 1\}^m$
The permutation box too is mapped from 0 to m_r -1 to 0 to m_r -1.
P = $\{0, m_r - 1\} \rightarrow \{0, m_r - 1\}$
We have a message of nr length. We divide that message into r blocks where each block will have n bit to pass the message from S box we will have n bit output. In the P box we'll have mr number of bits.

Feistel Network:

We've a plain text of 2n bits it'll be of even length. We will divide the message into two parts the left part will be called as l_0 and the right part will be called as r_0 both of them will be of n bits each. We can use right shift and left shift operators to get the right and left side of the message respectively. We've a round function called as F. It'll take two inputs of n and l bits and the output will be of n bits. These n bit will be xored with l_0. Whaever r_0 is present on n bit that'll be added in the l_0 as it is.

$$\mathbf{f: \{0,1\}^n * \{0,1\}^{\,l} \rightarrow \{0,1\}^n}$$

Therefore, here
$l_1 = r_0$
$r_1 = l_0 \bigoplus f(r_0, k)$
We can obtain the cipher text C as C = $l_1||r_1$
K is the secret key here.
Length of k = l bits.
Decryption:
$r_0 = l_1$
$l_0 = r_1 \bigoplus f(l_1, k)$
Here inversion of f is not required.

Iterated block cipher:

An iterated block cipher is block cipher involving the sequential repetition of an internal function (round function). The paramaeters include number of rounds r, the block size n, and the round keys k$_i$ of length l from the original secret key K. We've different round keys as k_1, k_2, k_3. F is the round function, P is the plain text block and K is the secret key. So the encryption will take place from by the following manner.

$$\text{P} \rightarrow \boxed{F} + k_1 \rightarrow \boxed{F} + k_2 \rightarrow \boxed{F} + k_3 \rightarrow \text{C}$$
$$\text{G(k)} \rightarrow k_1, k_2, k_3.$$

Here, G(k) is the key scheduling function whereas k_1, k_2, k_3 are the round keys.

One time padding:

OTP provides perfect secrecy under some conditions.
Here encryption is given by:
Enc(P,K) = P $\bigoplus$ K = C
Here encryption is given by:
Dec(C,K)= C $\bigoplus$ K = P Here the proability of the message is:
Pr [M=m | C =c] = Pr [M=m]
Pr [message| ciphertext] = Pr [message]
This algorithm works under cetain conditions:
1. The secret key K cannot be used to encrypt two messages.
2. length(k)¿=length(P)
3. K is uniformly selected from the key space.

[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2022-2023
Scribed by: Bhargavi Kamble (202051048) — Lecture (Week 3)

One time padding:

OTP provides perfect secrecy under some conditions.
Here encryption is given by:
Enc(P,K) = P $\bigoplus$ K = C
Here encryption is given by:
Dec(C,K)= C $\bigoplus$ K = P Here the proability of the message is:
Pr [M=m | C =c] = Pr [M=m]
Pr [message| ciphertext] = Pr [message]
This algorithm works under cetain conditions:
1. The secret key K cannot be used to encrypt two messages.
2. length(k)¿=length(P)
3. K is uniformly selected from the key space.

OTP on one bit:
Here message m $\in$ {0,1} and key K $\in$ {0,1}
Pr[m = 0] = P
Pr[m = 1] = (1-P)
Pr[k = 0] = 1/2
Pr[k = 1] = 1/2
Encryption:

$$c = m \bigoplus k$$

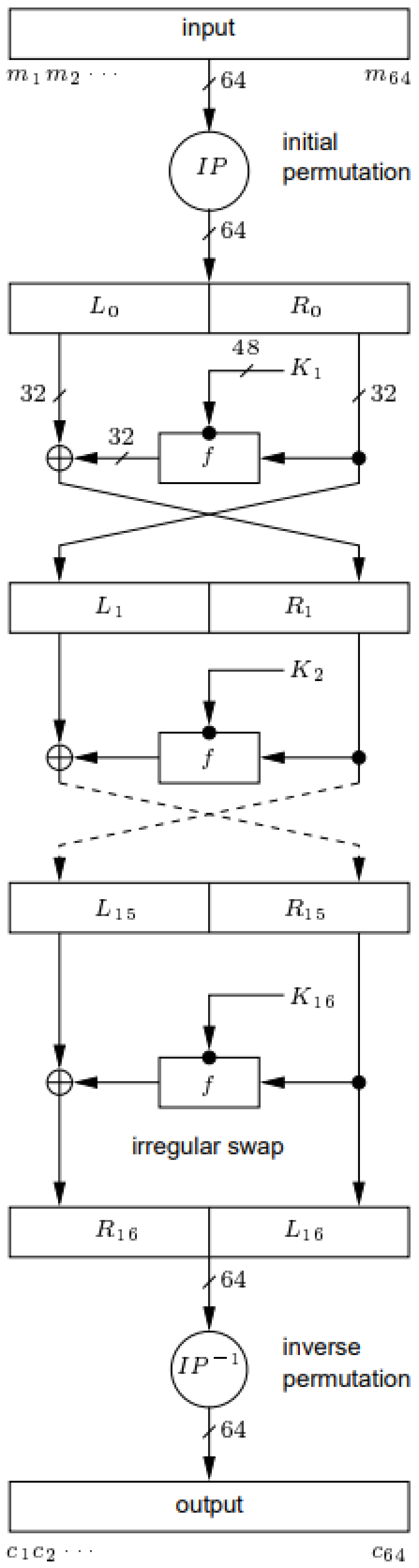
c=0 => {m=0,k=0} U {m=1,k=1}
Pr[C = 0] = Pr[m = 0, k = 0] + Pr[m = 1, k = 1]
= Pr[m = 0] * Pr[k = 0] + Pr[m = 1] * Pr[k = 1]
= (p * 1/2) + ((1 - p) * 1/2)
= 1/2
similarly,
Pr [C =1] = 1/2
we know,
Pr (A / B) = Pr (AB) / Pr (B)
Pr (AB) = Pr (B / A) * P (A)
Pr [m = 0 | c = 0] = Pr [m = 0, c = 0] / Pr [c = 0]
= Pr [m = 0 | c = 0] * Pr [m = 0] / 1/2
= Pr [k = 0] * Pr [m = 0] / 1/2
= 1 / 2 * Pr [m = 0] / 1/2 = Pr [m = 0]
Therefore, it provides perfect secrecy.

Conditions:
1. M1 $\bigoplus$ k = C1 and M2 $\bigoplus$ k = C2
If two keys are similar it'll reveal information on messages.
As C1 $\bigoplus$ C2 = (m1 $\bigoplus$ k) $\bigoplus$ (m2 $\bigoplus$ k)
= m1 $\bigoplus$ m2
Cipher text difference will give you message difference.
2. len (k) >= len (P)
As each bit of k has to be xored with each bit of P the length of k should be greater or equal to P. Say suppose we've P of 32 bits and k of 16 the remaining bits are xored with zero which gives away the part of message therefore perfect secrecy can't be achieved in this case.
l is the length of the key k.
P = P1 P2 P3 P4....Pl....Pn
k = k1 k2 k3 k4....klki..kt

C = (P1 $\bigoplus$ k1)(P2 $\bigoplus$ k2).....(Pl $\bigoplus$ kl)(Pl+1 $\bigoplus$ ki)..(Pn $\bigoplus$ kt)
Since these conditions are not always feasible to follow in real life OTP's implementation becomes difficult so it's not used in real life.

Data Encryption Standard (DES):

This is a block cipher which was designed by IBM.
1. Here, block size = 64 bit
2. Number of rounds = 16
3. Security key size = 64 bit with 8 parity check bits.
4. It is based on feistel network.
Encryption:
Here we take 64 bit input for plain text use 64 bit key and DES algorithm to produce 64 bit cipher text.
Decryption:
For decryption we take 64 bit cipher text as input use 64 bit key and DES^{-1} algorithm to produce 64 bit original text.
The secret key contains 8 parity check bits.
01101010 1101000 ... 01
After discarding 8 bits we've remaining of 56 bit secret key. DES should provide 56 bit security.
In DES we have 16 rounds keys k1,k2,k3...k16.
These round keys are generated by key scheduling algorithm.
Key scheduling algorithm will take secret key as a input.
G(k) = k1,k2,k3..k16 here G is the key scheduling algorithm.
len(ki) = 48 bits


input
$m_1 m_2 \cdots$
m_{64}
64
IP
initial permutation
64
L_0
R_0
48
K_1
32
32
32
f
L_1
R_1
K_2
f
L_{15}
R_{15}
K_{16}
f
irregular swap
R_{16}
L_{16}
64
IP^{-1}
inverse permutation
64
output
$c_1 c_2 \cdots$
c_{64}

Initial Permutation:
IP: $\{0,1\}^{64} \rightarrow \{0,1\}^{64}$
It's a mapping of 64 bits to 64 bits. This mapping can be done as shown in the following table.

IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Here,
IP(m1m2m3...m58) = m58m50m42m34, this is how initial permutation of a specific message is calculated the position of the bits is mapped as mentioned in the table above.
For IP^{-1} the table is as mentioned below.

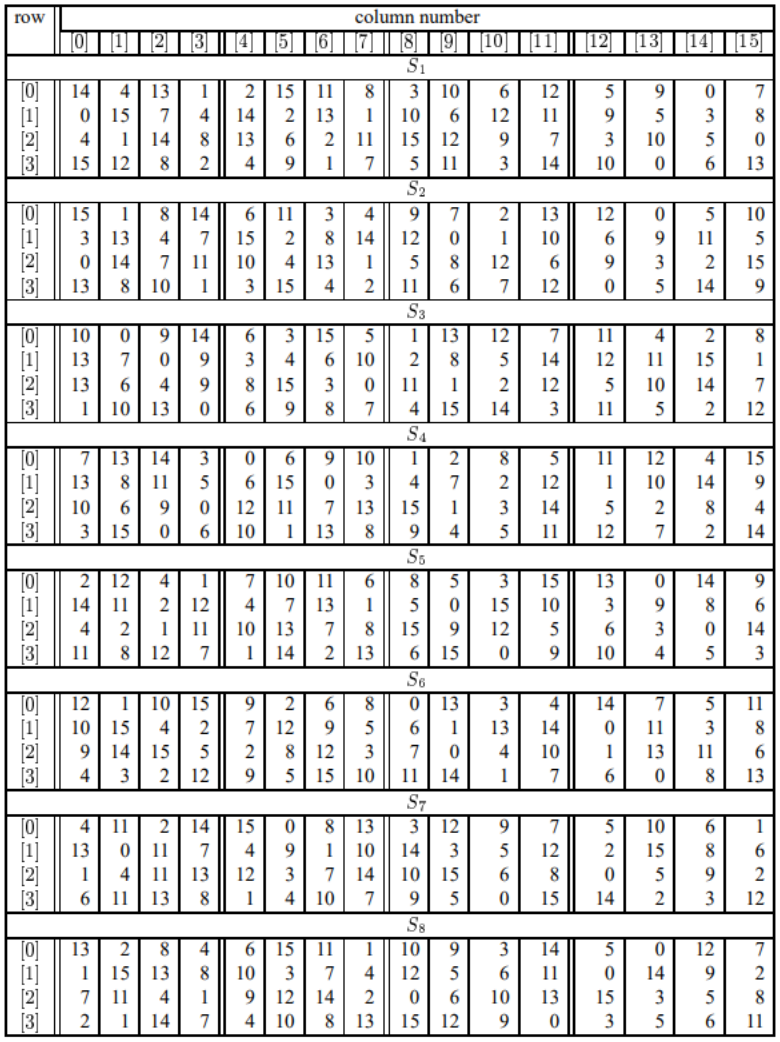
IP^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Round function (f) :

The round function is given by,
f(Ri, Ki) = Xi
here Ri is 32 bit,
Ki is 48 bit,
Xi is 32 bit.
$f(Ri,Ki) = P(S(E(Ri)\bigoplus Ki))$
Here P is the permutation box which maps 32 bits to 32 bits.
P: $\{0,1\}^{32} \rightarrow \{0,1\}^{32}$
S is the substitution box which maps 48 bits to 32 bits.
S: $\{0,1\}^{48} \rightarrow \{0,1\}^{32}$
E is the expansion function which maps 32 bits to 48 bits. E: $\{0,1\}^{32} \rightarrow \{0,1\}^{48}$
Just like IP we have a table for expansion table too, the mapping here is done based on following table:

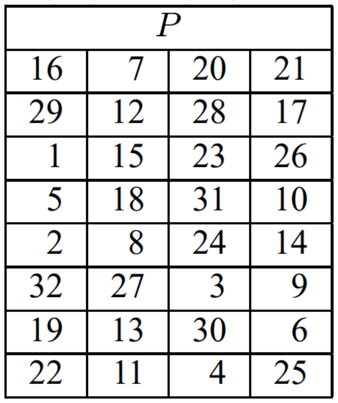
E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

We know we use substitution box for S: $\{0,1\}^{48} \rightarrow \{0,1\}^{32}$ mapping.
S(X) = Y, where X is the 48 bit input and Y is the 32 bit output.
X = B1B2B3B4B5B6B7B8, where len(Bi) = 6 bit
We've 8 S boxes as S1.....S8
Si: $\{0,1\}^6 \rightarrow \{0,1\}^4$ $\forall$ i =1,2,...,8
Si(Bi) = Ci
S(X) = S1(B1), S2(B2),... S8(B8)
S(X) = 32 bit
Bi = b1b2b3b4b6 bi $\in$ {0,1}
r = (2*b1 + b6) 0 <= r <= 3
r is the integer representation of (b1b6)
c is the integer representation of (b2b3b4b5)
0 <= c <= 15
0 <= aij <= 15
Si(Bi) = arc = 4 bits The mapping is done based on table mentioned below:



row	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
								S_1								
[0]	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
[1]	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
[2]	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
[3]	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
								S_2								
[0]	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
[1]	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
[2]	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
[3]	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
								S_3								
[0]	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
[1]	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
[2]	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
[3]	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
								S_4								
[0]	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
[1]	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
[2]	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
[3]	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
								S_5								
[0]	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
[1]	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
[2]	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
[3]	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
								S_6								
[0]	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
[1]	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
[2]	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
[3]	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
								S_7								
[0]	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
[1]	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
[2]	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
[3]	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
								S_8								
[0]	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
[1]	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
[2]	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
[3]	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

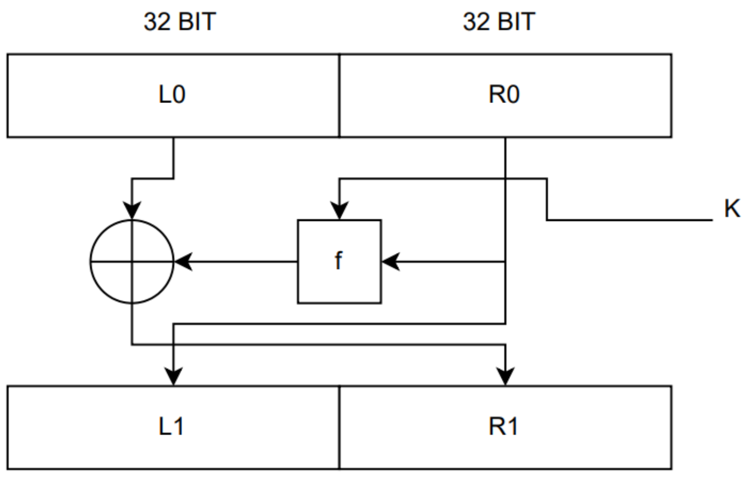
For the permutation function too there is a table which can be used for mapping. Here P is defined as
P: $\{0,1\}^{32} \rightarrow \{0,1\}^{32}$



P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Key Scheduling Algorithm:

Input: 64 bit key K
Output: 16 round key Ki, 1<= i <= 16 where length of ki is 48 bit.
1. First we have to define 16 constants Vi, 1<= i <= 16 where Vi = 1 if i $\in$ {1,2,9,16} else Vi =2.
2. Discard 8 parity check bit from K. The 56 key bit is k‘.
3. T = PC1(k‘); PC1: $\{0,1\}^{56} \rightarrow \{0,1\}^{56}$
4. (C0, D0) = T where C0 is of 28 bit and D0 is of 28 bit. 5. for i = 1 to 16
Ci = $C_{i-1} \hookleftarrow$ Vi
Di = $D_{i-1} \hookleftarrow$ Vi
Ki = PC2(Ci,Di)
PC2: $\{0,1\}^{56} \rightarrow \{0,1\}^{48}$



k = 48 bit
f = P(S(E(R0)$\bigoplus$k))
M = L0 || R0
C1 = L1 || R1
Given:
FN(M,K) = C1
FN($\overline{M}$, $\overline{K}$) = C2
We now have to find the relation between C1 and C2
L1 = R0
R1 = L0 $\bigoplus$ f(R0,k)
$\overline{K} \bigoplus$ E ($\overline{R0}$) = $\overline{E(R0)} \bigoplus \overline{K}$ =E(R0)$\bigoplus$ K
P(S($\overline{K}$ $\bigoplus$ E($\overline{R0}$) = P(S(E(R0) $\bigoplus$ K))
First,
M, K
L0 || R0 = M
L1 = R0
R1 = L0 $\bigoplus$ f(R0,k) now we consider,
M,K
$\overline{L1}||\overline{R0}$ = $\overline{M}$
L’ = $\overline{R0}$ = $\overline{L1}$

R' = $\overline{L0} \bigoplus F(\overline{R0}, \overline{k}) = \overline{R1}$
Therefore,
L1' || R1' = $\overline{L1}||\overline{R1}$
C2 = $\overline{C1}$

[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2022-2023
Scribed by: Bhargavi Kamble (202051048) — Lecture (Week 4)

Attack Models:

1. Cipher text only attack:
Here attacker only knows the cipher text from the encryption model. We don't know the secret key or the plain text. The main goal is to get the plain text corresponding to the cipher text or recover the secret key. If we are able to find the secret key we can break the algorithm.
2. Known plain text attack model:
Here the attacker knows some bits from of the plain text of the corresponding cipher text and the cipher text. Here the main goal is to generate new plain text, cipher text pair or recover the secret key.From this information attacker tries to find the weakness in the model.
3. Chosen Plain text attack model:
Here attacker can choose his plain text and then he's/she's provided the corresponding cipher texts. Here the main goal is to generate new plain text, cipher text pair or recover the secret key.
4. Chosen Cipher text attack model:
Attacker can choose a different cipher text and we can get the corresponding plain text. It is must stronger attack model than the previous ones. Here the goal is to generate a new plain text, cipher text pair or recover the secret key.

Brute force on DES:

We know that,
DES(M, K) = C
DES($\overline{M}, \overline{K}$) = $\overline{C}$
In DES we have a key of 64 bits but out of that 8 bits are parity check bits therefore it leaves the total to 56 bits. If we have to try an exhaustive search approach to find the key the total number of permutations are 2^{56}. One key will give a meaningful message that will be our original key.
Now we can prove that we can retrieve the search for key in 2^{55} searches using the above property.
Attacker choses bet the two plain texts M and $\overline{M}$ the challenge here is to find the key K.
C1 = DES(M,K) C2 = DES($\overline{M}, \overline{K}$). Here both C1 and C2 are encrypted using k the main goal is to compute the K here. Attacker knows that if the message $\overline{M iscomputedusing K}$ the resultant will give $\overline{C}$. Attacker is getting C1 and C2
DES($\overline{\overline{M}}, \overline{K}$) = $\overline{C2}$
which gives us DES($M, \overline{K}$) = $\overline{C2}$
We have the set of keys as K = {k1, k2, ..., k2^{56}}
Attacker selects K1 ∈ K.
He knows that $\overline{K1}$ ∈ K.
Attacker performs DES(M, $\overline{K1}$) = C'if C' is not equal to C or C' is not equal to $\overline{C2}$ then we can discard key K1 and its compliment $\overline{K1}$.
If C' ≠ C1 that means K1 ≠ K.
If C' ≠ C2 that means K1 ≠ K and $\overline{K1}$ ≠ K.
In every search attacker the eliminating two choices therefore the total number of exhaustive search reduces down to 2^{56} searches.

Double Encryption:

DES is not secure for multiple reasons. One of the solution for increasing security was to Double encrypt the plain text but is really secure? The process is discussed as following.
So we have a key K which is a concatenation of two keys K1 and K2.
len(k1) = 56 bits
len(k2) = 56 bits
Therefore the length of K is equal to 112 bits.

Encryption: P $\rightarrow$ K1 + $\boxed{Enc_{DES}}$ $\rightarrow$ K2 + $\boxed{Enc_{DES}}$ $\rightarrow$ C

Decryption: C $\rightarrow$ K2 + $\boxed{Dec_{DES}}$ $\rightarrow$ K1 + $\boxed{Dec_{DES}}$ $\rightarrow$ C

This can also be done as

Encryption: P $\rightarrow$ K1 + $\boxed{Enc_{DES}}$ $\rightarrow$ K2 + $\boxed{Dec_{DES}}$ $\rightarrow$ C

Decryption: C $\rightarrow$ K2 + $\boxed{Enc_{DES}}$ $\rightarrow$ K1 + $\boxed{Dec_{DES}}$ $\rightarrow$ C

We can have EE, ED, DE and DD with these compositions we can have different setups.
Double encryption will not provide extra security.
Attacker knows plain text M and the corresponsing C.
C = Enc(Enc(M,K1),K2)
Keys = {SK1,SK2,....,SK2^{56}}
Enc = (M,Ski) = xi
Dec = (C,Skj) = Yj
If xi = Yj for some i,j then the key Ski || Skj.

M $\rightarrow$ Ski + $\boxed{Enc}$ $\rightarrow$ Skj + $\boxed{Enc}$ $\rightarrow$ C

The number of searches required for single encryption and double encryption are approximately equal.

Triple Encryption:

Since double encryption doesn't provide extra security triple encryption is an alternate way to achieve extra security. It provides 2-n bit security. Here too we obtain key K by concatenating two keys k1 and k2. The encryption can be done as following:

P $\rightarrow$ K1 + $\boxed{Enc_{DES}}$ $\rightarrow$ K2 + $\boxed{Enc_{DES}}$ $\rightarrow$ K1 + $\boxed{Enc_{DES}}$ $\rightarrow$ C

EEE, EDE, DED....etc are different setups we can use.
If we use DES in similar way then its called Triple DES .

Advanced Encryption Standard (AES):

We have to understand certain mathematical results.

Binary Operation:

A binary operation * on a set S is a mapping from S X S to S.
This * sign is a rule which assigns to each ordered pair of elements from S to an element of S.
* : S X S $\rightarrow$ S

Group:

A group (G, *) consists of a set G with a binary operation * on G. Satisfying the following axioms:
1. * is assosiative on G. a * (b * c) = (a * b) * c $\forall$ a,b,c $\in$ G.
2. There is an element e $\in$ G called the identity element such that, a * e = a = e * a $\forall$ a $\in$ G.
3. For each a $\in$ G there exists an element a^{-1} $\in$ G callled the inverse of 'a' such that: a X a^{-1} = e = a^{-1} X a $\forall$ a $\in$ G.
A group G is called abelian or commutative if a * b = b * a $\forall$ a,b$\in$ G.

Example:

1. * : Matrix multiplication over square matrices on order n x n. M: set of n x n matrices over R. Is (M, *) a group?
No it is not. As inverse of every matrix is not possible as determinant of many matrices is zero.
If M = set of all invertible matrix then too (M, *) will not be a group as A * B $\neq$ B * A it is not commutative.
2. If we consider a set of all integers (Z, +) then?
Yes it is a group.
Since,
a + b + c = (a + b) + c
a + 0 = a = 0 + a $\forall$ a $\in$ Z.
$\forall$ a $\in$ Z there exists -a $\in$ Z such that a + (-a) = 0 = - a + a.
3. (Z, X) then?
It is not a group. As,
a x (b x c) = (a x b) x c
a x 1 = a = 1 x a
But a $\in$ Z there doesn't exist a^{-1} $\in$ Z.
4. Q: Set of all rational numbers.
(Q, X) $\rightarrow$ not a group
(Q - 0, X) $\rightarrow$ is a group

If G is finite then (G, *) is a finite group. $|G|$: **cardinality of G**.
Ex: (Zn,+) is a group.
Ex: (Zn-0,*) is a group.
x *n y = (x*y) mod n
*n is the multiplication modulo n.
i) x *n (y *n z) = (x *n y) *n z
ii) x *n 1 = 1 *n x = x
iii) a *n b = 1 = b *n a $\forall$ a
(Zn*, *n) is a group.

[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy Winter 2022-2023
Scribed by: Bhargavi Kamble (202051048) Lecture (Week 5)

Group:

Group is a set with binary operation and this binary operation will have certain properties. It will be associative, there will be one identity element and there will be inverse of every element and the group will also be commutative. If we take an element and call it alpha. Let $\alpha 0$ be the identity element and $\alpha 0 \alpha 1 \alpha 2 \alpha 3 \in$ G. For any b $\in$ G if $\exists$ i >= 0 such that b = α^i then α is called the generator of (G,*). If we operate identity with α we will get α. We donate it with $\alpha 1$ it is similar to multiplication. Now if we operate $\alpha 1$ with α we will get $\alpha 2$ and similarly we can generate all α^n elements.
Similarly, if the operation was (G, +) then the generation of α would have been in $1\alpha 2\alpha$... so on. As $\alpha + \alpha = 2\alpha$.
(G, *) = $< \alpha >$
G $\subseteq$ $< \alpha >$

If G is a finite group in which, for each n ¿ 0, G contains at most n elements of order dividing n, then G must be cyclic.
(G, *)
$|G|$: finite
O (a) = m
a $\in$ G
a^m = e
e = $a^0, a^1, a^2, ...a^{m-1}$ $\in$ G
H = {e=a^0, a^1, a^2, a^3,...., a^{m-1}
1) h $\subseteq$ G (cyclic group)
2) H is group under *
Lagrange's Theorem
If G is a finite group and H is a subgroup of G then $|H|$ divides $|G|$
G is a finite group
a $\in$ G
H is a subgroup of G.
From Lagrange's theorem,
$|H| \mid |G|$ = O(a) $\mid G$
If the order of a $\in$ G is t then O(a^k) = t / gcd(t,k)
If gcd(t,k) = 1 then O(a^k) = t = O(a)
$| < a^k > |$ = $| < a > |$
x $\in$ $| < a^k > |$
x = $(a^k)^i$ = a^{kj} $\in$ $| < a > |$
Since, $| < a^k > |$ = $| < a > |$
a^k is also an generator of $| < a > |$.
Z^*_{19} = { x | gcd(x,19) = 1, 1 =< x =< 18 }
*19 = multiplication modulo of 19

x *19 y = x * Y modulo 19
Lets find the generators of (Z*19, *19)
$< 2 >$ = {1,2,4,8,16,13,7,14,9,18,17,15,11,3,6,12,5,10} = $Z*_{19}$
Here we started with 2 and it generated all the numbers.

Ring:

A ring (R, +R, XR) consists of one set R with two binary operations arbitrarily denoted by +R (addition) and XR (multiplication) on R satisfying the following properties.
1. (R, +R) is a abelian group with the identity element OR.
2. The operation Xr is associative i.e, a XR (b XR C) = (a XR b) XR C $\forall$ a,b,c $\in$ R.
3. There is a multiplicative identity denoted by 1R with 1R $\neq$ OR: Such that, 1R XR a = a XR 1R = a $\forall$ a $\in$ R. 4. The operation XR is distributive over +R. i.e: (b + RC) XR a = (bXR a) +R (C XR a) a XR (b +R C) = (a XR b) +R (a XR C)
EX1: (Z, +, $\cdot$) we have to check if this is a ring or not.
Z is a set of all integers.
It is a ring.
EX2: (R, +R, XR) is a ring
$\cdot$ a XR b = b XR a $\forall$ a,b $\in$ R then (R, +R, XR) is a commutative ring.
EX3: (Z, +, $\cdot$) is a ring.
a $\cdot$ b = b $\cdot$ a $\forall$ a,b $\in$ Z is Commutative ring.
An element 'a' of a ring R is called unit or an invertible element if there is an element b $\in$ R such that a XR b = 1R.
Now we have to find is (Z, +n, *n) is a ring or not.
X +n y = X + y mod n
x *n y = x * y mod n
(X +n y) *n Z = X Kn Z +n y Kn Z.
It is group as well as commutative group. Under the second operation we have the identity element too. It is associative and distributive over addition too. Therefore, it is a ring.
The set of units in a Ring R forms a group under multiplication operation. This is known as group f units of R.
Field:
A field is a non empty set F together with two binary operation + and * for which the following properties are satisfied.
1. (F, +) is an abelian group.
2. If OF denotes the additive identity element of (F, +) then (F {OF}, *) is a commutative/ abelian group.
3. $\forall$ a,b,c $\in$ F we have a * (b + C) = (a * b) + (a * C).
We have to find if (Zp, +p, *P) is a field or not. Zp = {0,....,p-1}. P is a prime number here. It is a commutative group as 0 is the identity here. It is associative as well as commutative therefore under first operation its a group. Similarly it is a commutative group under the second operation too.
We are considering prime numbers only therefore we will have a co-prime number to it. Therefore above expression is a field.
EX: Is (Z, +, $\cdot$) a field?
Ans: No
EX: Is (Q, + , $\cdot$) a field?

Ans: (Q, +) = abelian group.
0 is the additive identity
1 is the multiplicative identity
(Q {0}, ·) is abelian group.
Therefore it is a field.
EX: P = prime number.
Fp = {0,1,2,....,p-1}
Is (Fp, +p, *p) a field or not?
Ans: x +p y = (x +y) mod p
x *p y = (x*y) mod p
It is a field.
(Fp {0} ∃ b ∈ Fp {0} such that
a *p b =1
=(a * b) mod p =1
(ab-1) = t *p
1 = a * b + t1p
gcd(a,p) = a.b + t1p

Field Extension:

Suppose k2 is field with addition (+) and multiplication (*). Suppose k1 ⊆ k2 is closed under both these operations such that k1 itself is field with the rstriction of + and * then k1 is called a subfield of k2 and k2 is called a field extension of k1.
F = field (F, +, *)
F[x] = {a0, a1x + a2x+... | ai ∈ F }
F[x] is the polynomial ring.
+: polynomial addition
*: polynomial multiplication
P1(x) ∈ F[x]
P2(x) ∈ F[x]
p1(x) = a0 + a1x + a2x^2
p2(x) = b0 + b1x + b2x^2
p1(x) + p2(x) = (a0 + a1x + a2x^2) + (b0 + b1x + b2x^2)
= (a0 + b0) + (a1 + b1)x + (a2 + b2)x^2
(a1 + b1) = Field addition.
a0 + a1x + a2x^2 + ... + an-1x^{n-1}
+ b0 + b1x + b2x^2+ ..... + bn-1x^{n-1}
= (a0 + b0) + (a1 + b1) x + (a2 + b2) x^2 + ... + (an-1 + bn-1)x^{2n-2}
(ai + bi): additive operation on F as ai, bi ∈ F.
Similarly for multiplication too,
a0 + a1x + a2x^2 + ... + an-1x^{n-1}
* b0 + b1x + b2x^2+ ..... + bn-1x^{n-1}
= (a0 * b0) + (a1 * b1) x + (a2 * b2) x^2 + ... + (an-1 * bn-1)x^{2n-2}
ai * bi = field multiplication in F as ai, bi ∈ F. Addition between the elements has to be done in the field.
(F[x], +, *) is a polynomial ring.
1) (F[x], +) must be an abelian group.

2) * is associative.
3) 1 is the multiplicative identity.
4) * is distributive over +.
F2 = {0,1} (F2, +2, *2) = field
F2[x] = a0, a1x, a2x^2+..... | ai $\in$ F2
p(x) = x +1 $\in$ F2[x]
q(x) = x^2 + x + 1 $\in$ F2[x]
p(x) + q(x) = (x+1) + (x^2 + x + 1)
= x^2 + (1 + 1)x + (1+1)
= x^2
p(x) + q(x) = (x+1) * (x^2 + x +1) = x^3 + 1
A polynomial p(x) $\in$ F[x] of degree n (n ¿= 1) is called irreducible if it cannot be written in the form of p1(x) * p2(x) with p1(x), p2(x) $\in$ F[x] and degree of p1(x), p2(x) must be ¿=1. P(x) $\neq$ p1(x) * p2(x)
x^2 + 1 $\in$ F2[x]
(x+1) * (x+1) = x^2 + 1
Therefore, x^2 + 1 is reducible in F2[x]
I = $< p(x) >$ = {q(x).p(x) | q(x) $\in$ F[x]}
I = ideal generator by p(x)
F[x] / $< P(x) >$
q(x) $\in$ F[x]
r(n) $\in$ F[x] / $< p(x) >$
If p(x) is irreducible polynomial then (F[x] / $< P(x) >$, +, *) becomes a field.

x^2 + x + 1 $\in$ F2[x]. f2={0,1}
p(x) = x^2 + x + 1 is irreducible.
F2[x] / $< x^2 + x + 1 >$
q(x) = d(x).p(x) + r(x)
deg(r(x)) ¡ 2
r(x) $\in$ {0,1,x,x+1}

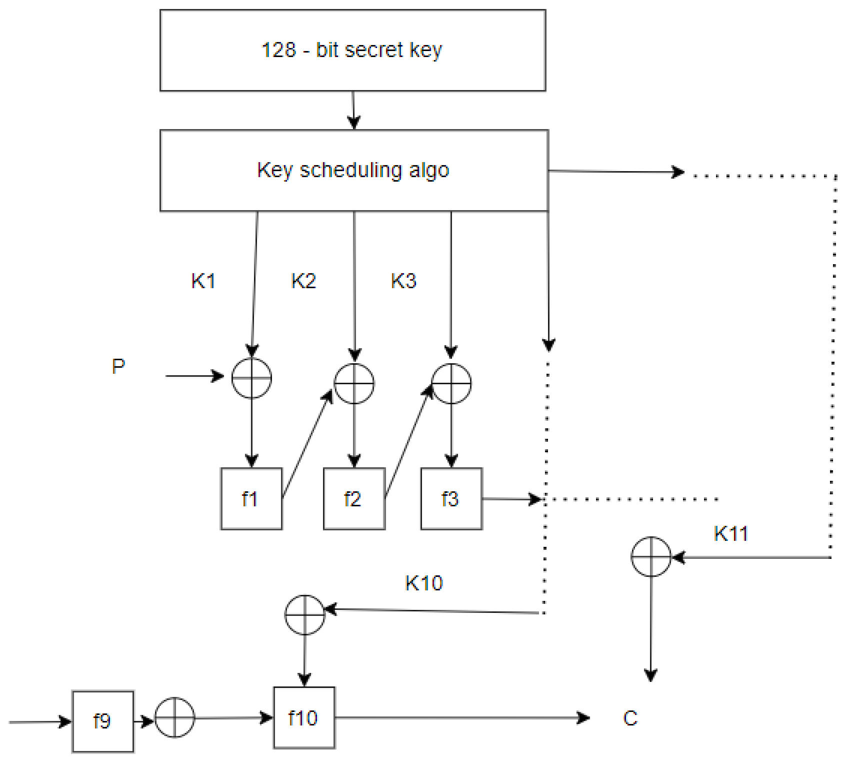
F2[x]/$< x^2 + x + 1 >$
x^2 + x + 1 =0
α is the root of x^2 + x + 1 =0
α^2 + α + 1 = 0
α^2 = α + 1
{0,1=α^0, α^1, α^2 = α + 1}
= {0,1,α, $\alpha + 1$}

Advanced Encryption Standard (AES)

It is standardized by NIST
- Rijndael
Winner of Advanced Encryption Standard Competition.
- Winner of competition was named as AES.
AES:
i. Iterated block cipher

ii. It is based on SPN
AES-128
i. Block size = 128 bit
ii. Number of rounds = 10
iii. Secret key size = 128 bit
AES-192
i. Block size = 128 bit
ii. Number of rounds = 12
iii. Secret key size = 192 bit
AES -256
i. Block size = 128 bit
ii. Number of rounds = 14
iii. Secret key size = 256 bit

AES-128 Design:



[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2022-2023
Scribed by: Bhargavi Kamble (202051048) — Lecture (Week 6)

AES:

We have to understand two things:
1. Round function
2. Key scheduling algo.

Round function of AES-128:

We generate 10 round keys f1,f2,...,f10.
1. f1=f2=....=f9
2. f10 is different from fi, i =0 to 9. First 9 round function are exactly same, 10th round function is different from other 9 round functions.
The first 9 round function i.e f1...f9 are based on the following function:
i. Sub-bytes.
ii. Shift rows.
iii. Mix Column.
fi: $\{0,1\}^{128}$ = $\{0,1\}^{128}$
The 10th r.f (i.e: f10) is based on:
i. Sub-bytes.
ii. Shift rows.
fi: $\{0,1\}^{128}$ = $\{0,1\}^{128}$
The function for first 9 rf can be written as:
MaxCol(ShiftRows(SubBytes(x))) = fi(x)
It is a bi-jection therefore, we can have inverse at each step.

$128 \rightarrow \boxed{fi} \rightarrow$ 128 bit

$128 \rightarrow \boxed{Sub-Bytes}$ $128 \rightarrow \boxed{Shiftrows}$ $128 \rightarrow \boxed{MixColumn}$ 128

Sub-bytes: $\{0,1\}^{128}$ = $\{0,1\}^{128}$

S = input

s = $\begin{bmatrix} s00 & s01 & s02 & s03 \\ s10 & s11 & s12 & s13 \\ s20 & s21 & s22 & s23 \\ s30 & s31 & s32 & s33 \end{bmatrix}$
si,j = 8 bit
ii. Plaintext P = 128 bit
P = P0P1....P15
len(Ri) = 8

$$P = \begin{bmatrix} P0 & P4 & P8 & P12 \\ P1 & P5 & P9 & P13 \\ P2 & P6 & P10 & P14 \\ P3 & P7 & P11 & P15 \end{bmatrix}$$

P $\bigoplus$ K1 = S where,

S = s = $\begin{bmatrix} s00 & s01 & s02 & s03 \\ s10 & s11 & s12 & s13 \\ s20 & s21 & s22 & s23 \\ s30 & s31 & s32 & s33 \end{bmatrix}$

Si will be of 4 * 4 matrix.
K1 = 128 bit

K0....k15 the corresponding matrix can be given by, K = $\begin{bmatrix} K0 & K4 & K8 & P12 \\ K1 & K5 & K9 & P13 \\ K2 & K6 & K10 & K14 \\ K3 & K7 & K11 & K15 \end{bmatrix}$

K1 = 128 bit

0.1 Sub-Bytes:

S = (Sij)$_{4*4}$
S: {0,1}^{8} = {0,1}^{8}
S(0) = 0
1. C7C6C5C4C3C2C1C0 = (01100011) = 63
2. S(Sij) = (a7a6a5a4a3a2a1a0)
3. For i = 0 to 7
bi = (ai + a$_{(i+4)mod8}$ + a$_{(i+5)mod8}$ + a$_{(i+6)mod8}$ + a$_{(i+7)mod8}$ + a) mod 2
4. We will get b7b6.....b0
5. Sij = (b7b6....b0)

S: {0,1}^{8} = {0,1}^{8}
S(0) = 0
x $\neq$ 0 $\in$ {0,1}^{8}
S(x) = Y $\in$ {0,1}^{8}
x = (a7a6a5a4a3a2a1a0), ai $\in$ {0,1}
P(X) = a0 + a1x + a2x^2 + ... + a7x^7 $\in$ F2[X]
deg(P(x)) <= 7
ai $\in$ F2
P(x) $\in$ F2[x]
(F2[x], +, *)
g(x) = x^8 + x^4 + x^3 + x + 1
g(x) is a primitive polynomial.
(F2[x] / $< g(x) >$, +, *) = field
Find the multiplication inverse of P(x) under modulo x^8 + x^4 + x^3 + x + 1
p(x).q(x) = 1 mod (x^8 + x^4 + x^3 + x + 1)
p(x).q(x) - 1 = h(x) (x^8 + x^4 + x^3 + x + 1)

1 = p(x). q(x) + h1(x) * (x^8 + x^4 + x^3 + x + 1)
gcd(a,b) = as + bt
gcd(p(x), (x^8 + x^4 + x^3 + x + 1)) = 1
How can we find q(x)?
We can do that by using extended euclidean algorithm.
q(x) : polynomial of degree at the most 7
q(x) = r0 + r1x + r2x^2+ ..... + r7r^7
q(x) = (r7r6r5...r0) ∈ $\{1,0\}^8$
S(x) = Y = (r7r6...r0)

Example:
S(01010011) = ?
p(x) = x^6 + x^4 + x + 1 (deg = 6)
g(x) = x^8 + x^6 + x^3 + x + 1
x = 01010011
Now we have to find the multiplication inverse:

$$\begin{array}{l}
x^6+x^4+x+1)\overline{x^8+x^4+x^3+x+1}(x^2+1 \\
\qquad\qquad\qquad x^8+x^6+x^3+x^2 \\
\qquad\qquad\qquad\quad \overline{x^6+x^4+x^2+x+1} \\
\qquad\qquad\qquad\quad x^6+x^4+x+1 \\
\qquad\qquad\qquad\qquad\quad \overline{x^2)x^6+x^4+x+1}(x^4+x^2 \\
\qquad\qquad\qquad\qquad\qquad x^6 \\
\qquad\qquad\qquad\qquad\qquad\quad \overline{x^4+x+1} \\
\qquad\qquad\qquad\qquad\qquad\quad x^4 \\
\qquad\qquad\qquad\qquad\qquad\quad \overline{x+1)}\overline{x^2}\quad (x \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad x^2+1 \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \overline{1}
\end{array}$$

1 = q(x).p(x) + h(x).g(x)
1 = x^2 + ((x+1)*(x+1))
= x^2 + (x+1) [$(x^6+x^4+x^2+x+1)$ + x^2 + (x^4+x^2)]
= (x+1)$(x^6+x^4+x^2+x+1)$ + [1 + (x +1) (x^4+x^2)]x^2
= (x+1)$(x^6+x^4+x^2+x+1)$ + (1 + $x^5+x^4+x^3+x^2+1)x^2$
= (x+1)$(x^6+x^4+x^2+x+1)$ + (1 + $x^5+x^4+x^3+x^2+1$) + (x^2+1) (x^6 + x^4 + x + 1)]
= h1(x).g(x) + (x + 1 + x^2 + x^7 + x^6 + 1 + $x^5+x^4+x^3+x^2$) (x^6 + x^4 + x + 1)
= h1(x).g(x) + + (x^7 + x^6 + x^3 + x)x^6 + x^4 + x + 1)
q(x) = x^7 + x^6 + x^3 + x)
which is the multiplication inverse of x^6 + x^4 + x + 1
S(01010011) = (1001010) = (a7....a0)
c = (0110011)
bi = (ai + $a_{(i+4) mod 8}$ + $a_{(i+5) mod 8}$ + $a_{(i+6) mod 8}$ + $a_{(i+7) mod 8}$ + a) mod 2
b0 =1, b1 =0, b2 =1, b3 =1, b4=0, b5=1, b6=1, b7=1
above value are calculated using bi formula.
Sub-bytes (0101 0011) = (1110 1101)
Subbytes (53) = ED
S: $\{0,1\}^8$ = $\{0,1\}^8$
Input = (XY) Subbyte (Input) = element present in the row numer and column number. These

values can be mapped by the table mentioned in Stinson's book page:112.

[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2022-2023

Scribed by: Bhargavi Kamble (202051048) — Lecture (Week 7)

Shiftrows:

Shiftrows: $\{0,1\}^{128} = \{0,1\}^{128}$

Here we have a input of 128 bits and the output is of 128 bits too.
The position of elements are mapped to the positions mentioned in the second matrix.

$$\begin{bmatrix} s00 & s01 & s02 & s03 \\ s10 & s11 & s12 & s13 \\ s20 & s21 & s22 & s23 \\ s30 & s31 & s32 & s33 \end{bmatrix} \rightarrow \begin{bmatrix} s00 & s01 & s02 & s03 \\ s11 & s12 & s13 & s10 \\ s22 & s23 & s20 & s21 \\ s33 & s30 & s31 & s32 \end{bmatrix}$$

Mix column:

Mix Column: $\{0,1\}^{128} = \{0,1\}^{128}$

Here we have a input of 128 bits and the output is of 128 bits too.
$(\text{Sij})_{4*4} \rightarrow (\text{S‘ij})_{4*4}$
Consider the column C $\in$ {0,1,2,3}

$$\text{Column} = \begin{bmatrix} S0c \\ S1c \\ S2c \\ S3c \end{bmatrix}$$

Sic = (a7a6....a0)
Polynomial = a0 + a1x + ... + a7x^7
For i =0 to 3
ti = Binary to poly (Sic)
u0 = [(x * t0) + (x + 1) * t1 + t2 + t3] mod (x^8 + x^4 + x^3 + x + 1)
u1 = [(x * t1) + (x + 1) * t2 + t3 + t0] mod (x^8 + x^4 + x^3 + x + 1)
u2 = [(x * t2) + (x + 1) * t3 + t0 + t1] mod (x^8 + x^4 + x^3 + x + 1)
u3 = [(x * t3) + (x + 1) * t0 + t1 + t2] mod (x^5 + x^4 + x^3 + x + 1)
S‘ic =polynomial to binary (ui)

$$\begin{bmatrix} s0c \\ s1c \\ s2c \\ s3c \end{bmatrix} \xrightarrow{mixcolumn} \begin{bmatrix} s‘0c \\ s‘1c \\ s‘2c \\ s‘3c \end{bmatrix}$$

Mix Column:

$$\begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix} * \begin{bmatrix} s00 & s01 & s02 & s03 \\ s10 & s11 & s12 & s13 \\ s20 & s21 & s22 & s23 \\ s30 & s31 & s32 & s33 \end{bmatrix}$$

mod (x^8 + x^4 + x^3 + x + 1) = (S'ij)$_{4*4}$

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} * (Sij)mod(.) = (S'ij)_{4*4}$$

Find S' sfter joing mix column operation on s:

$$where s' = \begin{bmatrix} s'00 \\ s'10 \\ s'20 \\ s'30 \end{bmatrix} and s = \begin{bmatrix} s00 \\ s10 \\ s20 \\ s30 \end{bmatrix}$$

where s00 = 95, s10 = 65, s20 = FD and s30 = F3
Solution:
s00 = 95 = 10010101 = $x^7 + x^4 + x^2 + 1$
s10 = 65 = 01100101 = $x^6 + x^5 + x^2 + 1$
s20 = FD = 11111101 = $x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + 1$
s30 = F3 = 11110011 = $x^7 + x^6 + x^5 + x^4 + x + 1$
s'00 = (x * s00) + ((x+1) * s10) + s20 + s30
x * s00 = x * ($x^7 + x^4 + x^2 + 1$)
= $x^8 + x^5 + x^3 + x$
(x * s00) mod ($x^8 + x^5 + x^3 + x$ + 1) = ($x^8 + x^5 + x^3 + x$) mod ($x^8 + x^5 + x^3 + x$ + 1)

$$\begin{array}{r} x^8+x^4+x^3+x+1 \overline{)x^8+x^5+x^3+x}(1 \\ x^8+\underline{x^4+x^3+x+1} \\ x^5+x^4+1 \end{array}$$

$x^8 + x^4 + x^3 + x \equiv$ ($x^4 + x^3 + x + 1$) + $x^5 + x^3 + x$
$\equiv x^5 + x^4 + 1$ mod ($x^8 + x^4 + x^3 + x + 1$)
x * s00 = $x^5 + x^4 + 1$
(x + 1) * s10 = (x + 1) * ($x^6 + x^5 + x^2 + 1$)
= $x^7 + x^6 + x^3 + x + x^6 + x^5 + x^2 + 1$
= $x^7 + x^5 + x^3 + x^2 + x + 1$
s'00 = (x * s00) + (x +1) * s10 + s20 + s30
= (x^5+x^4+1) + (x^7+x^5+x^3+x^2+x+1) + (x^7+x^6+x^5+x^4+x^3+x^2+1) + (x^7+x^6+x^5+x^4+x+1)
= $x^7 + x^4$
s'00 = $x^7 + x^4$ = 10010000 = 90

Key Scheduling Algorithm of AES-128:

Input: 128 bit key
Output: 11 round keys, length of each round key is 128 bit.
Key = (Key[15], ..., Key[0])
16 bytes
We will prepare 44 words which are denoted by w[0],......w[43]
ROTWORD (B0, B1, B3, B3) = (B1, B2, B3, B0)
SUBWORD (B0, B1, B2. B3) = (B0‘, B1‘, B2‘, B3‘)
where Bi‘ = SubBytes (Bi) ∀ i =0,..,3
10 round constants (word)
Rcon[1] = 01000000
Rcon[2] = 02000000
Rcon[3] = 04000000
Rcon[4] = 08000000
Rcon[5] = 10000000
Rcon[6] = 20000000
Rcon[7] = 40000000
Rcon[8] = 80000000
Rcon[9] = 1B000000
Rcon[10] = 36000000

```
for i =0 to 3
w[i] = 9 key [4i], key[4i + 1], key[4i +2], key[4i + 3])
for i = 4 to 43
temp = w[i -1]
if i ≡ 0 mod 4
temp = SUBWORD(ROTWORD(temp)) ⊕ Rcon[i/4]
w[i] = w[i -4] ⊕ temp
return (w[0], .....m  w[43])
```

Round keys k1, k2, k3,......, k11
k1 = w[0] || w[1] || w[2] || w[3]
k2 = w[4] || w[5] || w[6] || w[7]
....
k11 = w[40] || w[41] || w[42] || w[43]
We have subbyte (A) = B
A = X || Y
Inv SubByte (b) → find B in the table also find X, Y
X || Y

$$\begin{bmatrix} s00 & s01 & s02 & s03 \\ s10 & s11 & s12 & s13 \\ s20 & s21 & s22 & s23 \\ s30 & s31 & s32 & s33 \end{bmatrix} \rightarrow \begin{bmatrix} s00 & s10 & s20 & s30 \\ s11 & s21 & s31 & s01 \\ ... & ... & ... & ... \\ ... & ... & ... & ... \end{bmatrix}$$

Mix Column:

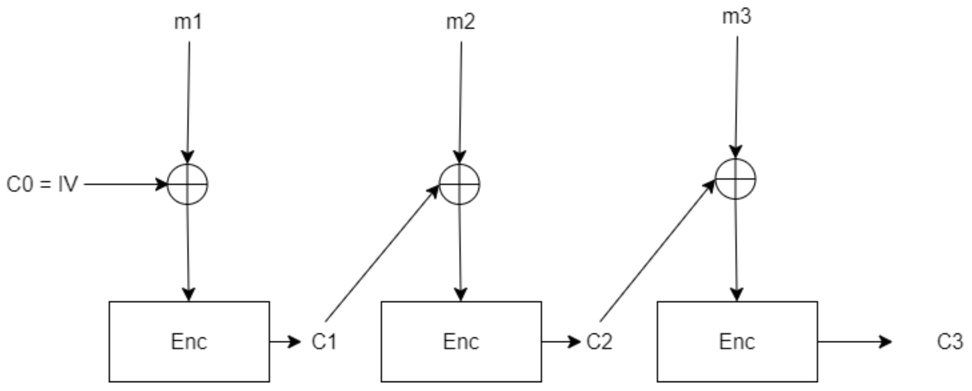
MixCol (S) = S‘
MixCol(MixCol(MixCol(MixCol(S))))

= M^4 * S = I * S
M^4 = I
MixCol(MixCol(MixCol(S‘)))) = S

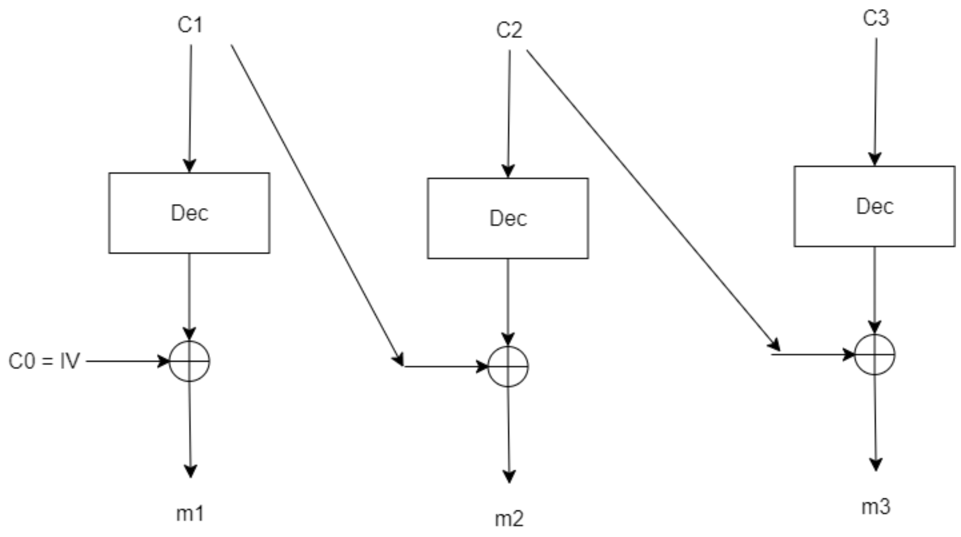
Modes Of Operations:
1. ECB (Electronic Codebook Mode)
2. CFB (Cipher Feedback Mode)
3. CBC (Cipher Block Chaining Mode)
4. OFB (Output Feedback Mode)
5. Counter mode
6. CCM (Counter with Cipher block Mode)
ECB:
M = m0 || m1 || .... || mt
Enc = It can encrypt l bit plaintext block
len(mi) = l bit
Encryption:
C = C0 || C1 || ..... || Ct
Ci = Enc(mi, K) ∀ i =0,....,t

CBC Mode:
M = m1 || m2 || ..... || mt
initialization vector = IV (public variable)
Enc = block size l
len(mi) = l
len(IV) = l
i) c0 = IV
ci = Enc(c_{i-1} $\bigoplus$ mi, k) ∀ i =1,....,t

Encryption:



Decryption:

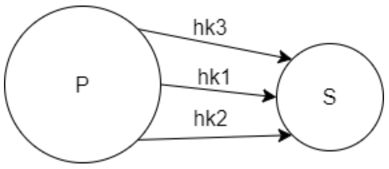


Hash Function:

h: A → B
h(x) = Y
1. If X is altered to X‘ then h(X‘) will be completely different from h(X)
2. Given Y it is practically infeasible to find X such that h(X) = Y
3. Given X and Y = h(X) it is practically infeasible to find X‘ such that h(X) = h(x‘)

Alice Bob
X = E(M, K) $\xrightarrow{X}$ X1 = Dec(X‘, K)
S1 = h(M,K) $\xrightarrow{S1}$ S2
If h(X1, K) = s2 then bob accept X1
Definition:
A hash family is a four tuple (P, S, K, H) where the following conditions are satisfied:
1. P is the set of all possible messages.
2. S is the set of all possible message digests or authentication tags.
3. K is the key space.
4. For each K1 ∈ K there is a hash function h_k1 ∈ H such that

h_k1 : P → S
here $|P| >= |S|$ more interestingly $|P| >= 2 * |2|$



H: set of all hash functions
h_k1: hash function

If key is involved in the computation of hashed value then that hash function is known as keyed hash function.
If the key is not required to compute the hashed value then that hash function is known as unkeyed hash fucntion.
Problem 1:
h: p $\rightarrow$ S
Given y $\in$ S find x $\in$ P such that h(x) = y
This problem is known as pre image finding problem.
For on hash function h if you cannot find preimage in a feasible time then h is known as preimage resistant hash function.
Finding preimage is computationally hard for preimage resistant hash function.
Problem 2:
h: P $\rightarrow$ S
Given x $\in$ P and h(x) find x' $\in$ P such that x'$\neq$ x and h(x') = h(x)
This prolblem is known as second pre image finding problem.
If finding second pre image is computationally hard for h then h is known as second pre image resistant hash function.
Problem 3:
h: P $\rightarrow$ S
Find x, x' $\in$ P such that x *neq* x' and h(x) and h(x')
This problem is known as collision finding problem.
For an hash function h if finding collision is computationally hard then h is known as collision resistant hash function.

Ideal Hash Function:

Let h: P $\rightarrow$ S be an hash function.
h will be called ideal hash function if given x $\in$ P to find h(x) either you have to apply h on x or you have to look into the table corresponding to h (hash table).

Pre Image Finding Algorithm:

Given y $\in$ Y find x $\in$ X such that h(x) = yx
h: X $\rightarrow$ Y
$|Y|$ = m
Y = { y1, y2,...., ym} x1 $\rightarrow$ y1, x2 $\rightarrow$ y2, ..., xm $\rightarrow$ ym
h: X $\rightarrow$ Y
choose any X0 $\subseteq$ X such that $|X0|$ = Q for each X $\in$ X0.
Compute yx = h(x) if yx =y, return x.
X0 = {x1,x2,..........,xq}
Ei: event h(xi) =y; 1 ¡= i ¡= Q
Pr[Ei] = 1 M
Pr[Ei] = 1 - 1 M
Pr[E1 $\cup$ E2 $\cup$ E3 $\cup$ ..... $\cup$ EQ] = 1 - Pr[E1^C $\cap$ E2^C $\cap$ E3^C $\cup$ ..... $\cup$ EQ^C]
= 1 - $\prod_{i=1}^{Q}$ Pr[Ei^C]
= 1 - (1 - 1 / M)^Q

= Q / M Pr[Pre image finding] = Q/M
Complexity of finding pre image = O(M)

[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2022-2023
Scribed by: Bhargavi Kamble (202051048) — Lecture (Week 7)

Shiftrows:

Shiftrows: $\{0,1\}^{128} = \{0,1\}^{128}$

Here we have a input of 128 bits and the output is of 128 bits too.
The position of elements are mapped to the positions mentioned in the second matrix.

$$\begin{bmatrix} s00 & s01 & s02 & s03 \\ s10 & s11 & s12 & s13 \\ s20 & s21 & s22 & s23 \\ s30 & s31 & s32 & s33 \end{bmatrix} \rightarrow \begin{bmatrix} s00 & s01 & s02 & s03 \\ s11 & s12 & s13 & s10 \\ s22 & s23 & s20 & s21 \\ s33 & s30 & s31 & s32 \end{bmatrix}$$

Mix column:

Mix Column: $\{0,1\}^{128} = \{0,1\}^{128}$

Here we have a input of 128 bits and the output is of 128 bits too.
$(Sij)_{4*4} \rightarrow (S`ij)_{4*4}$
Consider the column C $\in$ {0,1,2,3}

$$\text{Column} = \begin{bmatrix} S0c \\ S1c \\ S2c \\ S3c \end{bmatrix}$$

Sic = (a7a6....a0)
Polynomial = a0 + a1x + ... + a7x^7
For i =0 to 3
ti = Binary to poly (Sic)
u0 = [(x * t0) + (x + 1) * t1 + t2 + t3] mod (x^8 + x^4 + x^3 + x + 1)
u1 = [(x * t1) + (x + 1) * t2 + t3 + t0] mod (x^8 + x^4 + x^3 + x + 1)
u2 = [(x * t2) + (x + 1) * t3 + t0 + t1] mod (x^8 + x^4 + x^3 + x + 1)
u3 = [(x * t3) + (x + 1) * t0 + t1 + t2] mod (x^5 + x^4 + x^3 + x + 1)
S`ic =polynomial to binary (ui)

$$\begin{bmatrix} s0c \\ s1c \\ s2c \\ s3c \end{bmatrix} \xrightarrow{mixcolumn} \begin{bmatrix} s`0c \\ s`1c \\ s`2c \\ s`3c \end{bmatrix}$$

Mix Column:

$$\begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix} * \begin{bmatrix} s00 & s01 & s02 & s03 \\ s10 & s11 & s12 & s13 \\ s20 & s21 & s22 & s23 \\ s30 & s31 & s32 & s33 \end{bmatrix}$$

mod (x^8 + x^4 + x^3 + x + 1) = (S'ij)$_{4*4}$

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} * (Sij)mod(.) = (S'ij)_{4*4}$$

Find S' sfter joing mix column operation on s:

$$wheres' = \begin{bmatrix} s'00 \\ s'10 \\ s'20 \\ s'30 \end{bmatrix} ands = \begin{bmatrix} s00 \\ s10 \\ s20 \\ s30 \end{bmatrix}$$

where s00 = 95, s10 = 65, s20 = FD and s30 = F3
Solution:
s00 = 95 = 10010101 = $x^7 + x^4 + x^2 + 1$
s10 = 65 = 01100101 = $x^6 + x^5 + x^2 + 1$
s20 = FD = 11111101 = $x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + 1$
s30 = F3 = 11110011 = $x^7 + x^6 + x^5 + x^4 + x + 1$
s'00 = (x * s00) + ((x+1) * s10) + s20 + s30
x * s00 = x * ($x^7 + x^4 + x^2 + 1$)
= $x^8 + x^5 + x^3 + x$
(x * s00) mod ($x^8 + x^5 + x^3 + x$ + 1) = ($x^8 + x^5 + x^3 + x$) mod ($x^8 + x^5 + x^3 + x$ + 1)

$$\begin{array}{r} x^8+x^4+x^3+x+1\overline{)x^8+x^5+x^3+x}(1 \\ \underline{x^8+x^4+x^3+x+1} \\ x^5+x^4+1 \end{array}$$

$x^8 + x^4 + x^3 + x \equiv$ ($x^4 + x^3 + x + 1$) + $x^5 + x^3 + x$
$\equiv x^5 + x^4 + 1$ mod ($x^8 + x^4 + x^3 + x + 1$)
x * s00 = $x^5 + x^4 + 1$
(x + 1) * s10 = (x + 1) * ($x^6 + x^5 + x^2 + 1$)
= $x^7 + x^6 + x^3 + x + x^6 + x^5 + x^2 + 1$
= $x^7 + x^5 + x^3 + x^2 + x + 1$
s'00 = (x * s00) + (x +1) * s10 + s20 + s30
= (x^5+x^4+1) + (x^7+x^5+x^3+x^2+x+1) + (x^7+x^6+x^5+x^4+x^3+x^2+1) + (x^7+x^6+x^5+x^4+x+1)
= $x^7 + x^4$
s'00 = $x^7 + x^4$ = 10010000 = 90

Key Scheduling Algorithm of AES-128:

Input: 128 bit key
Output: 11 round keys, length of each round key is 128 bit.
Key = (Key[15], ..., Key[0])
16 bytes
We will prepare 44 words which are denoted by w[0],......w[43]
ROTWORD (B0, B1, B3, B3) = (B1, B2, B3, B0)
SUBWORD (B0, B1, B2. B3) = (B0‘, B1‘, B2‘, B3‘)
where Bi‘ = SubBytes (Bi) ∀ i =0,..,3
10 round constants (word)
Rcon[1] = 01000000
Rcon[2] = 02000000
Rcon[3] = 04000000
Rcon[4] = 08000000
Rcon[5] = 10000000
Rcon[6] = 20000000
Rcon[7] = 40000000
Rcon[8] = 80000000
Rcon[9] = 1B000000
Rcon[10] = 36000000
for i =0 to 3
w[i] = 9 key [4i], key[4i + 1], key[4i +2], key[4i + 3])
for i = 4 to 43
temp = w[i -1]
if i ≡ 0 mod 4
temp = SUBWORD(ROTWORD(temp)) $\bigoplus$ Rcon[i/4]
w[i] = w[i -4] $\bigoplus$ temp
return (w[0], ...m w[43])

Round keys k1, k2, k3,......, k11
k1 = w[0] || w[1] || w[2] || w[3]
k2 = w[4] || w[5] || w[6] || w[7]
....
k11 = w[40] || w[41] || w[42] || w[43]
We have subbyte (A) = B
A = X || Y
Inv SubByte (b) → find B in the table also find X, Y
X || Y

$$\begin{bmatrix} s00 & s01 & s02 & s03 \\ s10 & s11 & s12 & s13 \\ s20 & s21 & s22 & s23 \\ s30 & s31 & s32 & s33 \end{bmatrix} \rightarrow \begin{bmatrix} s00 & s10 & s20 & s30 \\ s11 & s21 & s31 & s01 \\ ... & ... & ... & ... \\ ... & ... & ... & ... \end{bmatrix}$$

Mix Column:

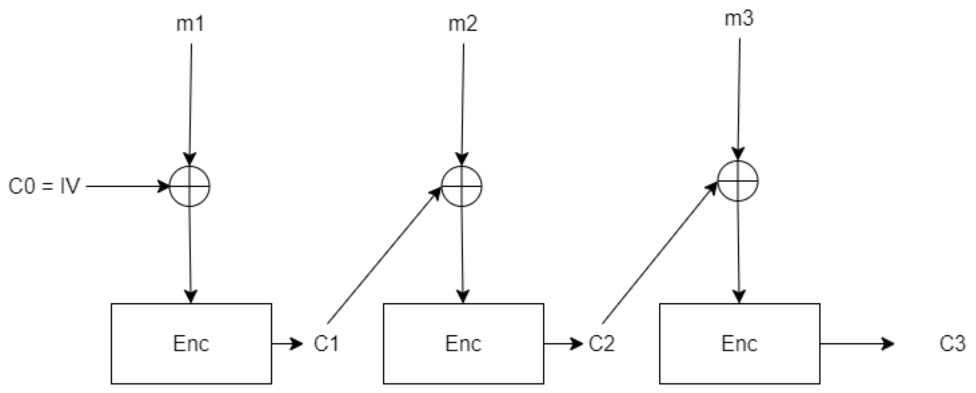
MixCol (S) = S‘
MixCol(MixCol(MixCol(MixCol(S))))

= M^4 * S = I * S
M^4 = I
MixCol(MixCol(MixCol(S‘)))) = S

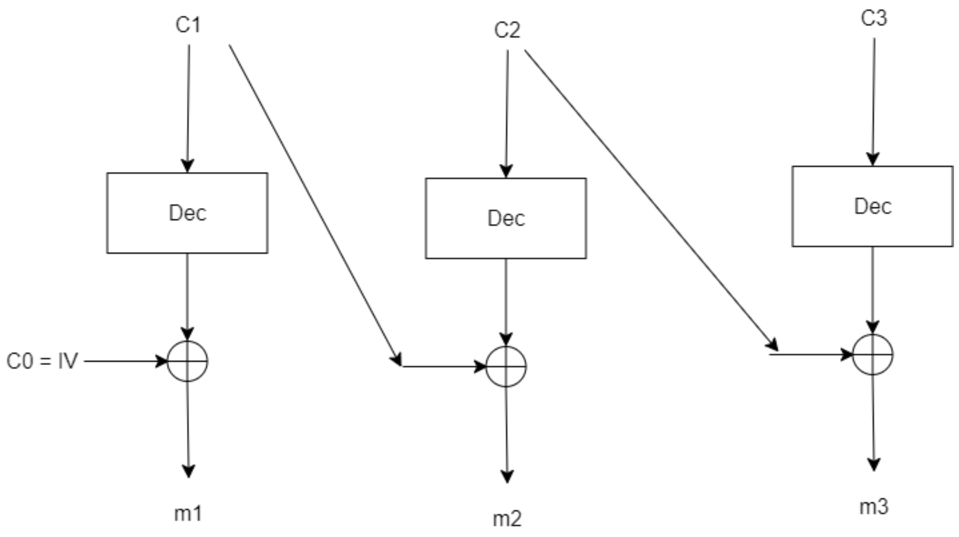
Modes Of Operations:
1. ECB (Electronic Codebook Mode)
2. CFB (Cipher Feedback Mode)
3. CBC (Cipher Block Chaining Mode)
4. OFB (Output Feedback Mode)
5. Counter mode
6. CCM (Counter with Cipher block Mode)
ECB:
M = m0 || m1 || .... || mt
Enc = It can encrypt l bit plaintext block
len(mi) = l bit
Encryption:
C = C0 || C1 || ..... || Ct
Ci = Enc(mi, K) ∀ i =0,....,t

CBC Mode:
M = m1 || m2 || ..... || mt
initialization vector = IV (public variable)
Enc = block size l
len(mi) = l
len(IV) = l
i) c0 = IV
ci = Enc(c_{i-1} $\bigoplus$ mi, k) ∀ i =1,....,t

Encryption:



Decryption:

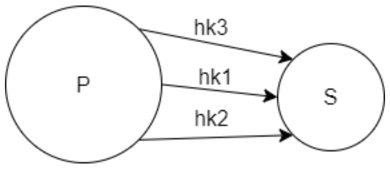


Hash Function:

h: A $\rightarrow$ B
h(x) = Y
1. If X is altered to X‘ then h(X‘) will be completely different from h(X)
2. Given Y it is practically infeasible to find X such that h(X) = Y
3. Given X and Y = h(X) it is practically infeasible to find X‘ such that h(X) = h(x‘)

Alice Bob
X = E(M, K) $\xrightarrow{X}$ X1 = Dec(X‘, K)
S1 = h(M,K) $\xrightarrow{S1}$ S2
If h(X1, K) = s2 then bob accept X1
Definition:
A hash family is a four tuple (P, S, K, H) where the following conditions are satisfied:
1. P is the set of all possible messages.
2. S is the set of all possible message digests or authentication tags.
3. K is the key space.
4. For each K1 $\in$ K there is a hash function h$_k$1 $\in$ H such that

h$_k$1 : P $\rightarrow$ S
here $|P| >= |S|$ more interestingly $|P| >= 2 * |2|$



H: set of all hash functions
h$_k$1: hash function

If key is involved in the computation of hashed value then that hash function is known as keyed hash function.
If the key is not required to compute the hashed value then that hash function is known as unkeyed hash fucntion.
Problem 1:
h: p $\to$ S
Given y $\in$ S find x $\in$ P such that h(x) = y
This problem is known as pre image finding problem.
For on hash function h if you cannot find preimage in a feasible time then h is known as preimage resistant hash function.
Finding preimage is computationally hard for preimage resistant hash function.
Problem 2:
h: P $\to$ S
Given x $\in$ P and h(x) find x' $\in$ P such that x'$\neq$ x and h(x') = h(x)
This prolblem is known as second pre image finding problem.
If finding second pre image is computationally hard for h then h is known as second pre image resistant hash function.
Problem 3:
h: P $\to$ S
Find x, x' $\in$ P such that x *neq* x' and h(x) and h(x')
This problem is known as collision finding problem.
For an hash function h if finding collision is computationally hard then h is known as collision resistant hash function.

Ideal Hash Function:

Let h: P $\to$ S be an hash function.
h will be called ideal hash function if given x $\in$ P to find h(x) either you have to apply h on x or you have to look into the table corresponding to h (hash table).

Pre Image Finding Algorithm:

Given y $\in$ Y find x $\in$ X such that h(x) = yx
h: X $\to$ Y
$|Y|$ = m
Y = { y1, y2,...., ym} x1 $\to$ y1, x2 $\to$ y2, ..., xm $\to$ ym
h: X $\to$ Y
choose any X0 $\subseteq$ X such that $|X0|$ = Q for each X $\in$ X0.
Compute yx = h(x) if yx =y, return x.
X0 = {x1,x2,..........,xq}
Ei: event h(xi) =y; 1 ¡= i ¡= Q
Pr[Ei] = 1 M
Pr[Ei] = 1 - 1 M
Pr[E1 $\cup$ E2 $\cup$ E3 $\cup$ ..... $\cup$ EQ] = 1 - Pr[E1^C $\cap$ E2^C $\cap$ E3^C $\cup$ ..... $\cup$ EQ^C]
= 1 - $\prod_{i=1}{}^Q$ Pr[Ei^C]
= 1 - (1 - 1 / M)^Q

= Q / M Pr[Pre image finding] = Q/M
Complexity of finding pre image = O(M)

[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2022-2023
Scribed by: Bhargavi Kamble (202051048) — Lecture (Week 8 9)

Second Preimage:

h: A $\rightarrow$ B
X0 $\subseteq$ A $|X0|$ = Q
h(xi) where xi $\in$ X0
$|B|$ = M
h(x)
x,h(x) ST h(x') = h(x) where xi $\in$ x0
X0 $\subseteq$ A {x}

Collision finding algorithm:

h: X $\rightarrow$ Y
— Y — = M
Find x, x' $\in$ X s.t, x $\neq$ x' and h(x) = h(x')
x0 $\subseteq$ x
$|x0|$ = Q
for each x $\in$ x0
x= {x1,...,xq}
compute yx = h(x)
if yx = yx' for some x $\neq$ x' then return (x, x')
else return failure.
Ei: event h(xi) $\notin$ {h(x1), h(x2), ..., h(xi-1)}
X0 = {x1,....,xq}
Pr[E1] = 1
E2: h(x2) $\notin$ {h(x1)}
h(x2) = Z
E2: h(x2) $\notin$ {h(x1)}
Pr[E2 | E1] = M-1/M therefore,
Pr[E1 $\cap$ E2] / Pr [E1] = M - 1/ M
Through this we can write,
Pr[E1 $\cap$ E2 $\cap$ E3 ... EQ]
= M-1/M * M -2/M * M-3/ M.... * M-(Q-1)/M
h: X $\rightarrow$ Y
Preimage $\rightarrow$ O($|Y|$)
Collision $\rightarrow$ O(sqrt($|Y|$))
Second preimage $\rightarrow$ O($|Y|$)
h: {0,1}* $\rightarrow$ {0,1}^{m}
h is the secure hash function.
Preimage $\rightarrow$ O(2^m)
Collision $rightarrow$ O($2^{m/2}$)

Compression function:

h: {0,1}$^{m+t}$ $\rightarrow$ {0,1}^{m}
Preimage $\rightarrow$ O(2^m)
Collision $\rightarrow$ O($2^{m/2}$)
Target: Construct H: {0,1}* $\rightarrow$ {0,1}^{m}
Securith od H will ompletely depend on security of h.
Given x$\in$ {0,1}*
$|x|$ = length of x
$|x|$ >= m + t + 1
From x construct y by using a public function s.t: $|y| \equiv 0$(mod t)

y = y1 || y2 || ... || yr
s.t: $|yi|$ = t
Z0 = IV
Z1 = h(z0 || y1)
Z2 = h(z1 || y2)
and so on.
Zr = H(x)
This is known as iterative hash funtion.

Length Extension Attack

Suppose we have a message $M = y_1||y_2||...||y_r$ and the final hash value of this message is Z_r. Let $Z_i = h(Z_{i-1}||y_i)$ for $1 \leq i \leq r$ and $Z_0 = IV$, where IV is the initial value. Now, given (M, Z_r), we want to find (M_2, Z_r) without computing $H(M_2, Z_r)$. We can do this by defining M_2 as $M||y_{r+1}$ and computing $Z = h(Z_r||y_{r+1})$. Then, (M_2, Z_r) is $(M||y_{r+1}, Z)$.

Merkle-Damgard Principle

The Merkle-Damgard principle is a method of constructing a hash function by using a compression function to process fixed-length blocks of the message iteratively. Here's an implementation of the Merkle-Damgard construction:

Given a message x of length n and a compression function *compress* that maps $m+t$-bit strings to m-bit strings, where $t \geq 2$:

Let $k = \lceil \frac{n}{t-1} \rceil$ and $d = k(t-1) - n$.
For $i = 1$ to $k-1$, define y_i as the $(t-1)$-bit string $x_{(i-1)(t-1)+1}||x_{(i-1)(t-1)+2}||...||x_{i(t-1)}$, and let y_k be the $((t-1)-d)$-bit string
$x_{(k-1)(t-1)+1}||x_{(k-1)(t-1)+2}||...||x_n||0^d$. Let $Z_1 = 0^{m+1}||y_1$ and $g_1 = compress(Z_1)$.
For $i = 1$ to $k-1$, let $Z_{i+1} = g_i||1||y_{i+1}$ and $g_{i+1} = compress(Z_{i+1})$.
Finally, let $h(x) = g_{k+1}$ and return $h(x)$.
Here, g_i represents the hash value after processing the first i blocks of the message. The construction pads the message to be a multiple of the block size and uses a final block with a special bit pattern to indicate the end of the message. The compression function takes the current hash value and the next block of the message as input and produces a new hash value as output.

SHA

Secure Hash Algorithm
1. SHA - 160
2. SHA - 256
3. SHA - 512

SHA - I PAD (x)

$|x|$ <= 2^64 -1
d = (447 - $|x|$) mod 512
l = binary ($|x|$)
y = x || 1 || O^d || l
ROTL^S(X): Circular left shift on X by S position.
ft(B,C,D)
= $(B \wedge C) \vee ((\neg B) \vee D)$ 0 <= t <= 19
= $B \bigoplus C \bigoplus D$ 20 <= t <= 39
= $(B \wedge C) \vee (B \vee D) \vee (C \vee D)$ 40 <= t <= 59
= $B \oplus C \oplus D$ 60 <= t <= 79

SHA-I (x)

y = SHA-I-PAD (x)
y = M1 || M2 || ... || Mn
$|Mi|$ = 512
H0 = 67452301
H1 = EFCDAB89
H2 = 98BADCFE
H3 = 10325476
H4 = C3D2E1F0
kt = 5A827999 0 <= t <= 19
= 6ED9EBA1 20 <= t <= 39
= 8F1BBCDC 40 <= t <= 59
= CA62C1D6 60 <= t <= 79
for i=1 to n
$M_i = w_0||w_1||w_2||...||w_{15}$ $\quad w_i$ =32
for t=16 to 79
    $w_t = ROTL^1(w_{t-3} \oplus w_{t-8} \oplus w_{t-14} \oplus w_{t-16)}$
A = H0
B = H1
C = H2
D = H3
E = H4
Using the for loop as mentioned above we get,
E = D
D = C
C = ROTL^{30} B

B = A
A = temp
H0 = H0 + A
H1 = H1 + B
H2 = H2 + C
H3 = H3 + D
H4 = H4 + E
return (H0 || H1 || H2 || H3 || H4)

Message Authentication Code (MAC)

Alice **Bob**

$C = E(M, K) \xrightarrow{\quad\quad\quad C \quad\quad\quad} \sim C$

$MAC = h(M, K) \xrightarrow{\quad\quad\quad MAC \quad\quad\quad} \sim MAC$

Dec$(\sim C, K) = \sim M$
$h(\sim M, K) = \text{MAC}_1$
If $\sim MAC = \text{MAC}_1$
then accept $\sim M$ as $\sim M = M$

Hash-based Message Authentication Code (HMAC)

ipad = 363636... $\rightarrow$ 512 bits
opad = $5c5c5c$... $\rightarrow$ 512 bits
$K \rightarrow$ secret key
SHA $\rightarrow$ SHA -1
$\text{HMAC}_K(x) = \text{SHA}((K \oplus \text{opad}))||\,\text{SHA}((K \oplus \text{ipad})||x)$

CBC-MAC (Cipher Block Chaining - Message Authentication Code)

$x = (x_0||x_1||x_2||x_3||...||x_n)$
IV = 00...0
$y_0 = \text{IV}$
for $i = 1$ to n
$y_i = \text{Enc}((y_{i-1} \oplus x_i), K)$ return y_n

OTP:

C = M $\bigoplus$ K
1. C1 = M1 $\bigoplus$ K
C2 = M2 $\bigoplus$ K
2. len(K) >= len(M)
F(K, IV) = Zi $\in$ {0,1}
where K is the secret key and IV is the public parameter Initialization Vector.
We have Z0,Z1,....Zn-1 we'll xor it with m0,m1,...mn-1. To get c0 = m0 $\bigoplus$ z0, c1, m1 $\bigoplus$ c1, ..., cn-1 = mn-1 $\bigoplus$ zn-1.

1. Here Z will be a random looking string.
2. If we give same input (K, IV) in different time it will generate same zi.
3. F(K, IV) if K is selected randomly and is kept secret then the output Z0,...,Zn-1 will be indistinguishable from bits string generated by using a random bit generator.
F(K, IV) = Z0....,Zn-1
C0in = x0,.....,xn-1
4. F(K, IV) is a pseudorandom bit generator F(K, IV) = Zi 0 $<= i <= n-1$.
5. F(K, IV) = Zi 0 $<= i <= n-1$.
length of the output bits will be >>> length of K. IV is public whereas K is a secret key of 80 bit.
6. F(K, IV) if we modify atleast one bit of IV or K then there will be an unpredictable change in the output (Zi).
If F(K, IV1) = Zi (1) 0 $<= i <= n-1$
and F(K, IV2) = Zi (2) 0 $<= i <= n-1$
Zi(1), Zi(2) are uncorrelated.

Stream Cipher:

It is of two types:
1. **Synchronous Stream Cipher**
2. **Self Synchronizing or Asynchronous Stream Cipher**

Synchronous Stream Cipher:

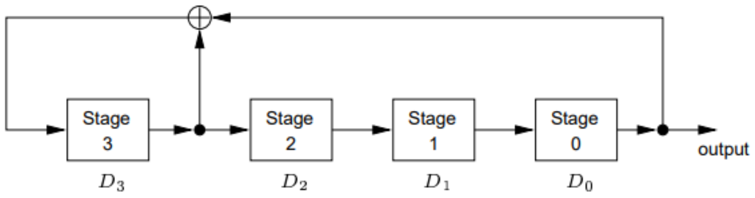
A synchronous stream cipher is one in which the keystream is generated independently of the plaintext bits and the ciphertext bits.
Stream update function: Si+1 = f(Si,K)
KeyStream generation function: Zi = g(Si, K)
Ciphertext generation function: Ci = h(Zi, Mi)
Here S0 is the initial state and may be determined from the secret key K and IV.

Self Synchronizing or Asynchronous Stream Cipher:

A self synchronizing stream cipher is one in which the key stream bits are generated as a function of the key and a fixed number of previous ciphertext bits.
State function: σi = (Ci-t, Ci-t+1,...,Ci-1)
Keystream generation function: Zi = g(σi, K)
Ciphertext generation function: Ci = h(Zi, Mi)
Here σ0 = (C-t, Ct+1,...,C-1) is the non secret initial state.
Zi = g(Si, K)
Zi $\in$ {0,1} = Key Stream bits.
Ci = h(Zi, Mi) = Cipher text bits.
Ci = mi $\bigoplus$ zi

0.1 LFSR:

Linear Feedback Shift Register
Si ∈ {0,1} i=0,...,n-1
Suppose we have a register of n bits it will be represented as
$Sn-1|Sn-2|......|S0$
At t=0,
S0 = $Sn-1|Sn-2|......S1|S0|$
At t=1,
S1 = $|*|Sn-1|......|S2|S1|$ S0 = Output
Feedback bit = Sn = * = L(S0,S1,....,Sn-1)=L(S0)
At t=2,
S2 = $|*|Sn|Sn-1|......|S2|$ S1 = Output
Feedback bit = Sn+1 = * =L(S1)



L(S0,...,Sn-1) = Sn ∈ {0,1}
L:$\{0,1\}^n \rightarrow \{0,1\}$
La = a0.s0 $\bigoplus$ a1.s1 $\bigoplus$..... an-1.sn-1
ai ∈ {0,1}

Period of an LFSR:

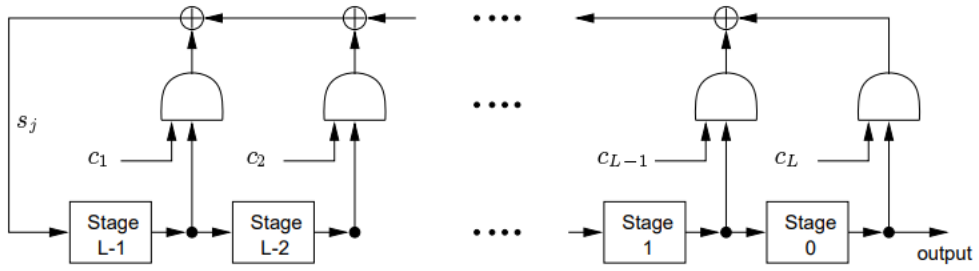
S0 is non zero initial state S0 will be repeated after m clocking of the LFSR then m will be the period of the LFSR.
eg:

t	Stage 2	Stage 1	Stage 0
0	0	0	0
1	1	0	0
2	1	1	0
3	1	1	1

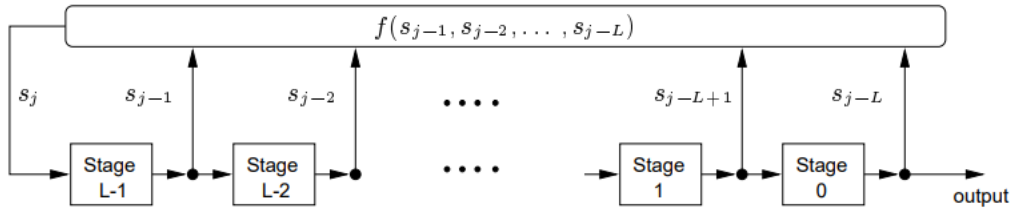
t	Stage 2	Stage 1	Stage 0
4	0	1	1
5	1	0	1
6	0	1	0
7	0	0	1

for the above eg the period will be of 7.
An n-nit LFSR will be called full periodic if the period of the LFSR is 2^n -1.
Suppose we have a block of 1|1|1|
Now at t = 0 the block will be,
1|1|1|
At t=1 it will be, 1|1|1|
Period = LCM{P1,P2,...,PN}
If I consider a zero state consider any linear feedback function then every time zero state will be generated.
Consider an n bit lfsr,
$Sn-1|Sn-2|......|S0$

Sn = L(S0,...,Sn-1)
= c1.sn-1 $\bigoplus$ c2.sn-1... $\bigoplus$ cn.s0
Sn+1 = L(S1,...,Sn)



L = c1.sn-1 $\bigoplus$ c2.sn-1... $\bigoplus$ cn.s0
f(x) = 1 + c1x + c2x^2 +....+ cnx^n
f(x) $\in$ F2[x]
ci $\in$ 0,1
n bit LFSR $\longleftrightarrow$ Liner feedback function $\longleftrightarrow$ polynomial of degree n in F2[x]
So repeats adter 2^n -1 clocking and hence is full periodic LFSR.
1. If the connection polynomial is primitive polynomial then the corresponding LFSR will have full period.
n-bit LFSR $\rightarrow$ period will be 2^n -1.
2. If the polynomial is irreducible then the period of the LFSR will divide 2^n -1.
3. If it is reducible then different state will have ifferent cycle length (different period).
Suppose we have a n-bit LFSR,
$|Kn-1|Kn-2|....|K0|$
Xi are the output bits.
K = (K0,...,Kn-1)
Output bits Xi = Keystream bits Zi
mi $\bigoplus$ zi = ci = ciphertext bits.
Zi = L(K0,...,Kn-1)
o = K2 $\bigoplus$ K6 $\bigoplus$
1) Known Plaintext attack:
Zi = mi $\bigoplus$ ci
Z0,....,Zn-1
Z0=K0, Z1=K1 ,...., Zn-1 = Kn-1
If I give you keystream bits then you will be able to prepare a system liner equations. By solving the system you will get back the state.

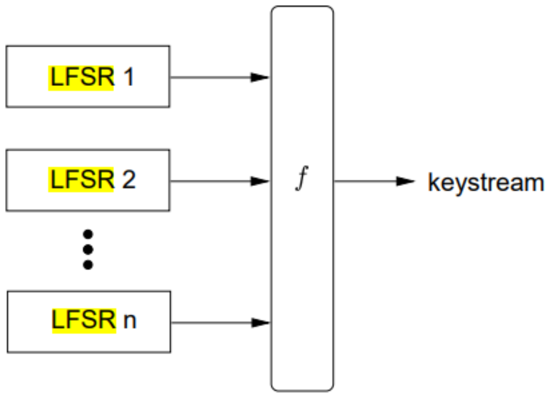


Suppose Zi are the outputs from function f
Zi $\in$ {0,1}
Ci = mi $\bigoplus$ zi

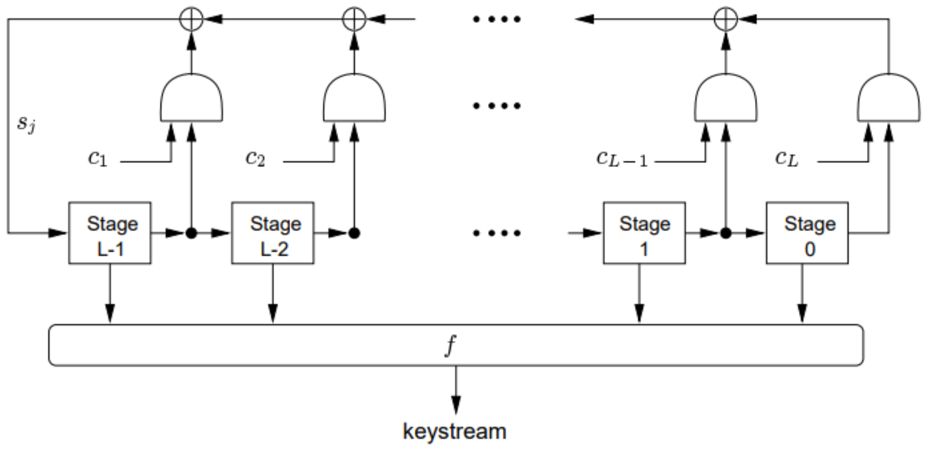
Here state update of LFSR takes place by:
1) Linear feedback
2) Shifting
The state update function of LFSR is α
St+1 = α(St)
Z(t+1) = f(St+1)
t-th state:
S^{t}n-1, S^{t}n-2,...,S^{t}0
(t+1)-th state
S^{t+1}, S$^{t+1}n-2$,...,S^{t+1}0 where S$^{t+1}n-1$, S$^{t+1}n-2$,...,S^{t+1}0
where S^{t+1}0=S^{t}1, S^{t+1}1, S^{t}2....S^{t+1}n-2 = S^{t}n-1, S^{t+1}n-1 = L(S^{t}0, S^{t}n-1)

$$S^{t+1} = \begin{bmatrix} S_0^{t+1} \\ S_1^{t+1} \\ \dots \\ \dots \\ S_{n-1}^{t+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 & \dots & \dots & \dots & 0 \\ 0 & 0 & 1 & \dots & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & \dots & \dots & 1 \end{bmatrix} \begin{bmatrix} \text{S}^t0 \\ \text{S}^t1 \\ \text{..........} \\ \text{S}^t\text{n-1} \end{bmatrix}$$

L = Cn-1S0 $\bigoplus$ cn-2s1.... $\bigoplus$ c0sn-1
LFSR with combiner function: f: {0,1} $\rightarrow$ {0,1}



In Non Linear feedback bit shift register (NFSR) feedback function is non linear.
Another general technique for destroying the linearity inherent in LFSRs is to generate the keystream as some nonlinear function of the stages of a single LFSR. Such keystream generators are called nonlinear filter generators, and f is called the filtering function.


s_j
c_1
c_2
c_{L-1}
c_L
Stage L-1
Stage L-2
Stage 1
Stage 0
f
keystream

[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy Winter 2022-2023
Scribed by: Bhargavi Kamble (202051048) Lecture (Week 10)

Grain Stream Cipher:

It is based on NLFSR or LFSR and non linear filter function. In LFSR the feedback used to be linear unlike NLFSR where the feedback was non linear. Most of the stream ciphers are based on NLFSR. We will be performing shifting operations in NLFSR similar to LFSR except for the feedback function.
The feedback function for LFSR is defined by f(x) for LFSR whereas for NLFSR it is defined by g(x).
The state bits are denoted by si LFSR whereas for NFLSR it is denoted by bi.
We will first observe the 80 bit NLFSR feedback function.
If we have a linear feedback function we can construct a polynomial.Grain is a synchronous stream cipher that operates on a 80-bit key and a 64-bit initialization vector (IV) to generate a stream of key stream bits. The key and IV are used to initialize the internal state of the cipher, which is then used to generate the key stream bits.The internal state of Grain consists of 80 bits, divided into 5-bit segments, labeled S_0 through S_{15}. The internal state is updated at each clock cycle by the following procedure:

1. The bits $S_0, S_1, \ldots, S_{14}$ are shifted to the left by one bit, with S_0 becoming the new value of S_{15}.
2. The feedback bit f is calculated as the output of the following non-linear function:
$$f = S_0 \oplus S_1 \oplus S_2 \oplus S_3 \oplus S_4 \oplus S_6 \oplus S_{10} \oplus S_{15} \oplus IV_0 \oplus IV_3 \oplus IV_5 \oplus IV_7 \oplus IV_{10} \oplus IV_{12} \oplus IV_{13}$$
3. The bit S_{15} is updated as follows:
$$S_{15} = f \oplus S_0 \oplus K_0$$
4. The output bit z is calculated as follows:
$$z = S_2 \oplus S_{15} \oplus K_1$$

The feedback polynomial of the LFSR, $f(x)$, is a primitive polynomial of degree 80. It is defined as:

$$f(x) = x^{80} + x^{45} + x^{13} + x^3 + 1$$

This polynomial is irreducible over the binary field $\mathbb{F}_2$, meaning that it cannot be factored into the product of two non-constant polynomials with coefficients in $\mathbb{F}_2$. Its primitive nature means that it generates all non-zero elements of the binary field $\mathbb{F}_{2^{80}}$ when used as a generating polynomial for an LFSR with 80 bits.

The feedback polynomial $f(x)$ is used in Grain to generate the feedback bit f during each clock cycle of the internal state update procedure.

To generate the key stream, the internal state of Grain is updated for 2n clock cycles, where n is the number of key stream bits to be generated. The output key stream bits are the values of z generated by the internal state update procedure at each clock cycle.

To remove any possible ambiguity we also define the update function of the LFSR as:

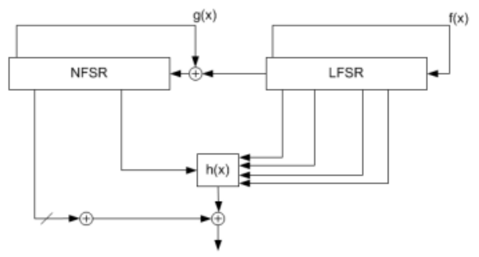
$$S_{i+80} = S_{i+62} + S_{i+51} + S_{i+38} + S_{i+23} + S_{i+13} + S_i$$

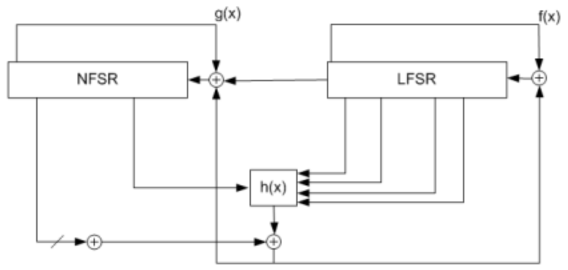
where i is the current index of the LFSR state. This equation defines the operation that updates the LFSR state from one clock cycle to the next. The notation S_{i+80} represents the state bit that has an index of $i + 80$ in the LFSR state vector, and similarly for the other state bits. Note that this update function is equivalent to shifting the state vector to the left by 1 bit and computing the new value of the least significant bit as the XOR of a subset of the state bits, with coefficients given by the feedback polynomial $f(x)$.

The feedback polynomial of the NFSR, $g(x)$, is defined as
$g(x) = 1 + x^{18} + x^{20} + x^{28} + x^{35} + x^{43} + x^{47} + x^{52}x^{59} + x^{66} + x^{71} + x^{80} + x^{17}x^{20} + x^{43}x^{47} x^{65}x^{71}+x^{20}x^{28}+x^{35}x^{47}x^{52}+x^{59}x^{65}\ x^{17}x^{35}x^{52}+x^{71}x^{80}+x^{20}x^{28}x^{43}x^{47}\ x^{17}x^{20}x^{59}x^{65}+x^{17}x^{20}x^{28}x^{35}x^{43} x^{47}x^{52}x^{59}$.
Again, to remove any possible ambiguity we also write the update function of the NFSR. Note that the bit s_i which is masked with the input is included in the update function below.
$b_{i+80} = s_i + b_{i+62} + b_{i+60} + b_{i+52} + b_{i+45} + b_{i+37} + b_{i+33} + b_{i+28} + b_{i+21}\ \ b_{i+14} + b_{i+9} + b_i + b_{i+63}b_{i+60} + b_{i+37}b_{i+33} + b_{i+15}b_{i+9}\ \ b_{i+60}b_{i+52}b_{i+45} + b_{i+33}b_{i+28}b_{i+21} + b_{i+63}b_{i+45}b_{i+28}b_{i+9}\ \ b_{i+60}b_{i+52}b_{i+37}b_{i+33} + b_{i+63}b_{i+60}b_{i+21}b_{i+15}\ b_{i+63}b_{i+60}b_{i+52}b_{i+45}b_{i+37} + b_{i+33}b_{i+28}b_{i+21}b_{i+15}b_{i+9}\ b_{i+52}b_{i+45}b_{i+37}b_{i+33}b_{i+28}b_{i+21}$.
Before generating any keystream, the cipher must be initialized with the key and the initialization vector (IV). Let the bits of the key be denoted as k_i, $0 \leq i \leq 79$, and the bits of the IV be denoted as IV_i, $0 \leq i \leq 63$.

The initialization of the key is done as follows:

1. Load the non-linear feedback shift register (NFSR) with the key bits: $b_i = k_i$, $0 \leq i \leq 79$
2. Load the first 64 bits of the linear feedback shift register (LFSR) with the IV: $s_i = IV_i$, $0 \leq i \leq 63$.
3. Fill the remaining bits of the LFSR with ones: $s_i = 1$, $64 \leq i \leq 79$. This ensures that the LFSR cannot be initialized to the all-zero state.
4. Clock the cipher 160 times without producing any running key. Instead, the output function is fed back and XORed with the input, both to the LFSR and to the NFSR.





RC4

The RC4 cipher has two components, namely, the Key Scheduling Algorithm (KSA) and the Pseudo-Random Generation Algorithm (PRGA). The KSA uses the key K to shuffle the elements of S and the PRGA uses this scrambled permutation to generate pseudo-random keystream bytes.

Two indices, i and j, are used in RC4. i is a deterministic index that is incremented by 1 (modulo N) in each step and j serves as a pseudo-random index that is updated depending on the secret key K and the state S. The KSA initializes both i and j to 0, and S to be the identity permutation. It then steps i across S looping N times, and updates j by adding the i-th entries of S and K. Each iteration ends with a swap of the two bytes in S pointed by the current values of i and j. **Input:** Secret key array $K[0 \dots N-1]$. **Output:** Scrambled permutation array $S[0 \dots N-1]$.
Initialization: $i = 0, \dots, N-1$
$S[i] = i$
$j = 0$
Scrambling: $i = 0, \dots, N-1$
$j = (j + S[i] + K[i]) \bmod N$
Swap$(S[i], S[j])$

Public Key Cryptography:

Alice **Bob**
K K
$C = E(M, K) \xrightarrow{\qquad C \qquad} M = Dec(C, K)$
Alice and Bob are using the same secret key, denoted as K, to encrypt and decrypt a message M, the ciphertext C can be obtained by encrypting M with K. Therefore, to obtain the original message M from the ciphertext C, it is necessary to decrypt C using the same key K, which can be expressed as M = Dec(C, K).

0.1 Diffie and Hellman (1976):

Diffie-Hellman key exchange algorithm introduces new area in cryptography known as public key cryptography.
It's publishes in IEEE Transactions on Information Theory.
Alice g^i G → Cyclic group = $< g >$ Bob
(G, *)
1. a,b ∈ G then a * b ∈ G.

2. ∃ an element e ∈ G s.t. a * e = a = e * a. Where e is the identity element og G.
3. a * (b * c) = (a * b) * c
4. For element a ∈ G there exists an inverse element a^{-1}.
The shared secret key can be written as g^{ab}. This can be shown as follows: In a group $(G, \cdot)$, for all $a \in G$, there exists $g \in G$ and $i \in \mathbb{Z}$ such that $a = \underbrace{g \cdot g \cdot \ldots \cdot g}_{i \text{ times}}$ and $G = \langle g \rangle$. Here, $e = g^0$ and $g = g^1$.
Alice chooses a secret integer a and computes $A = g^a \bmod p$. She sends A to Bob. Bob chooses a secret integer b and computes $B = g^b \bmod p$. He sends B to Alice. Alice computes the shared secret key $K = B^a \bmod p$. Bob computes the shared secret key $K = A^b \bmod p$. Now both Alice and Bob have the same shared secret key K.

It is important to keep in mind that, in reality, the values of p and g are typically set to be high enough to make computing discrete logarithms modulo p computationally impossible.

$$T_a B = g^a \cdot g^b \bmod p \qquad = g^{a+b} \bmod p \ = g^{b+a} \bmod p \qquad = g^b \cdot g^a \bmod p \ = T_b A \bmod p$$

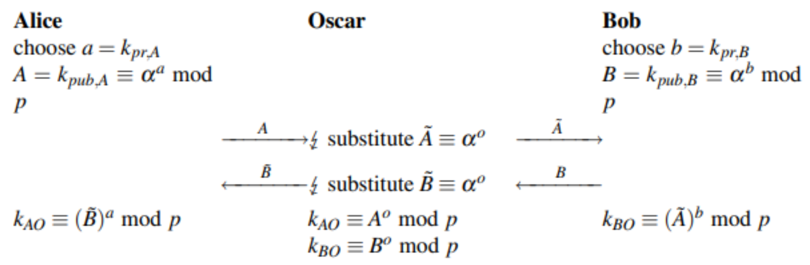
Therefore, the shared secret key is g^{ab}.
Based on the property of group G = < g > finding x from g^x will be computationally difficult. This hard problem is known as Discrete Log Problem.
y = g^x
$log_g y$ = x
for(i =2; i ¡ n-1;i++)
if(g^x == g^i)
t=i;
break;
1. $|G|$: large
2. *: good.

$$\begin{tikzcd}
\textbf{Alice} & & \textbf{Bob} \\
g^a \arrow[rr, leftrightarrow, "g^a"] & & g^b \\
(g^b)^a & = & (g^a)^a
\end{tikzcd}$$

(g^a) it is hard to find a.

0.2 Man in middle attack

Man-in-the-Middle Attack Against the DHKE


Alice
choose $a = k_{pr,A}$
$A = k_{pub,A} \equiv \alpha^a$ mod p
Oscar
Bob
choose $b = k_{pr,B}$
$B = k_{pub,B} \equiv \alpha^b$ mod p
A
substitute $\tilde{A} \equiv \alpha^o$
$\tilde{A}$
$\tilde{B}$
substitute $\tilde{B} \equiv \alpha^o$
B
$k_{AO} \equiv (\tilde{B})^a$ mod p
$k_{AO} \equiv A^o$ mod p
$k_{BO} \equiv B^o$ mod p
$k_{BO} \equiv (\tilde{A})^b$ mod p

Oscar intercepts the communication and establishes separate secret keys with Alice and Bob. He then relays messages between them while decrypting and re-encrypting with the respective secret keys, without either of them knowing that Oscar is intercepting their communication. This allows Oscar to read and modify messages being exchanged between Alice and Bob, essentially allowing him to execute a man-in-the-middle attack.

0.3 Square and Multiply:

Compute: $x^c = \sum_{i=0}^{l-1} C_i 2^i$,
where $C_i \in 0, 1$ and
$C \to C_{l-1}, \ldots, C_0$ is the binary representation of C. $z = 1$
for $i = l - 1$ to 0, do:

$$z = z^2$$

$$\text{if } C_i = 1, \text{ then } z = z * x$$

Return $x^c = z$.

0.4 RSA

RSA is first Algorithm in public key setup. $\phi(m)$ is the number of positive integers smaller than m which are co-prime to m.
$\phi(8) = 4$ {1, 3, 5, 7}
$\phi(p) = p - 1$ p: prime
$\phi(p^k) = p^k - p^{(k-1)}$
$\gcd(a, m) = 1$
$S = x \mod m$
$S = r_1, r_2, \ldots, r_m$
$S_1 = ar_1, ar_2, \ldots, ar_m$
It is necessary to have $\gcd(a, m) = 1$ to have unique characters.
Proof:
Let $ar_i = ar_j$, $i \neq j$ $ari \equiv arj \pmod{m}, r_i \neq r_j \pmod{m}$ $\gcd(a, m) = 1$ $1 = ab + ms$ $\exists$ b s.t. $ab \equiv 1 \pmod{m}$ $ari \equiv arj \pmod{m}$ $bari \equiv barj \pmod{m}$ $ri \equiv rj \pmod{m}$

[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2022-2023
Scribed by: Bhargavi Kamble (202051048) — Lecture (Week 11)

gcd(a,m) =1
S={x mod m}
= {r1,r2,...,rm}
S1 ={ ar1, ar2,..., arm}
ari = arj   ri $\neq$ rj
ari $\equiv$ arj(mod m)   ri $\not\equiv$ arj
gcd(a,m) =1
1 = a.b + m.s
$\exists$ b s.t. a.b $\equiv$ 1 (mod m)
ari $\equiv$ arj mod m
b.a ri $\equiv$ b.a rj mod m
ri $\equiv$ rj mod m
ari $\not\equiv$ arj (mod m)

Euler's Theorem:

If gcd(a,m) =1 then ϕ(m) $\equiv$ 1 mod m
S = { x | gcd(x,m) = 1}
= { s1, s2, ..., s$_{\phi(m)}$ }
gcd(si, m) =1
S1 = {a.s1, a.s2,..., a.s$_{\phi(m)}$ }
asi $\not\equiv$ asj (mod m)
If asi $\equiv$ asj (mod m)
If asi $\equiv$ asj (mod m)
si $\equiv$ sj(mod m)
gcd(a,m) =1
b.a $\equiv$ 1 (mod m)
$|S|$ = ϕ(m)
$|S1|$ = ϕ(m)
Si = asj(mod m)

$$\prod_{i=1}^{\phi(m)} S_i \equiv \prod_{j=1}^{\phi(m)} aS_j (modm)$$

$$\prod_{i=1}^{\phi(m)} S_i \equiv \prod_{j=1}^{\phi(m)\phi(m)} aS_j (modm)$$

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

Fermat's Theorem:

If p is a prime number and p X a then $a^{p-1} \equiv 1$ mod p
ϕ(p) = p - 1
p X a => gcd(a,p) =1
=> $a^{\phi(p)}$ = 1 (mod p)
=> $a^{(p-1)}$ = a (mod p)
=> $a^{(p)}$ = a (mod p)

RSA Cryptosystem:

Few facts:
1) g(a,m) = 1 then, $a^{\phi(p)} \equiv 1$ (mod m).
2) $a^{p-1} \equiv 1$ (mod p)
RSA: Rivest, Shamir, Adleman in 1977
i) n = pq here p,q are primes
ii) plaintext space = Zn
ciphertext space = Zn
iii) Key space = {K=(n,p,q,e,d) | ed $\equiv$ 1 (mod $\phi(n)$ }
iv) Encryption:
E(X,K) = C
C = E(x,k) = x^e (mod n)
v) Decryption:
Dec(C,k) = x
x = Dec(C,k) = c^d(mod n)
ed $\equiv$ 1 mod ϕ(n)
=>ed -1 = t. ϕ(n)
=> 1 = e.d + t1ϕ(n)
1 = gcd(e,ϕ(n)) = e.d + t1ϕ(n)
Encryption:
c = x^e(mod n)
Decryption:
x = c^d(mod n)
c^d = $(x^e)^d$ mod n
= x^{ed} mod n
= $x^{1+t.\phi(n)}$ mod n
e.d $\equiv$ 1 (modϕ(n))
e.d -1 = t.ϕ(n)
ed = 1 + t.ϕ(n)
= x.$x^{t.\phi(n)}$ mod n
= x.$x^{t.[(p-1).(q-1)]}$ mod n
= x.$x^{t.[(p-1).(q-1)]}$ mod (pq)
Since, n = p.q
ϕ(n) = ϕ(p.q)
= (p-1)(q-1)
$x^{t.[(p-1).(q-1)]}$ mod (pq)
we check,

x$^{t.[(p-1).(q-1)]}$ mod (pq)
≡(x$^{p-1)t(q-1)}$ mod (p)
≡ 1 (mod p)
Now we check x$^t(p-1)(q-1)$ (mod q)
≡(x$^{q-1)t(p-1)}$ mod (p)
≡ 1
we finally have,
x$^{t(p-1)(q-1)}$ ≡ 1 (mod pq)
We have {n,e} = public

Alice	Bob
n = p.q	
e.d ≡ 1 (modϕ(n))	y = x^e(mod n)
Public key = n,e	
Secret key = p,q,d	
y^d mod n = x	$\overset{y}{\leftarrow}$

p,q = large primes.
Finding p,q from n is conditionally hard problem.
n= pq
Solve factorization = Solve RSA
RSA Problem:
(n,e) = public
c = x^e
find x
Solve RSA problem = factor n

[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2022-2023

Scribed by: Bhargavi Kamble (202051048) — Lecture (Week 12)

Alice Bob

Sign or m

n = p.q

Q(n) = (p-1)(q-1)

e.d $\equiv$ 1(mod Q(n))

PK: e,n v = s^e mod n

SK: d,p,q $\xrightarrow{s}$ if v =m then output =1 else output =0

S = m^d mod n $\xleftarrow{y}$encryption

S(Signature) = m^d mod n $\xrightarrow{S}$ m(verification) = S^e mod n

The above represents RSA signature algorithm.

RSA encryption:

n = p.q

Q(n) = (p-1)(q-1)

e.d = 1(mod Q(n))

c = m^e(mod n)

c1 = m^{e}1 mod n

c2 = m^{e}2 mod n

c1 * c2 = m^{e}1m^{e}2 mod n = (m1m2)^e mod n

s1 = m1^d(mod n)

s2 = m2^d(mod n)

s1*s2 = (m1*m2)^d (mod n)

Sign (M,K) = S

f(S) = Sign(f(M),K) (Computation on authenticated data)

S1 = (h(m1))^d mod n

S2 = (h(m2))^d mod n

S1S2 $\neq$ (h(m1m2))^d mod n

ax $\equiv$ b(mod m)

=> ax - my = b

=> ax0 + my0 = gcd(a,m)

If gcd(a,m) | b then ax - my = b will have solution.

gcd(a,m) | b => t.gcs(a,m) = b

= ax0 + my0 = gcd(a,m)

ax $\equiv$ b(mod m)

=> ax - my = b

x = x0 + m / gcd(a,m) n

y = y0 + a / gcd(a,m) n

x $\equiv$ a1(mod m1) ....(i)

x $\equiv$ a2(mod m2) ....(ii)

Here gcd(m1,m2) = 1

=> eqn (i) has solution.

x $\equiv$ a1 (mod m1)

=> x = a1 + m1y ....(*)
If x is also a solution of (ii)
x $\equiv$ a2 (mod m2)
=> a1 + m1y $\equiv$ a2(mod m2)
=> m1y $\equiv$ (a2 -a1) (mod m2)
gcd(m1, m2) = 1
This equation will have solution of the form y = y0 + m2 / gcd(m1, m2) n
=> y = y0 + m2n
From (*)
x = a1 + m1y
= a1 + m1(y0 + m2n)
=> x = (a1 + m1y0) + n. m1m2
=> x = x0 + n.m1m2
=> x *equiv* x0 (mod m1m2)

Chinese Remainder Theorem

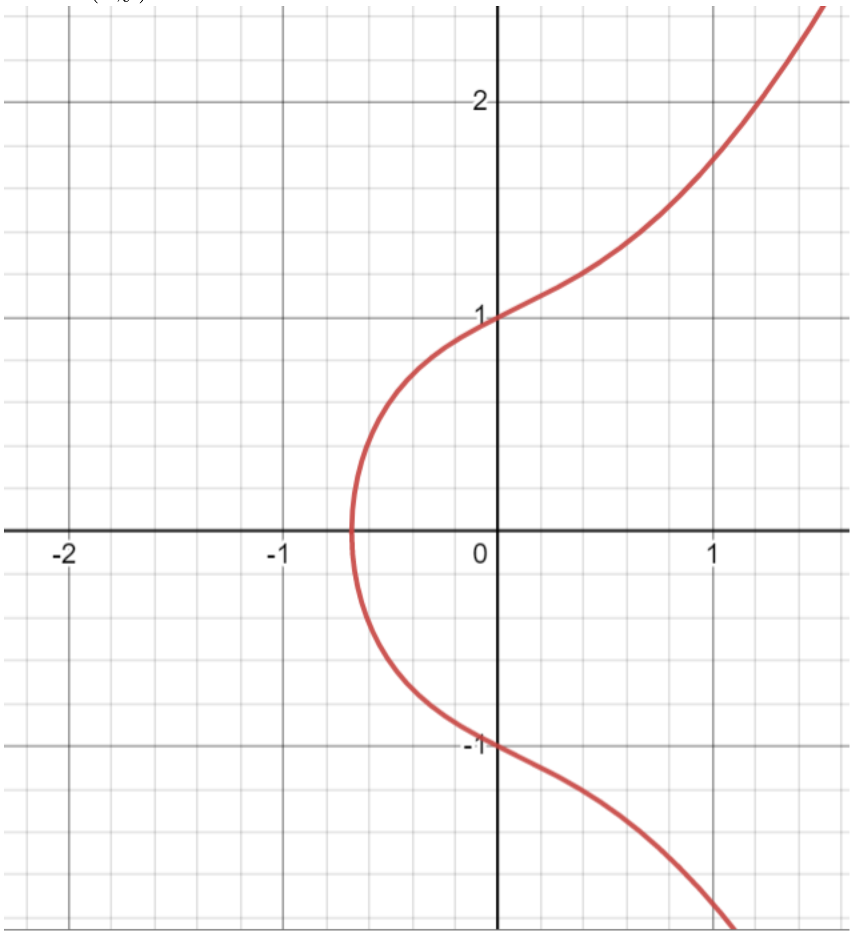
The system,
x $\equiv$ a1 (mod m1)
x $\equiv$ a2 (mod m2)
.
.
.
x $\equiv$ ar (mod mr)
where m1, m2,..., mr are pairwise relatively prime then the above system has a unique solution modulo (m1,m2,...,mr).
=> Let for each j=1,2....,r.
define δj
δj = 1 (mod j) = 0 (mod mi) if i $\neq$ j
x = $\sum_{j=1}^{r} \delta j.aj$ satiesfies all the equations.
x = δ1a1+ δ2a2 + ... + δrar
=> x $\equiv$ aj (mod mj)
M = m1.m2...mr
gcd(M/mj , mj) = 1
As gcd(mj,mi) = 1 if i $\neq$ j
=> M/mj bj $\equiv$ 1 (mod mj)
*delta*j = M/mj bj
*delta*j (mod mi) i $\neq$ j
= M / mj bj (mod mi)
= (m1m2 .... mj-1mj+1 ..... mimi+1...mr)bj
= t.mi (mod mi) (mod mi)
= 0
δj = 1 (mod j) = 0 (mod mi) if i $\neq$ j
x = $\sum_{j=1}^{r} \delta j.aj$
= existence of common solution

Uniqueness

Assume x' is another solution of the above system then x' ≡ x (mod m1...mr)
x' ≡ x (mod m1)
x' ≡ x (mod m2)
.
.
.
x ≡ x (mod mr)
x' ≡ a1 (mod m1)
x ≡ a1 (mod m1)
=> x' ≡ x (mod m1)
=> x' ≡ x (mod m1...mr)
=> x = $\sum_{j=1}^{r} \delta j.aj$ is the unique solutiopn under modulo.

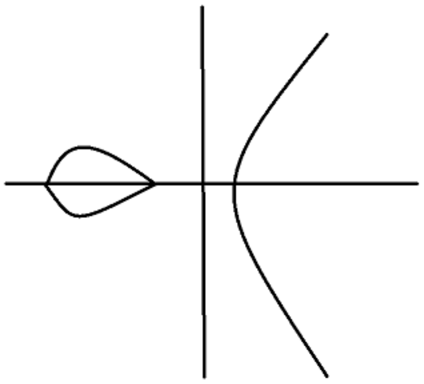
Elliptic Curve

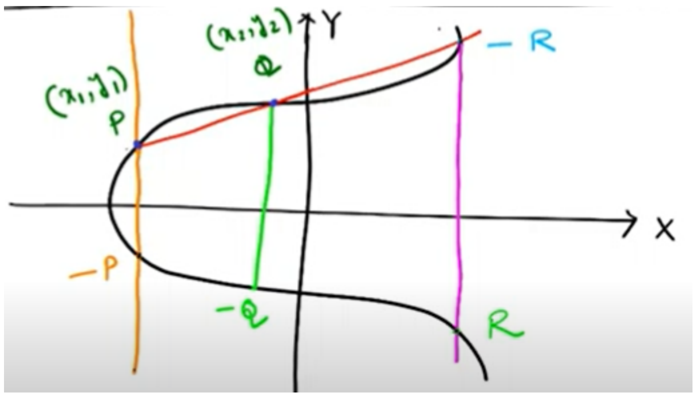
i. a,b ∈ R
ii. $4a^3 + 27b^2 \neq 0$
iii. Equation of the curve:
$y^2 = x^3 + ax + b$
where $(x,y) \in R^2$



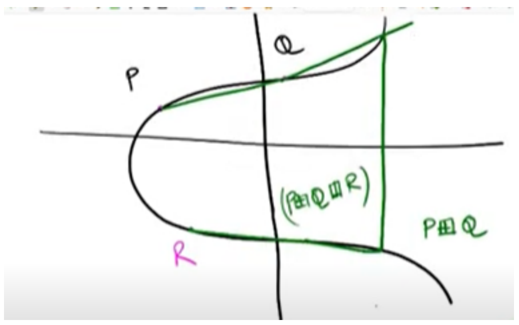
At A, y=0

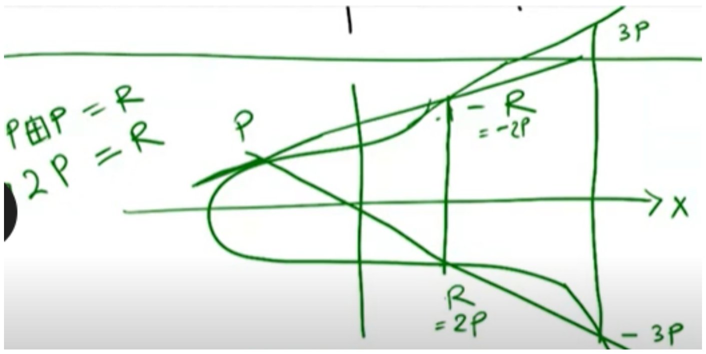
$x^3 + ax + b = 0$ ...(*)
a) Three roots are real
b) One real root, Two complex roots.
Equation (*) will have three dustinct roots and if $4a^3 + 27b^2 \neq 0$
(roots can be real or complex)





1. P ⊞ Q = R
2. O = point at infinity
3. P ⊞ (-P) = O
4. P ⊞ O = p
5. (P ⊞ Q) ⊞ R = P ⊞ (Q ⊞ R)



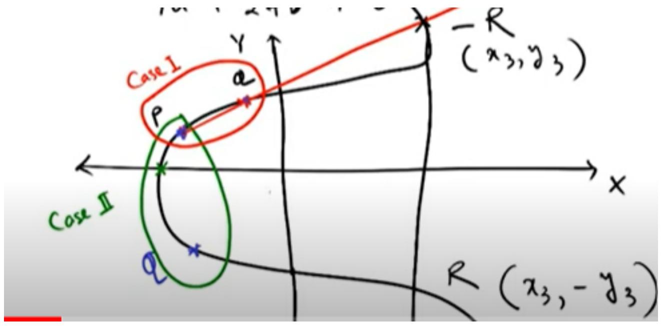


3P = 2P ⊞ P

Elliptic Curve:

$y^2 = x^3 + ax + b$ ...(1)

$4a^3 + 27b^2 \neq 0$



1. P ⊞ Q = R
2. P ⊞ P = 2P
3. (n-1) P ⊞ P = nP

Case I:

P → (x1,y1), Q→ (x2,y2)

y = mx + c ...(a)

m = y2 - y1/ x2 - x1

y1 = mx1 + c

c = y1 - mx1

c = y2 - mx2

Equation of the straight line at (a) will cut the curve (1)

$y^2 = x^3 + ax + b$

$(mx + c)^2 = x^3 + ax + b$

$x^3 - m^2x^2 + (a - 2mc)\, x + (b - c^2) = 0$

(x1,y1), (x2,y2) will satisfy this equation.

If x3 is another solution of the above system.

$x1 + x2 + x3 = m^2$

$x3 = m^2 - x1 - x2$

y3 = y1 + m(x3 -x1)

R → (x3, -y3)

$P \boxplus Q = R = (x3, -y3)$
Case II:
P (x1,y1), Q(x2,y2)
x1 = x2, y1 = -y2
$P \boxplus Q = O$
Case III:
P (x1,y1), Q(x2,y2)
x1 = x2, y1 = y2
y = mx + c
$y^2 = x^3 + ax + c$
$\Rightarrow 2y\ dy/dx = 3x^2 + a$
$\Rightarrow dy/dx = 3x^2 + a /2y = m$
c = y1 -mx1
$y^2 = x^3 + ax + b$
$\Rightarrow (mx + c)^2 = x^3 + ax + b$
$x1 + x2 + x3 = + m^2$
$\Rightarrow x3 = m^2 - x1 - x2$
m = y3 - y1 / x3 - x1
=> y3 = y1 + m(x3 -x1)
R(x3, -y3)
We will be considering the sane curve in Zp * Zp where p is a prime number.
$y^2 = x^3 + ax + b$, $(x,y) \in Zp * Zp$ and $a,b \in Zp$.
$4a^3 + 27b^2 \neq 0 \bmod p$.
Case I:
$x3 = m^2 - x1 - x2 \in Zp$
m = y2 -y1/x2 - x1
$y3 = y1 + m(x3 -x1) \in Zp$
ECDH:
E,P = Public

Alice		Bob
a		b
ap	$\xrightarrow{ap}$	bP
a(bp)	$\xleftarrow{bp}$	b(aP)
abp	abp	bap

Secret key Bob : b
Public key Bob : bP
Security of ECDH relies on the fact that finding x from xP and P is computationally difficult.
This hard problem is known as Discrete log problem on EC.
RSA Signature
Encryption : $c = x^e \bmod n$
Decryption : $x = c^d \bmod n$
Signature : $s = x^d \bmod n$
Verification: $x = s^e \bmod n$
ECDSA:

(E,P) = Public Keys
Secret Key: a

Public Key: ap

Alice   Bob

.   EC, G-Point (Base)

.   on the curve

.   There exists a large

.   prime number n S.T. n.G =0

.   (n-1) G$\boxplus$ G = nG

dA = Secret key

QA = dAG = Public key

m = message

1. e = hash(m)
2. Z = Ln left most bits of e when Ln is the bit length of n.
3. K = randomly from [1, n-1]
4. (x1, y1) = K.G
5. r = x1 mod n if r =0 then go to step 3.
6. S = K^-1 [Z + r.dA] mod n

if S =0 go to step 3

7. Signature (r,s) on message m.

Verification of ECDSA performd by Bob.

1. QA is not equal to 0.
2. QA lies on the curve EC or not.
3. n * QA = n (dA. G) = dA (n. G) = 0
4. U1 = Z. S^-1 mod n

U2 = r.S^-1 mod n

5. (x2, y2) = U1G + U2QA

if (x2,y2) =0 then signature os invalid.

6. if r $\equiv$ x2 mod n then signature os valid otherwise invalid.

c = u1G + u2QA = Bob

= u1G + u2dAG

= (u1+ u2dA) G

= (Z.S^-1 + r.S^-1 dA) G

= (Z + r.dA) s^-1 G

Elgamal Public Kwy Cryptography

1. Select a prime P
2. Consider the group (Zp*, . mod p)

x, y $\in$ Zp*

x. y = (x * y) mod p

Zp* = {1,2,...,p-1}

3. Select a primitive element $\alpha \in$ Zp*

Zp* $\rightarrow$ Cylic group

Zp* = $< \alpha >$

any x $\in$ Zp* then x = α^i

4. Plaintext space = Zp *

Key Space = {(p, α, a, β) | $\beta = \alpha^a$ mod p}

5.K = (p, α, a, β)
Select a random number x $\in$ Zp -1 x is kept secret.
6. Encryption:
ek(m,x) = (y1,y2)
y1 = α^x (mod p)
y2 = m. β^x (mod p)
7. Decryption:
dk(y1,y2) = y2(y1^a)$^{-1}$ mod P
y1^a = (α^x)^a mod P
= β^x mod P
y2(y1^a)$^{-1}$ = (m.β^x. (β^x)$^{-1}$ (mod p)

Discrete Logarithmic Problem:

Given: Finite cyclic group G of order n generator α of G element $\beta \in$ G.
Find: integer x 0 <= x <= n-1. Such that $\alpha^x = \beta$

Baby - Step Giant - Step Algorithm

Idea:

1. Let g be the generator of a finite cyclic group G of order n, and let h be an element of G whose discrete logarithm is to be computed.
2. Choose an integer m such that $m \geq \sqrt{n}$.
3. Compute the m baby steps: $g^0, g^1, g^2, \ldots, g^{m-1}$.
4. Compute the inverse of g: g^{-1}.
5. Compute the m giant steps: $hg^{-1}, hg^{-2}, hg^{-3}, \ldots, hg^{-m+1}$.
6. Sort the baby steps and giant steps into separate lists, and build a hash table of the baby steps.
7. For each giant step hg^{-i} in the list of giant steps, look up the corresponding baby step g^j in the hash table. If such a j exists, then the discrete logarithm of h is $j + im$.
8. If no j is found for any of the giant steps, increase the value of m and repeat from step 3.

Kerberos Version 4

TGS: Ticket Generating Server
C: Client
AS: Authentication Server
v: Verifier.
1. C $\rightarrow$ AS : IDc || ID$_{TGS}$ || TS1
2. AS $\rightarrow$ C: E(SKc, [Sk$_{c,TGS}$ || ID$_{TGS}$ || TS2 || Lifetime2 || Ticket$_{TGS}$]

Ticket$_{TGS}$ = E(SK$_{c,TGS}$, [Sk$_{c,TGS}$ || ID$_c$ || ADc || ID$_{TGS}$ || Lifetime2]
3. C $\rightarrow$ TGS: IDv || Ticket$_{TGS}$ || Autheticatorc
Autheticatorc = E(SK$_{c,TGS}$, [IDc || ADc || TS3])
4. TGS $\rightarrow$ C : E(SK$_{c,TGS}$, [SK$_{c,v}$ || IDv || TS4 || Ticketv])
Ticketv = E(Skv, [SK$_{c,v}$ || ADc || IDv || TS4 || Lifetime4])
5. C $\rightarrow$ V : Ticketv || Authenticatorc
6. v $\rightarrow$ C: E(SK$_{c,v}$, [TS$_5$ +1])

[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2022-2023
Scribed by: Bhargavi Kamble (202051048) — Lecture (Week 13)

SSL:

Connection: A connection is a transport (in the OSI layering model definition) that provides a suitable type of service. For SSL, such connections are peer-to-peer relationships. The connections are transient. Every connection is associated with one session.

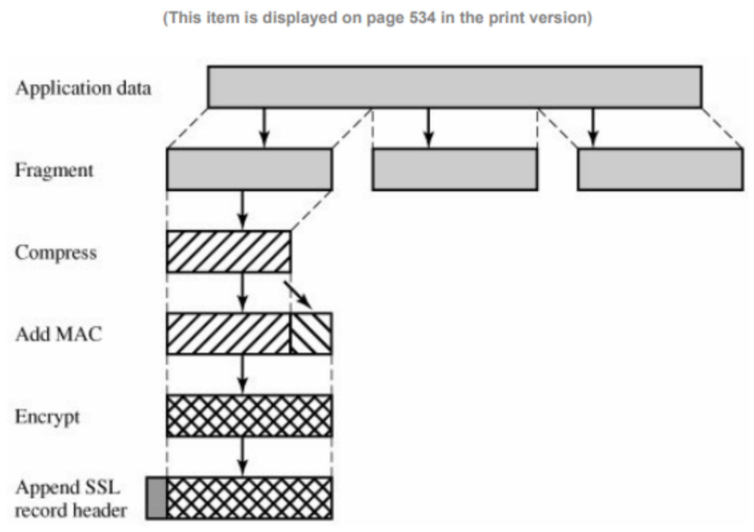
Session: An SSL session is an association between a client and a server. Sessions are created by the Handshake Protocol. Sessions define a set of cryptographic security parameters, which can be shared among multiple connections. Sessions are used to avoid the expensive negotiation of new security parameters for each connection.

A session state is defined by the following parameters (definitions taken from the SSL specification):

Session identifier: An arbitrary byte sequence chosen by the server to identify an active or resumable session state.
Peer certificate: An X509.v3 certificate of the peer. This element of the state may be null.
Compression method: The algorithm used to compress data prior to encryption.
Cipher spec: Specifies the bulk data encryption algorithm (such as null, AES, etc.) and a hash algorithm (such as MD5 or SHA-1) used for MAC calculation. It also defines cryptographic attributes such as the hashSize.
Master secret: 48-byte secret shared between the client and server.
Is resumable: A flag indicating whether the session can be used to initiate new connections.
A connection state is defined by the following parameters:
Server and client random: Byte sequences that are chosen by the server and client for each connection.
Server write MAC secret: The secret key used in MAC operations on data sent by the server.
Client write MAC secret: The secret key used in MAC operations on data sent by the client.
Server write key: The conventional encryption key for data encrypted by the server and decrypted by the client.
Client write key: The conventional encryption key for data encrypted by the client and decrypted by the server.
Initialization vectors: When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL Handshake Protocol. Thereafter the final ciphertext block from each record is preserved for use as the IV with the following record.
Sequence numbers: Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero. Sequence numbers may not exceed $2^{64}1$.

The SSL Record Protocol provides two services for SSL connections:

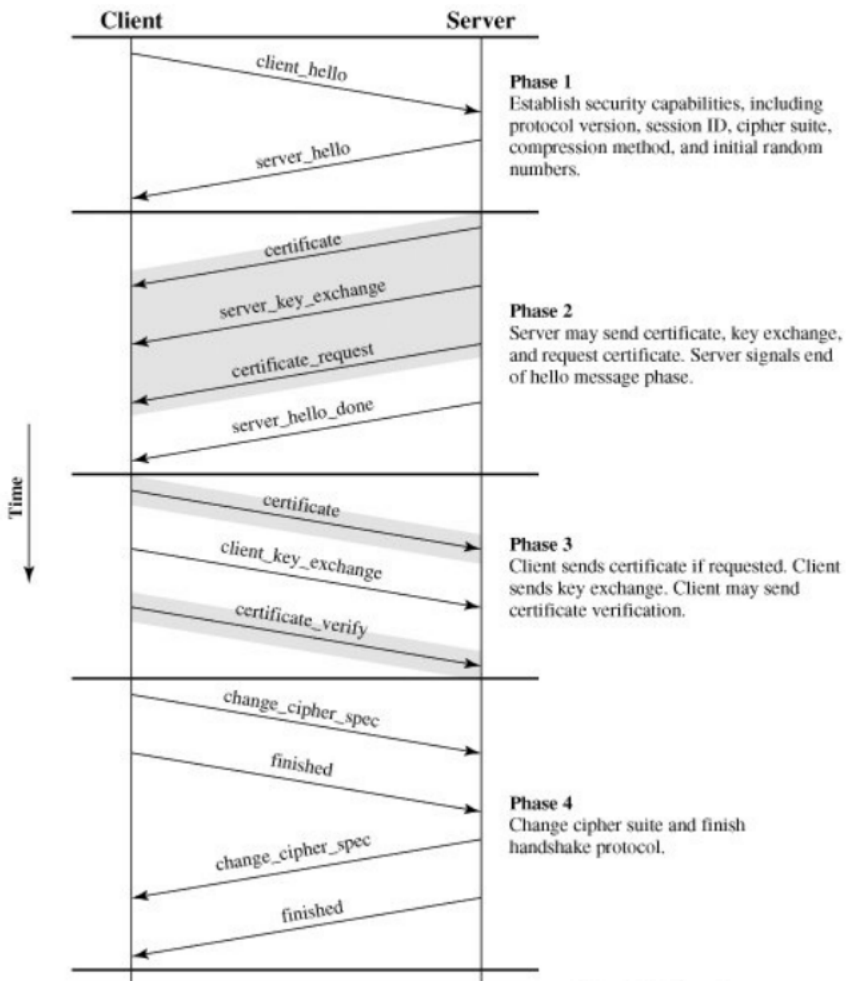
Confidentiality: The Handshake Protocol defines a shared secret key that is used for conventional encryption of SSL payloads.
Message Integrity: The Handshake Protocol also defines a shared secret key that is used to form a message authentication code (MAC).



hash(MAC_write_secret || pad_2 ||
hash(MAC_write_secret || pad_1 || seq_num || SSLCompressed. type ||
SSLCompressed. length || SSLCompressed. fragment))
Encryption algo:
AES, IDEA, RC2-40, DES-40, DES, 3DES, Fortezza, RC4-40, RC4-128.
Header:
Content Type (8 bits), Major Version (8 bits), Minor Version (8 bits), Compressed Length (16 bits)
Change Cipher Spec Protocol:
updates the cipher suite to be used on this connection.
Alert Protocol:
Always fatal:
- unexpected_message
- bad_record_mac
- decompression_failure
- handshake_failure
- illegal_parameter

Potentially non-fatal:
- close_notify
- no_certificate
- bad_certificate
- unsupported_certificate
- certificate_revoked
- certificate_expired
- certificate_unknown

Handshake Protocol:


Client
Server
client_hello
server_hello
Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.
certificate
server_key_exchange
certificate_request
server_hello_done
Phase 2
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.
Time
certificate
client_key_exchange
certificate_verify
Phase 3
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.
change_cipher_spec
finished
change_cipher_spec
finished
Phase 4
Change cipher suite and finish handshake protocol.

client_hello message:
Version, Random, Session ID, CipherSuite, Comperssion Method

After sending the client_hello message, the client waits for the server_hello message, which contains the same parameters as the client_hello message.

key exchange methods are supported:
RSA, Fixed Diffie-Hellman, Ephermeral Diffie-Hellman, Anonymous Diffie-Hellman.
CipherSpec:
CipherAlgorithm, MACAlgorithm, CipherType, IsExportable, HashSize, Key Material, IV Size.

server_key_exchange message:
Anonymous DiffieHellman, Ephemeral DiffieHellman,
RSA key exchange: the client cannot simply send a secret key encrypted with the server's public key. Instead, the server must create a temporary RSA public/private key pair and use the server_key_exchange message to send the public key. The message content includes the two parameters of the temporary RSA public key plus a signature.
As usual, a signature is created by taking the hash of a message and encrypting it with the sender's

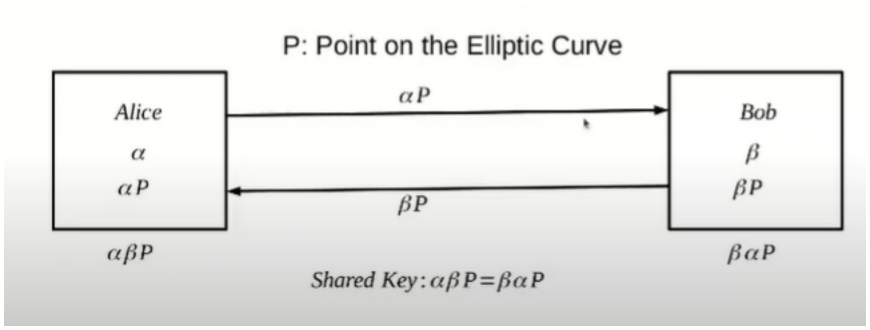
private key. In this case the hash is defined as
hash(ClientHello.random || ServerHello.random || ServerParams)

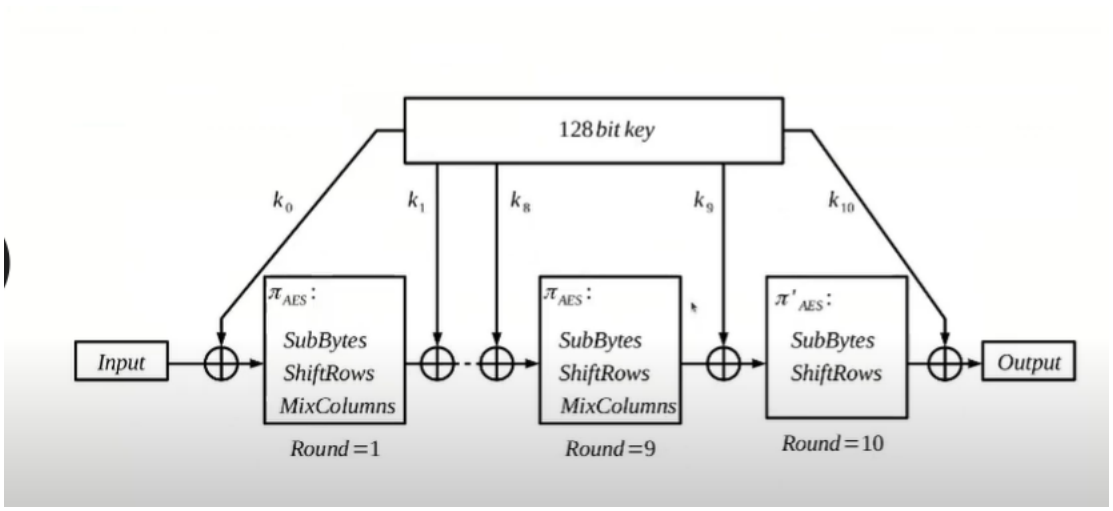
client_key_exchange message
RSA: The client generates a 48-byte pre-master secret and encrypts with the public key from the server's certificate or temporary RSA key from a server_key_exchange message. Its use to compute a master secret is explained later.
Ephemeral or Anonymous DiffieHellman
Fixed DiffieHellman

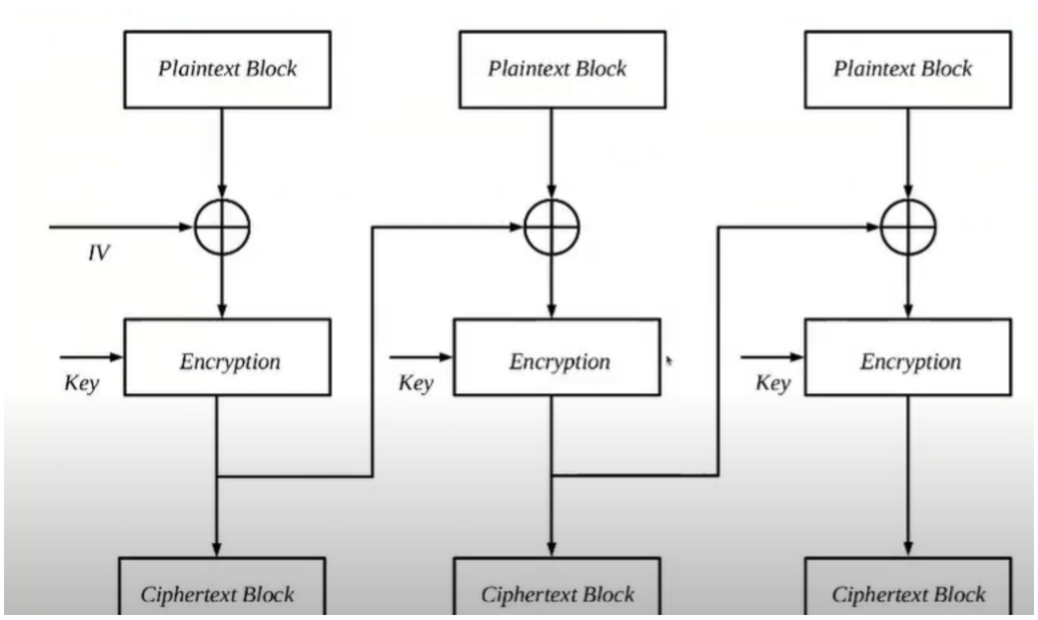
client may send a certificate_verify message
message signs a hash code based on the preceding messages, defined as follows:
CertificateVerify.signature.md5_hash MD5(master_secret || pad_2 || MD5(handshake_messages || master_secret || pad_1));
Certificate.signature.sha_hash
SHA(master_secret || pad_2 || SHA(handshake_messages|| master
_secret || pad_1));
The client sends a change_cipher_spec message and copies the pending CipherSpec into the current CipherSpec. The client then immediately sends the finished message concatenation of two hash values.
MD5(master_secret || pad2 || MD5(handshake_messages ||
Sender || master_secret || pad1))
SHA(master_secret || pad2 || SHA(handshake_messages ||
Sender || master_secret || pad1))
In response to these two messages, the server sends its own change_cipher_spec message, transfers the pending to the current CipherSpec, and sends its finished message. At this point the handshake is complete and the client and server may begin to exchange application layer data.
master_secret = MD5(pre_master_secret || SHA('A' ||
pre_master_secret || ClientHello.random ||
ServerHello.random)) ||
MD5(pre_master_secret || SHA('BB' ||
pre_master_secret || ClientHello.random ||
ServerHello.random)) ||
MD5(pre_master_secret || SHA('CCC' ||
pre_master_secret || ClientHello.random ||
ServerHello.random))
key_block = MD5(master_secret || SHA('A' || master_secret ||
ServerHello.random || ClientHello.random)) ||
MD5(master_secret || SHA('BB' —— master_secret ||
ServerHello.random || ClientHello.random)) ||
MD5(master_secret || SHA('CCC' || master_ secret || ServerHello.random || ClientHello.random)) || . . .

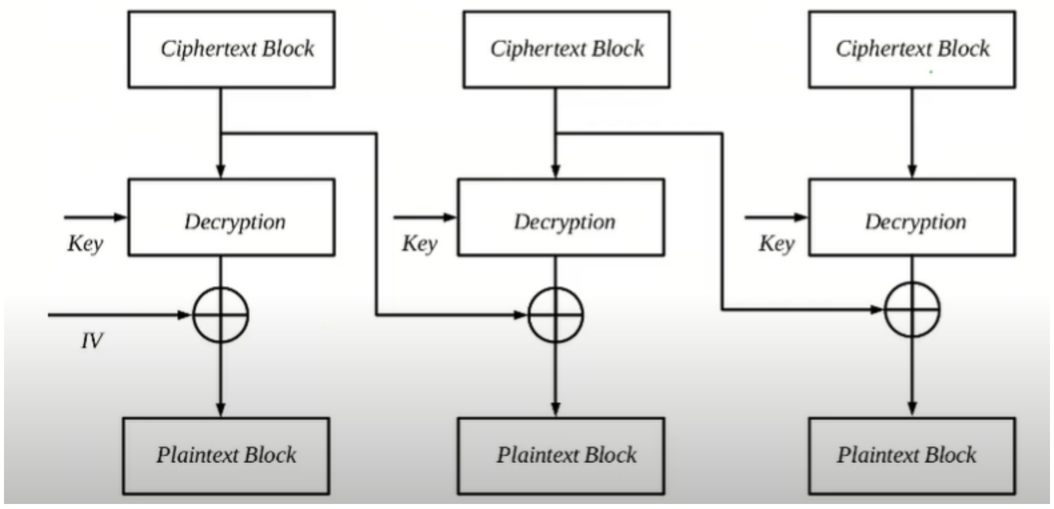
An Overview on Signal Protocol

Suppose alice sents a text to bob saying "Hi bob how are you?" Bob will receive the text as it is however the server will have no idea what the text was. This is called as end-to-end encryption.




128 bit key
k_0
k_1
k_8
k_9
k_{10}
Input
π_{AES}: SubBytes ShiftRows MixColumns
Round = 1
π_{AES}: SubBytes ShiftRows MixColumns
Round = 9
π'_{AES}: SubBytes ShiftRows
Round = 10
Output



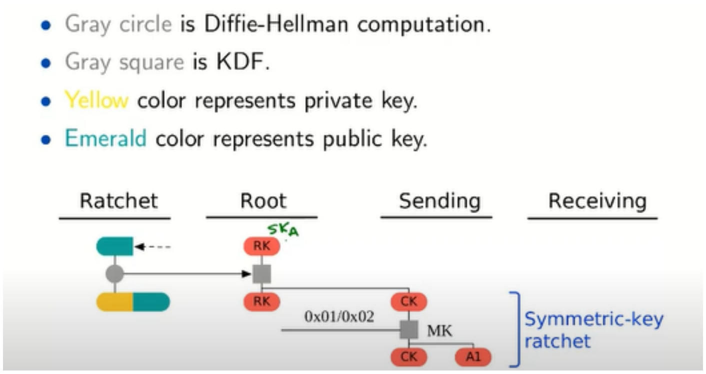


Assosiated data is authenticated but not encrypted.
Schemes are nonce-based (and deterministic)
Sender: C = Enc(K,N,AD,M)
Sender $\xrightarrow{K,N,AD,M}$ Receiver
Receiver: M: Dec(K,N.AD,C)
Generation of Shared Secret Key
Ensuring End-To-End Encryption
Secure transfer of data from one user to another via Server.
Ensuring Forward Security.
This protocol is used by several apps such as Signal, WhatsApp, Skype,etc.
It is an open source protocol.
Main Cryptographic Modules:
X3DH
ECDSA
Double Ratchlet
Curve X25519 or X448 is used in ECDH and ECDSA
Alice $\leftrightarrow Server \leftrightarrow Bob$
Alice registers himself to the server.
Alice $\xrightarrow{Registration}$ Server
Alice generates the following Packet$_A$.
Packet$_A$: Identity Key: IK$_A$
Upload Once to the Server - Long term
Signed prekey: SPK$_A$
Uploaded weekly or once in a month.
Prekey signature: Sig(IK$_A$, Encode(SPK$_A$))
Uploaded weeekly or once in a month
A set of one-time prekeys: (OPK$^1{}_A$, OPK$^2{}_A$,OPK$^3{}_A$,...)
(Optional) Uploaded when Server requesrts for it.
Bob generates the following Packet$_B$:
Packet$_B$: Indetity key: IK$_B$

Signed prekey: SPK$_B$
Prekey Signature: Sig(IK$_B$, Encode(SPK$_B$))
A set of one-time prekeys: (OPK$^1{}_A$, OPK$^2{}_A$,OPK$^3{}_A$,...)
Server stores Packet$_B$:
Packet$_B$: Indetity key: IK$_B$
Signed prekey: SPK$_B$
Prekey Signature: Sig(IK$_B$, Encode(SPK$_B$))
A set of one-time prekeys: (OPK$^1{}_A$, OPK$^2{}_A$,OPK$^3{}_A$,...)
1. Alice generates an ephemeral key EK$_A$
2. DH1 = DH(IK$_A$, SPK$_B$),
3. DH2 = DH(EK$_A$, IK$_B$),
4. DH3 = DH(EK$_A$, SPK$_B$),
5. If key bundle contains one-time prekey,
a. DH4 = DH(EK$_A$, OPK$_B$),
b. SK$_A$ = KDF(DH$_1$ || DH$_2$ || DH$_3$ || DH$_4$)
6. Else SK$_A$ = KDF(DH$_1$ || DH$_2$ || DH$_3$).

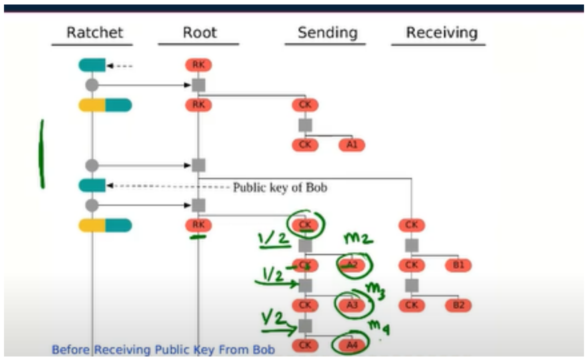
1. Alice computes associated data.
AD: Encode(IK$_A$) || Encode (IK$_B$)
2. Alice encrypts an initial message by using AD, SK$_A$ and generates corresponding initial ciphertext.
Alice $\xrightarrow{IC,IKA,BP}$ Server
Server $\xrightarrow{IC,IKA,BP}$ Bob

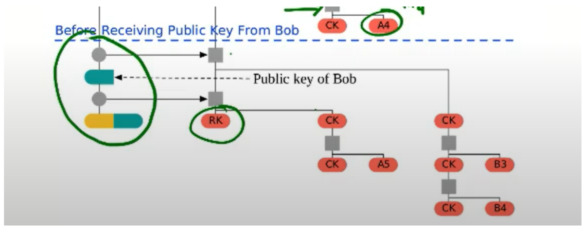
1. Bob recovers IK$_A$, EK$_A$ first.
2. Bob performs DH and KDF by using his secret keys.
3. Due to properties of DH and KDF SK$_B$ = SK$_A$
4. Constructs AD = Encode(IK$_A$) || Encode(IK$_B$).
5. Decrypts the initial ciphertext using SK$_B$, AD recover the original message.

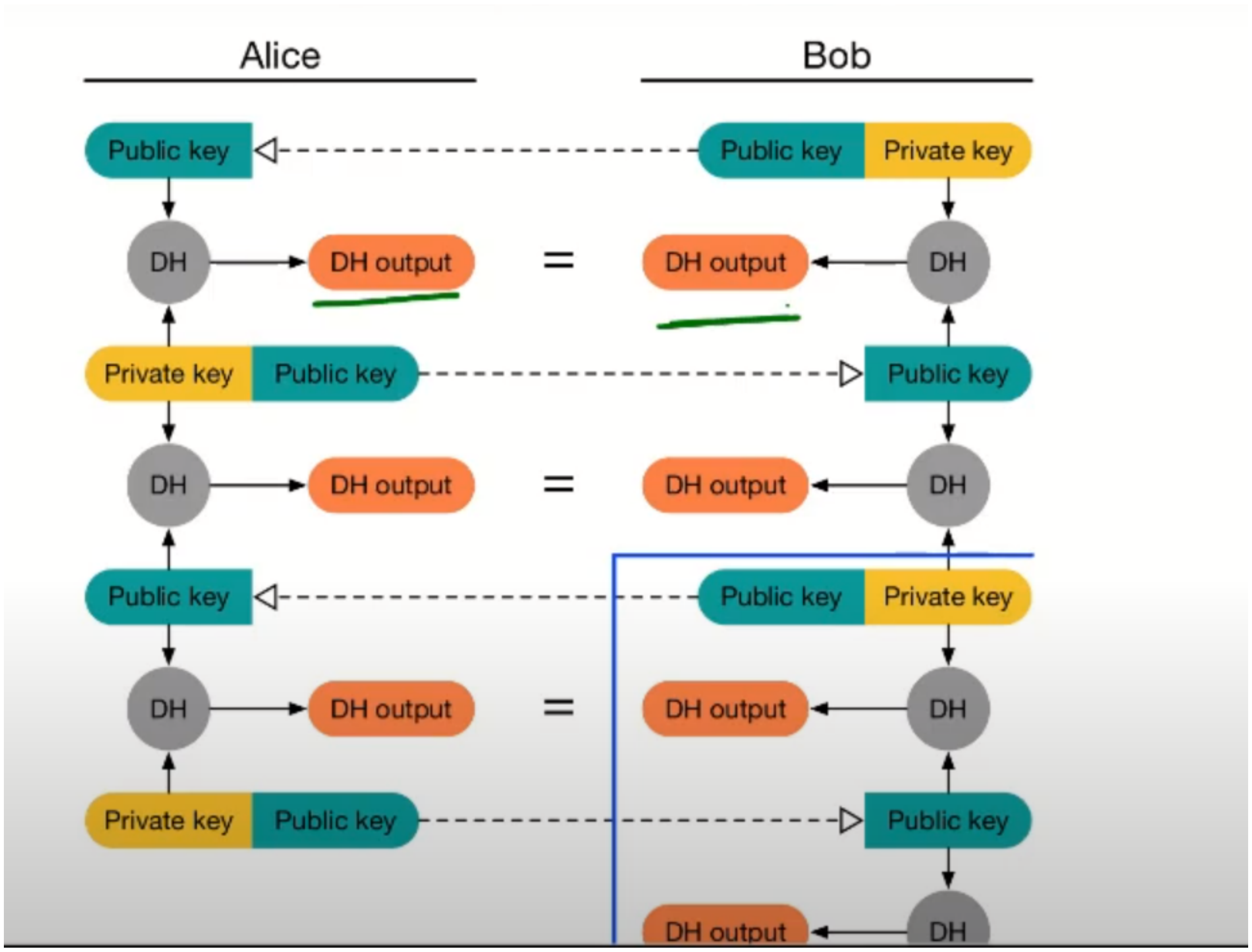


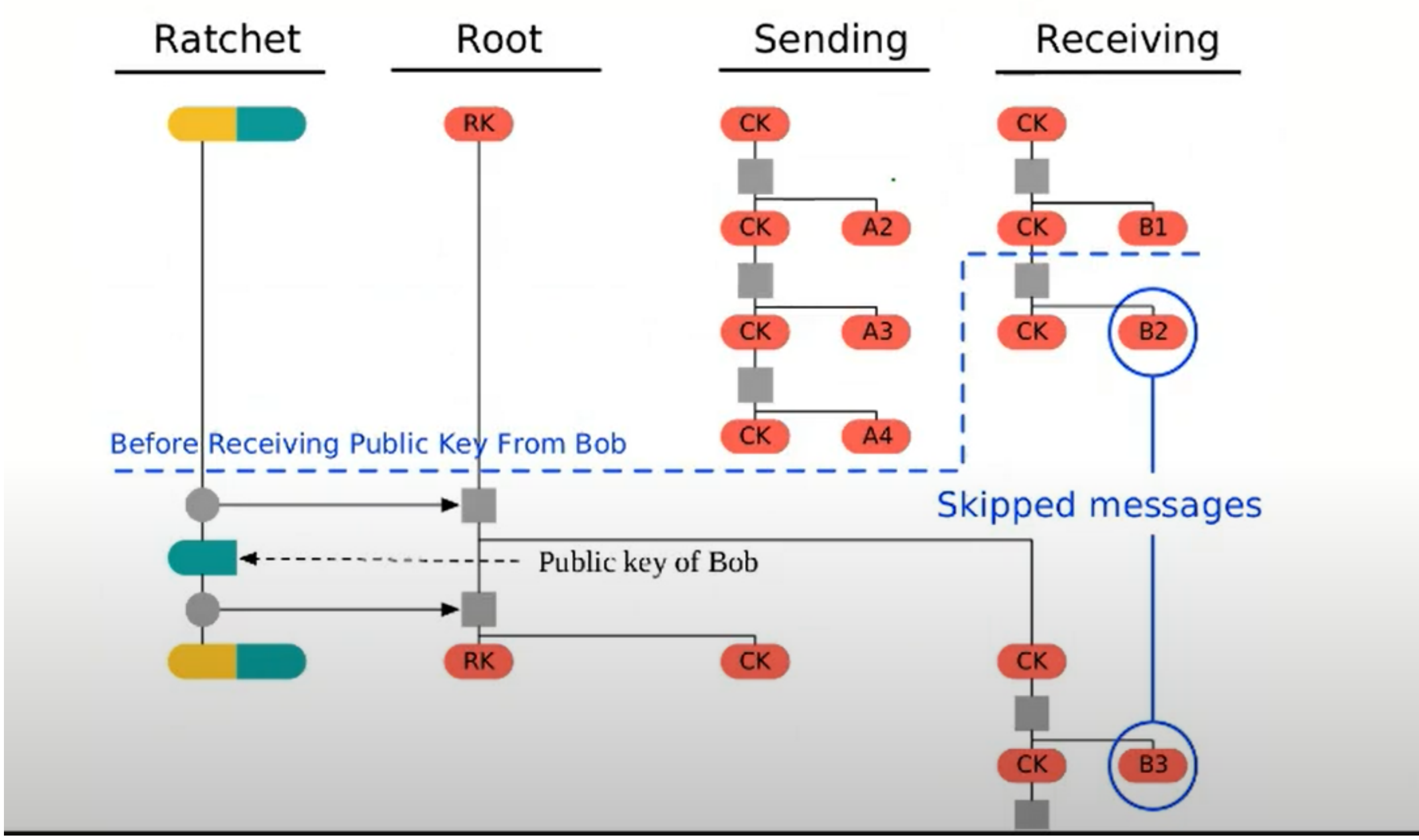
- Gray circle is Diffie-Hellman computation.
- Gray square is KDF.
- Yellow color represents private key.
- Emerald color represents public key.

Gray circle is Diffie-Hellman computation.
Gray square is KDF.
Yellow color represents private key.
Emerald key represents public key.
MK: Message Key
CK: Chain Key


Ratchet
Root
Sending
Receiving
RK
CK
A1
Public key of Bob
A2
A3
A4
B1
B2
m2
m3
m4
Before Receiving Public Key From Bob


Before Receiving Public Key From Bob
CK
A4
Public key of Bob
RK
A5
B3
B4


Alice
Bob
Public key
Private key
DH
DH output
=



Zero Knowledge proof using DLP

Both agreed on multipicative group Zp*. Prover wants to prove the knowledge of x satisfying y = g^x without revealing x.

Prover | Verifier

Computes y = g^x mod p $\xrightarrow{y}$ g^x =y

Select r randomly from [0, p-1]

Compute m = g^x mod p $\xrightarrow{m}$ g^r =m

. $\xleftarrow{e}$ Choose e randomly from [0,p-1]

Compute s = ex + r $\xrightarrow{s}$ Checks:$g^s \overset{?}{=} y^e m$

$g^s = g^{ex+r} = g^{ex}.g^r = y^e$