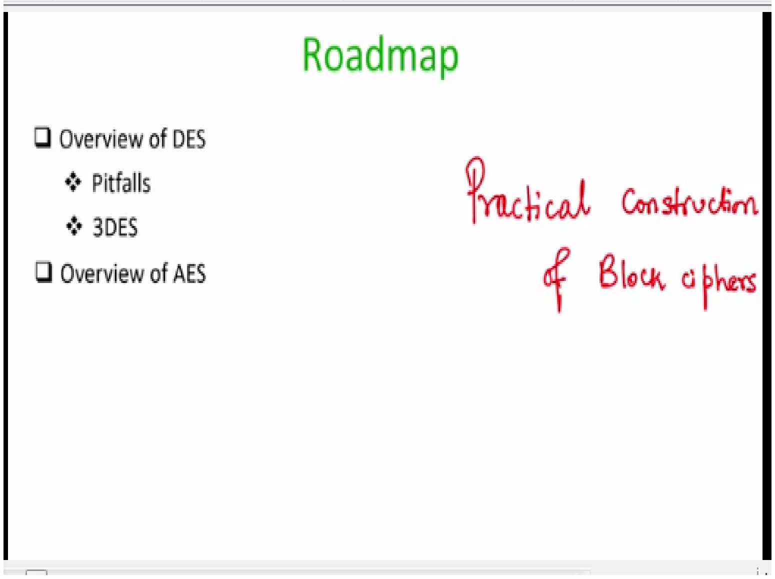
Foundations of Cryptography

Prof. Dr. Ashish Choudhury
(Former) Infosys Foundation Career Development Chair Professor
Indian Institute of Technology-Bengaluru

Lecture-20
Practical Constructions of Block ciphers Part II

Hello everyone, welcome to lecture 19. In this lecture we will continue our discussion regarding the practical constructions of block ciphers. The road map of this lecture is as follows.

(Refer Slide Time: 00:46)



Roadmap

- ❑ Overview of DES
  - ❖ Pitfalls
  - ❖ 3DES
- ❑ Overview of AES

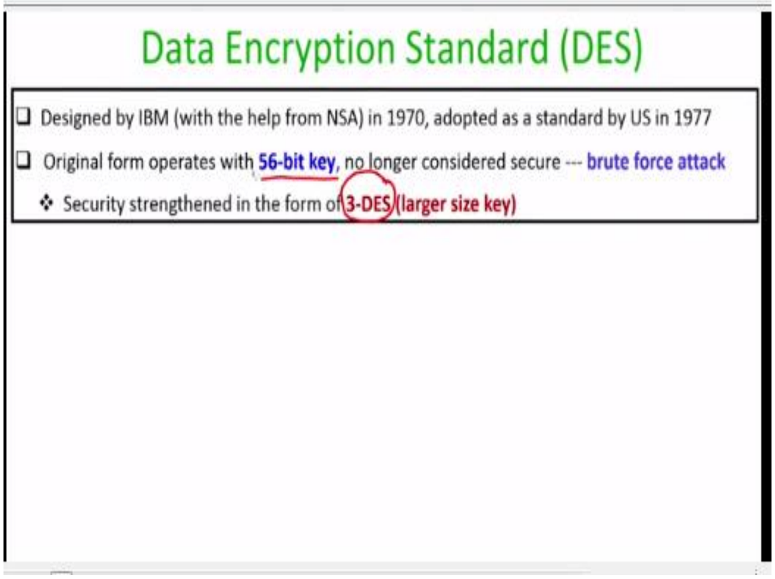
Practical Construction of Block ciphers

We will see that how using SPN and Feistel network, we can design a highly popular block cipher which we call us DES. We will also discuss some of the pitfalls of DES and how those pitfalls motivated the construction of another popular block cipher called AES and we will see a very high level overview of AES. Before I further continue, I would like to stress here that why during all this discussion, I am continuously using the term practical constructions of block ciphers. Because this is the way we instantiate our pseudo-random functions, pseudo-random permutations and strong pseudo-random permutations in reality. However, it turns out that we do not have any provable security guarantees for the DES, triple DES, AES. It is only that ever since their discovery, no practical flaw has been identified and no security breach has been identified.

And that is why we believe that DES, triple DES, AES behave like pseudo-random permutation and strong pseudo-random permutations. Whereas the constructions of pseudo-random permutations and pseudo random-functions based on one way functions that we had seen in some of our earlier lectures, they are provably secure, because we have a mathematical security proof that indeed they satisfy the definitions of pseudo-random function, pseudo-random permutations. But the reason I call those constructions as theoretical constructions, is that even though they are efficient, there running time is polynomial in the underlying security parameters. The underlying polynomials are so huge that we cannot afford them to use in practice. Whereas, the practical constructions that we are going to discuss in this lecture based on SPN, Feistel network, they are highly efficient.

But the downside is that you do not have any mathematical security guarantee, you just have heuristic security guarantee, namely ever since they are discovered, no security breaches have been identified. And that is why we can safely use them to instantiate any cryptography primitive where you want to instantiate pseudo-random functions, pseudo-random permutations and so on. So let us start our discussion here.

(Refer Slide Time: 03:08)

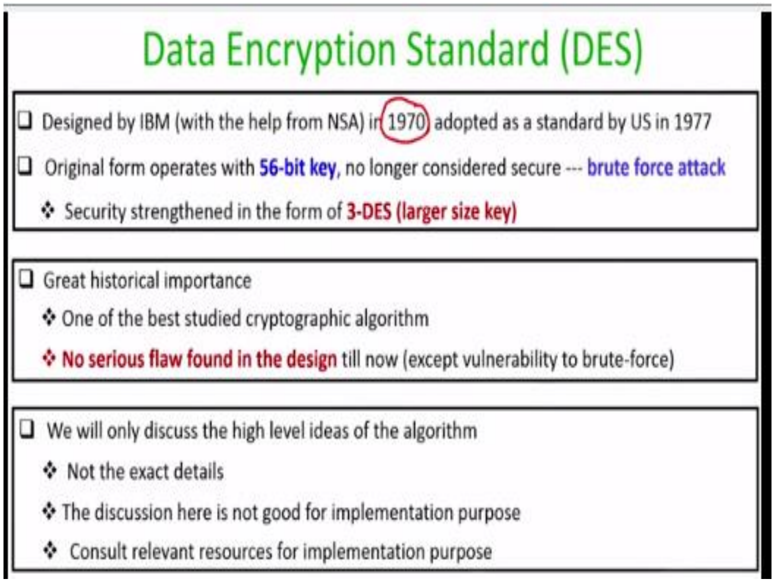


Data Encryption Standard (DES)

- ❑ Designed by IBM (with the help from NSA) in 1970, adopted as a standard by US in 1977
- ❑ Original form operates with 56-bit key, no longer considered secure --- brute force attack
  - ❖ Security strengthened in the form of 3-DES (larger size key)

So we start the discussion with respect to Data Encryption Standard or DES. It was designed by the IBM, with the help of NSA in 1970, adopted as a standard by the United States in 1977. The original form of DES operates with a key of size 56 bits and that is why it is no longer considered to be secure. Because the original form of the DES is susceptible to a brute force attack, namely if an adversary is given an (x, y) pair or several (x, y) pairs, where y is the value of DES with respect to an unknown 56 bit key and input x, where the x input is known to the adversary. Then just by doing computations of order 2^{56}, adversary can recover back the key. So that is why the security of the original proposal of the DES was strengthened and that is how we obtained the new construction, which we call as to 3-DES, which we will also discuss in this lecture and it has a larger key size.

(Refer Slide Time: 04:13)



Data Encryption Standard (DES)

- ❑ Designed by IBM (with the help from NSA) in 1970 adopted as a standard by US in 1977
- ❑ Original form operates with **56-bit key**, no longer considered secure --- **brute force attack**
  - ❖ Security strengthened in the form of **3-DES (larger size key)**

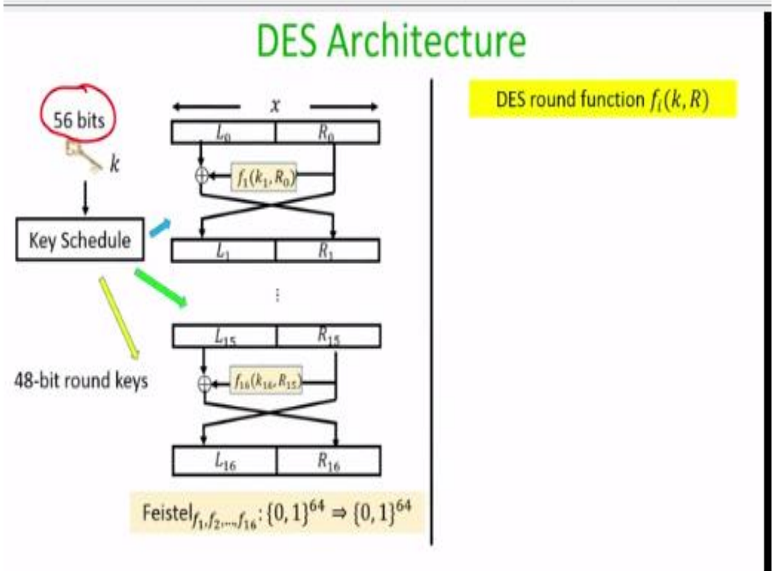
- ❑ Great historical importance
  - ❖ One of the best studied cryptographic algorithm
  - ❖ **No serious flaw found in the design** till now (except vulnerability to brute-force)

- ❑ We will only discuss the high level ideas of the algorithm
  - ❖ Not the exact details
  - ❖ The discussion here is not good for implementation purpose
  - ❖ Consult relevant resources for implementation purpose

DES has got great historical importance because it is one of the best studied cryptographic algorithms. And even though the original form of DES is susceptible to brute force attack, it is vulnerable to the brute force attack, it does not mean that it has any serious design flaw. Namely ever since 1970, no security flaw has been reported in the architecture or the design of the DES. The only complaint is that it operates with a small size key, that is all.

And in this lecture, we are going to discuss only the high level ideas of the algorithm. These are not the exact details and the discussion that we are going to have in this lecture is not good for implementation purpose. If you are interested in implementing DES, you should consult some of the relevant sources which are given in the textbook by Katz-Lindell.

(Refer Slide Time: 05:12)

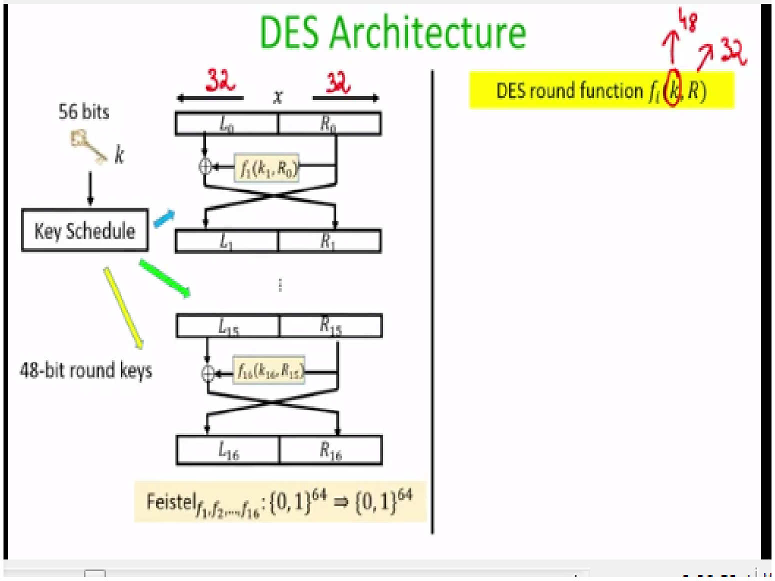


DES Architecture

DES round function $f_i(k, R)$

So the DES architecture is as follows. Basically it is a 16-round Feistel network, mapping 64 bit input to 64 bit output. And the key for the Feistel network is of 56 bits. And there will be a publicly known key scheduling algorithm based on which we will be determining 48 bit round key for each of the individual iterations. So each of these individual iterations is going to use a subset of 48 bits from the master key of size 56 bits. And scheduling of which 48-bit subset of the master key I am going to use for each iteration will be publicly known.

(Refer Slide Time: 06:01)

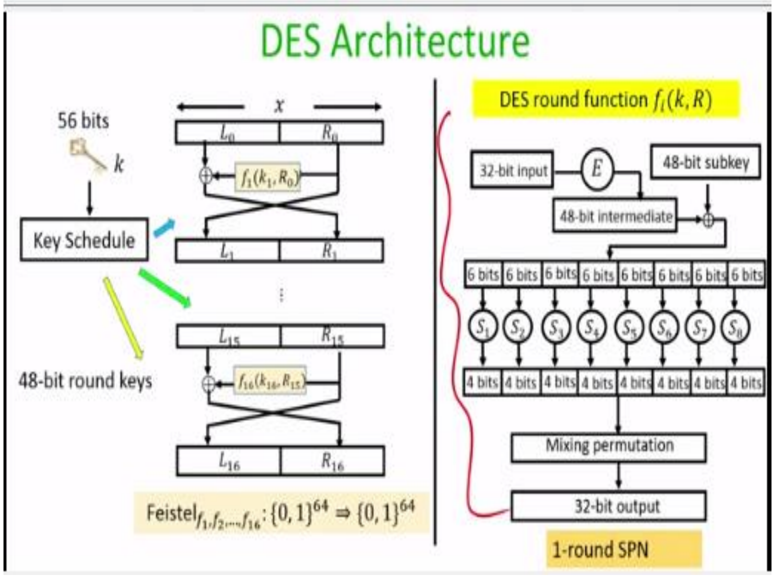


DES Architecture

DES round function $f_i(k, R)$

Now let us focus on the individual round functions. So remember, as per the description of the Feistel network, the round functions could be any arbitrary round function, it need not be invertible. So if I focus on the i^{th} round function, which takes the key of size 48 bits, and a right-half of the current x-input of 32 bits. Why 32 bits? Because remember as per the architecture of the Feistel network, each current x-input or each intermediate output, will be parsed as a collection of left part and a right part, where the left part will be of 32 bits and the right part will be 32 bits.

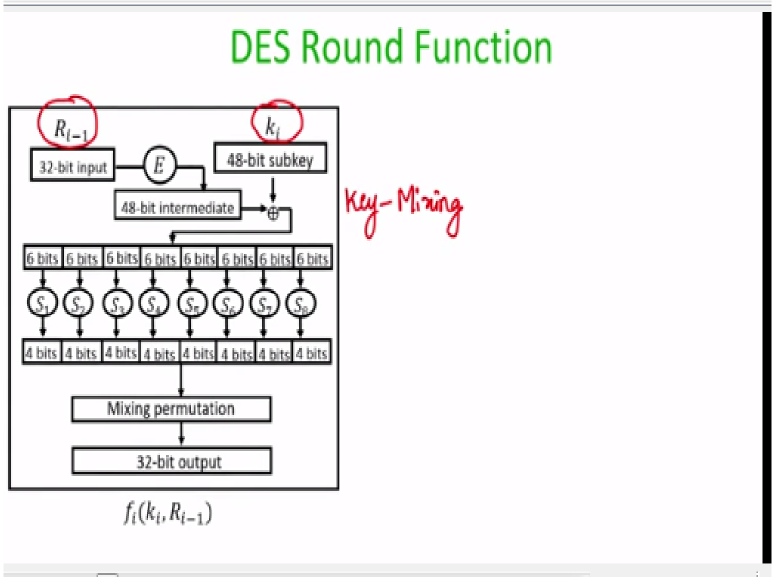
(Refer Slide Time: 06:40)



DES Architecture

So that is why my individual round function will be a keyed function, with a 48-bit key and a 32-bit input. And it turns out that the round function is nothing, but a 1-round is SPN. And that is why you can interpret DES to be a kind of keyed permutation obtained by composing both 1-round SPN and r-round Feistel network, where $r = 16$. So in some sense, we are actually trying to get best of the both primitives here. So for the DES round function, you have a key-mixing step, you have a substitution step and you have a mixing step.

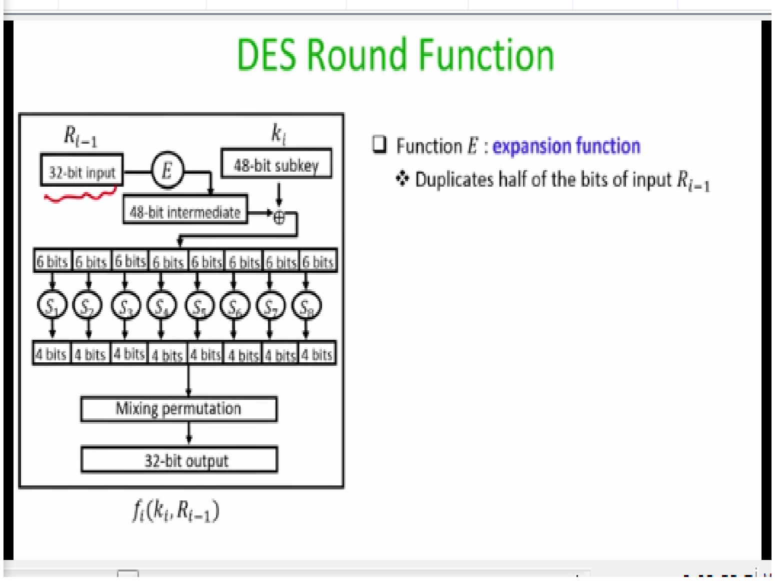
(Refer Slide Time: 07:23)



DES Round Function

So now let us go a little bit deeper into the description of the DES round function. So imagine, this is the i^{th} round function. It will take a sub-key k_i, determined by a publicly known key-scheduling algorithm, of size 48 bits. And it will take a block input, namely the R-part with respect to the Feistel network, whose size is 32 bits. So in SPN, the first step is the key-mixing step and in the key-mixing step, we actually perform the XOR of the sub-key with the block input. But if you see, the block input is of size 32 bits and the key size is 48 bits. So we have an incompatibility to perform the XOR operation.

(Refer Slide Time: 08:10)



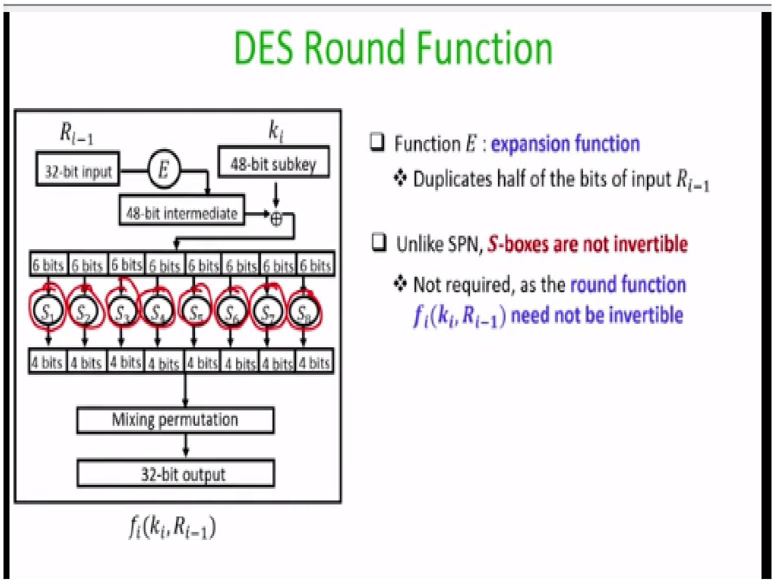
DES Round Function

- Function E : expansion function
  - Duplicates half of the bits of input R_{i-1}

So that is why we perform an expansion function here, which will be publicly known. And what exactly the expansion function does is, it basically duplicates half of the bits of the block input.

Namely, for the 32-bit R input, some of the 16 bits of the R-input, they are duplicated to obtain a 48-bit input, which is now used to do the key-mixing step here.

(Refer Slide Time: 08:34)



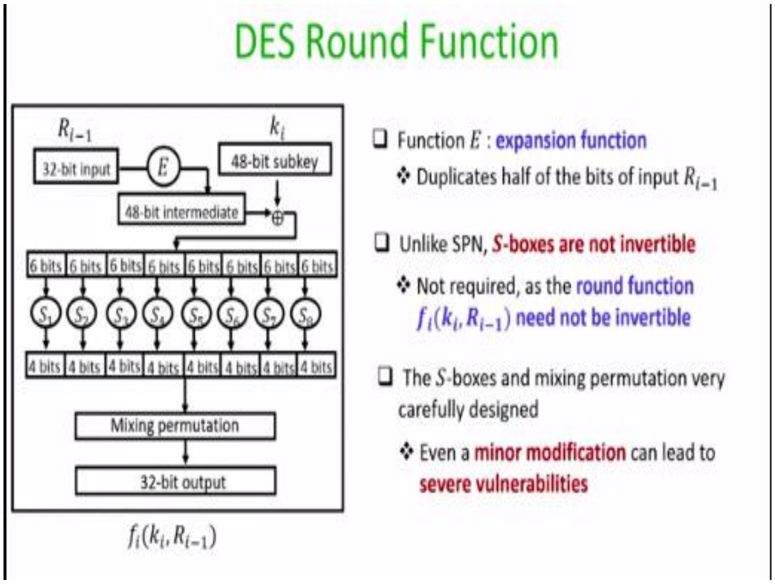
DES Round Function

- ❑ Function E : expansion function
  - ❖ Duplicates half of the bits of input R_{i-1}
- ❑ Unlike SPN, S-boxes are not invertible
  - ❖ Not required, as the round function $f_i(k_i, R_{i-1})$ need not be invertible

Now once we do the key-mixing, we parse the intermediate output as a collection of 8 pieces of 6 bits each. And each of these individual 6 bits go through an S box. Now the interesting aspect here is that unlike the SPN, where we require each of the S box to be a permutation or to be a 1-to-1 onto mapping, the S boxes here, they are not invertible. Because the inputs of the S boxes are 6 bits whereas the outputs of the S boxes are 4 bits.

But it suffice to have S boxes like this because remember, this 1 round SPN is going to serve as the round function for a Feistel network. And for the Feistel network, we do not require the round functions to be invertible. So even though these S boxes are not invertible, it is fine. So once we obtain the output of the key-mixing step, we parse it as 8 pieces of 6 bits each, make each of the individual pieces of 6 bits go through the respective S boxes.

(Refer Slide Time: 09:50)



DES Round Function

❑ Function E : expansion function

❖ Duplicates half of the bits of input R_{i-1}

❑ Unlike SPN, S-boxes are not invertible

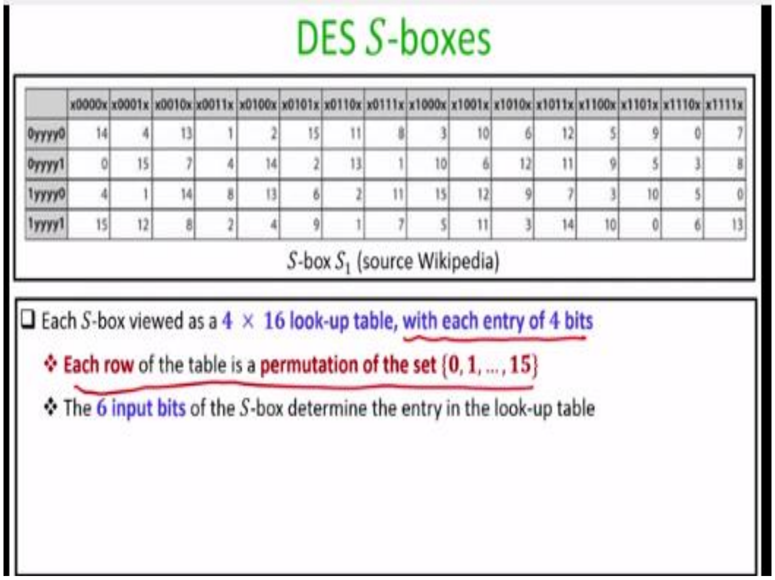
❖ Not required, as the round function $f_i(k_i, R_{i-1})$ need not be invertible

❑ The S-boxes and mixing permutation very carefully designed

❖ Even a minor modification can lead to severe vulnerabilities

And obtain a 32-bit output and then we apply the mixing-permutation, which will be publicly known to obtain the output of the i^{th} round function. So that is the internal description of your i^{th} round function of the DES. And the S boxes and the mixing permutations that are used in each of the round functions of the Feistel network inside DES, they have very special properties. And those special properties we are going to discuss very soon. What I want to stress here is that the S boxes and the mixing permutation that are used here, they are so carefully designed, that even a minor modification can lead to severe security vulnerabilities.

(Refer Slide Time: 10:37)



DES S-boxes

	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0yyyy1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
1yyyy0	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
1yyyy1	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S-box S_1 (source Wikipedia)

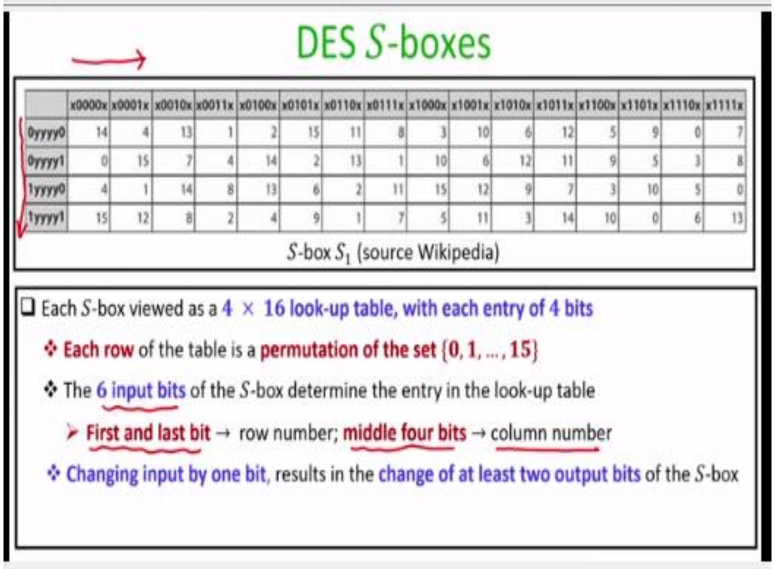
❑ Each S-box viewed as a 4×16 look-up table, with each entry of 4 bits

❖ Each row of the table is a permutation of the set $\{0, 1, \ldots, 15\}$

❖ The 6 input bits of the S-box determine the entry in the look-up table

Now let us see how exactly these S boxes in the context of DES look like. So, for your reference I am taking the first S box and I am taking the description on this first S box from Wikipedia. So, each S box you can imagine here as a 4 × 16 lookup table and each entry in the table consist of a 4-bit string. Because remember the output of an S box is a 4-bit output and each row in the table is a permutation of the set 0 to 15. So you can see here, in each of the 4 rows, you have the value of 0 to 15, occurring exactly once, but arranged in some arbitrarily looking order. And since each of the strings 0 to 15, can be represented by 4 bits, that is why I am saying that each entry in this table basically consist of 4 bits. Now how exactly we are going to access this lookup table? So remember, the input for S box is a 6-bit input.

(Refer Slide Time: 11:35)



DES S-boxes

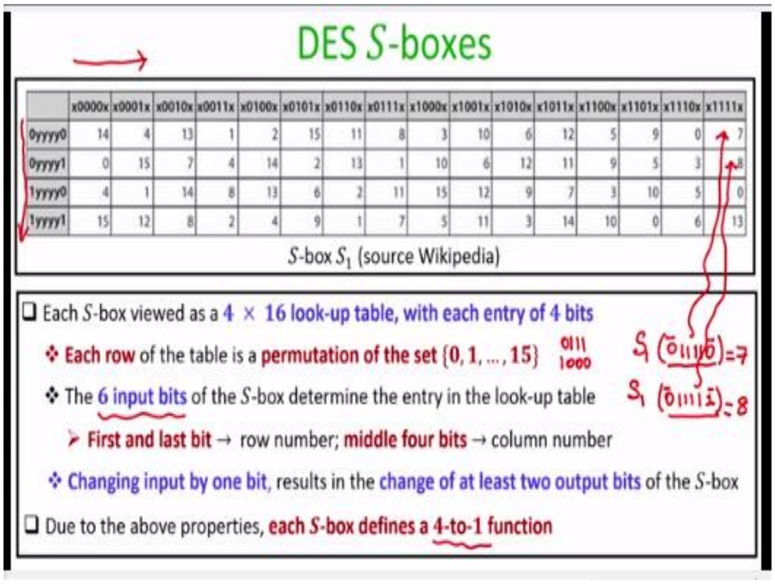
	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0yyyy1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
1yyyy0	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
1yyyy1	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S-box S_1 (source Wikipedia)

- ❑ Each S-box viewed as a 4 × 16 look-up table, with each entry of 4 bits
  - ❖ Each row of the table is a permutation of the set $\{0, 1, \ldots, 15\}$
  - ❖ The 6 input bits of the S-box determine the entry in the look-up table
    - ➤ First and last bit → row number; middle four bits → column number
  - ❖ Changing input by one bit, results in the change of at least two output bits of the S-box

So the input 6 bits of the S box, determine the entry that we are going to look for in this lookup table and how exactly we parse the 6 input bits. So the first and the last bit of the inputs of the S box, they determine your row number. So that is why you can see along the rows, I am just focusing on the first bit and the last bit, the middle 4 bits, I am not focusing. The middle 4 bits, they are used to access a particular column number. So that is why along the columns, it is the middle 4 bits which are highlighted and that is why we have 16 possible columns. Now, the way the numbers 0 to 15 are arranged in this S box is to ensure that if you change the input of the S box by 1 bit then it should result in the change of at least 2 output bits of the S box.

(Refer Slide Time: 12:34)



DES S-boxes

	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0yyyy1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
1yyyy0	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
1yyyy1	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

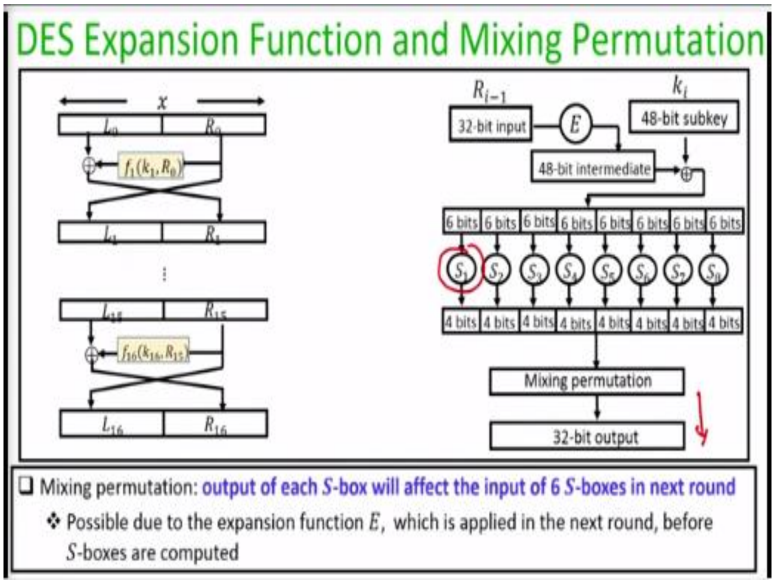
S-box S_1 (source Wikipedia)

- □ Each S-box viewed as a 4×16 look-up table, with each entry of 4 bits
  - ❖ Each row of the table is a permutation of the set $\{0, 1, \ldots, 15\}$ 0111 1000 $S_1(\bar{0}1111\bar{0}) = 7$
  - ❖ The 6 input bits of the S-box determine the entry in the look-up table $S_1(\bar{0}1111\bar{1}) = 8$
    - ➤ First and last bit → row number; middle four bits → column number
  - ❖ Changing input by one bit, results in the change of at least two output bits of the S-box
- □ Due to the above properties, each S-box defines a 4-to-1 function

So let us see whether it is ensured or not. So let us first try to see what exactly will be the value of the first S box on the input, say 011110. So since 0 and 0 are used for row number that means we want to have the first row and then we have to go for column number 1111. So namely, we come to the last column in the first row, which has the value 7, which in Binary is 0111. And now let us see what will be the value of this first S box on an input, say 011111, which differs from the input 011110 in exactly one bit. So the first bit and the last bit are used for accessing the row that means we have to go to the second row. And then we have to go to the column number accessed by the middle 4 bits, which 1111. So that means this will go to the last entry in the second row, which is 8, which in the Binary is 1000. And this output is different from the previous output 0111 in at least 2 bit positions.

So now it is easy to see that due to the above properties, each S box defines a 4-to-1 function. And why it defines a 4-to-1 function is because you have inputs of 6 bits and you have an output of 4 bits. So that means there will be 4 inputs mapping to the same output.

(Refer Slide Time: 15:06)



DES Expansion Function and Mixing Permutation

❑ Mixing permutation: output of each S-box will affect the input of 6 S-boxes in next round

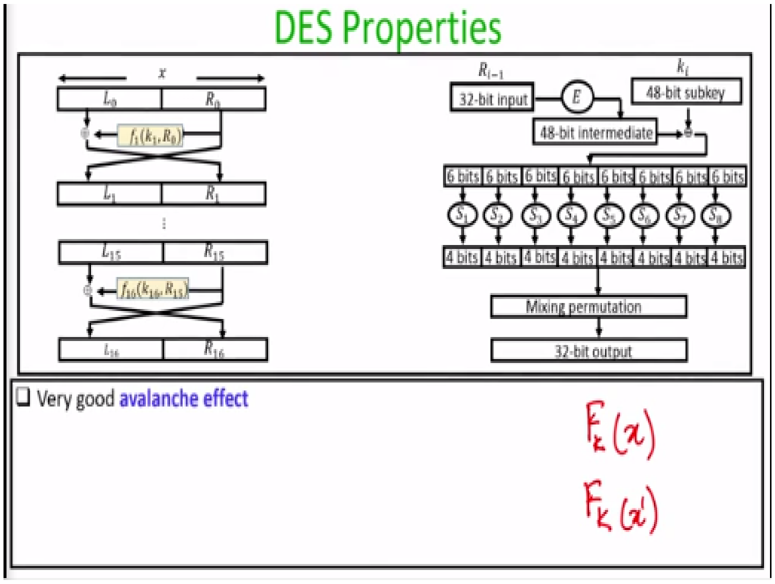
- ❖ Possible due to the expansion function E, which is applied in the next round, before S-boxes are computed

So, the DES expansion function and the mixing permutation, they also have some important properties here. So we already discuss the properties that we required from the S box, basically they ensure that if there is a change of 1 bit of input then the output differs in at least 2 bit positions. Now the expansion function and the mixing permutation they also have some important properties.

So, the mixing permutation have the property that the output of each S box will affect the input of 6 S boxes in the next iteration. That means if I consider the first S box here, you have 4 output bits coming from here. And if I do a mixing permutation, the mixing permutation is designed in such a way that the output of the mixing permutation followed by the expansion function which I am going to apply in the next iteration ensures that there will be 6 S boxes whose inputs will be differing here.

So, the expansion function, namely which 16 bits of the input are replicated, and the mixing permutations which determine the shuffling operation, both of them ensure this important property, namely the output of each S box affect the input of 6 S boxes in the next iteration. And remember that why we are interested in these special properties from the mixing permutation and S boxes.

(Refer Slide Time: 16:29)



DES Properties

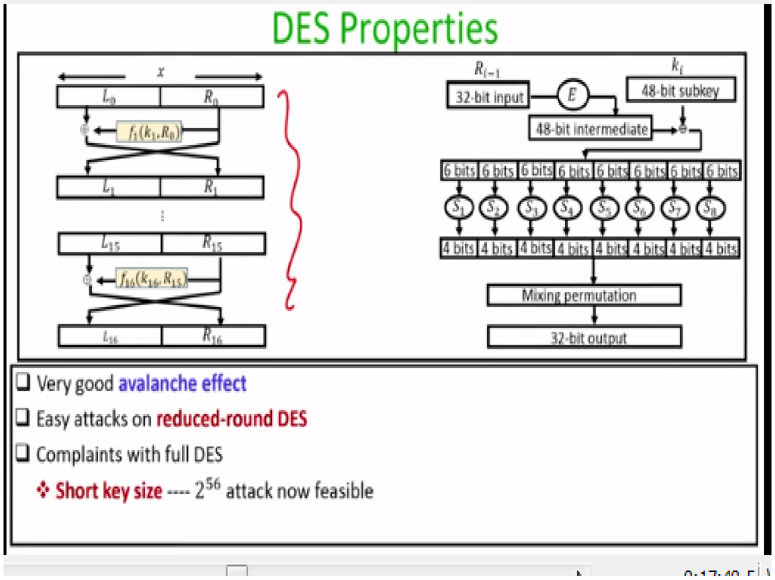
❑ Very good avalanche effect

$F_k(x)$

$F_k(x')$

Basically our goal here is to obtain good avalanche effect and just to recall what exactly is avalanche effect. The avalanche effect ensures that if you have a keyed permutation F_k, constructed using a Feistel network or SPN, then you require that F_k should have the property that $F_k(x)$ and $F_k(x')$ should differ in significant number of bit positions, even if x and x' differs only in less number of bits, say even in 1 bit. And we had seen in the last lecture that if you have some special properties from the mixing permutation and from the S boxes, we can achieve this avalanche effect.

(Refer Slide Time: 17:16)



DES Properties

❑ Very good avalanche effect

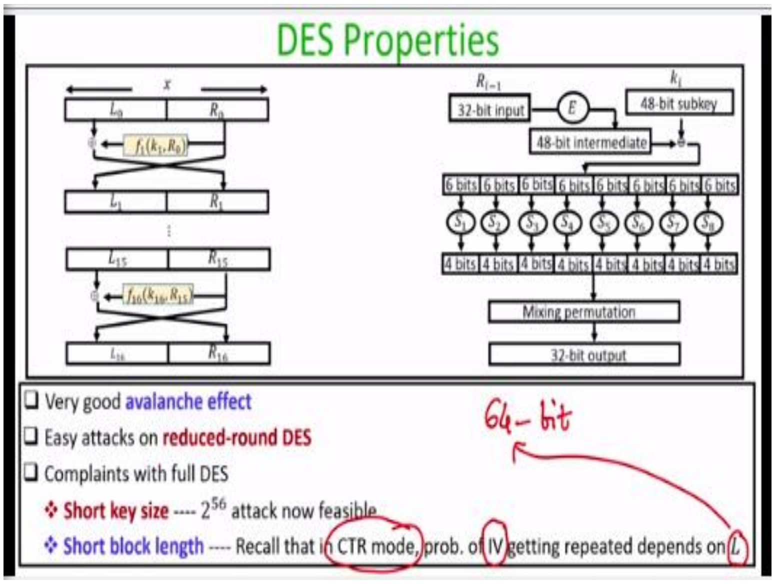
❑ Easy attacks on reduced-round DES

❑ Complaints with full DES

- Short key size ---- 2^{56} attack now feasible

And that is the principle we are following in the S boxes and a mixing permutation and expansion function that we are using in DES. It turns out that 16 rounds which we are using here in the Feistel network is very important. Because if you do not use a 16 round Feistel network and use a less round Feistel network, then there are easy attacks on the reduced round DES. But if you operate it with full 16 rounds, then the only complaints that have been reported till now is the short key size complaint. Namely, 2^{56} complexity brute force can be launched, given the current computing speed.

(Refer Slide Time: 17:52)



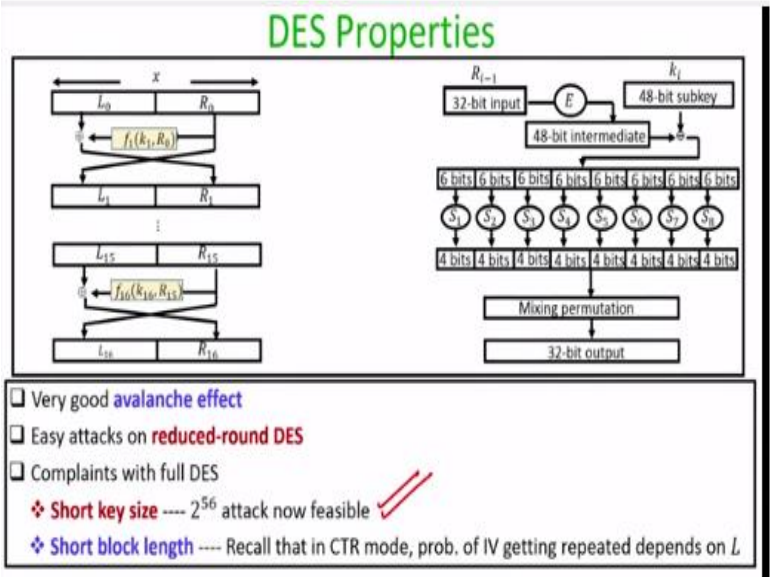
DES Properties

- □ Very good avalanche effect
- □ Easy attacks on reduced-round DES
- □ Complaints with full DES
  - ❖ Short key size ---- 2^{56} attack now feasible
  - ❖ Short block length ---- Recall that in CTR mode, prob. of IV getting repeated depends on L

64-bit

Another complaint with the DES is the short block length. So the block length here is 64 bits. And now you might be wondering that how a short block length can have a consequence on the security of the overall block cipher. So recall the counter mode of operation of the block cipher, or the pseudo-random permutation. And we had seen in the CPA-security analysis of the counter mode of operation that the CPA security of the counter mode of operation depends upon with how much probability the random IV that we use in encrypting the challenge plaintext gets repeated in any of the encryption-oracle queries. And the value of the IV and the probability of the IV getting repeated, depends upon the actual block length of the underlying keyed permutation. So L is the block size here and if my block size is small than the probability of the random IV which is used for encrypting the challenge plaintext getting repeated, becomes significantly high. So these are the 2 complaints reported with respect to the original construction of the DES. Apart from that no severe vulnerabilities have been reported till now.

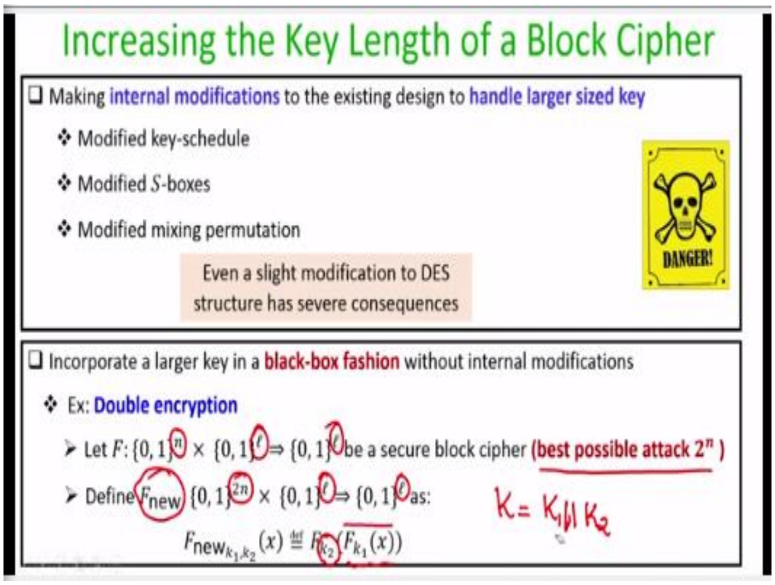
(Refer Slide Time: 19:16)



DES Properties

- ❑ Very good avalanche effect
- ❑ Easy attacks on reduced-round DES
- ❑ Complaints with full DES
  - ❖ Short key size ---- 2^{56} attack now feasible
  - ❖ Short block length ---- Recall that in CTR mode, prob. of IV getting repeated depends on L

(Refer Slide Time: 19:17)



Increasing the Key Length of a Block Cipher

- ❑ Making internal modifications to the existing design to handle larger sized key
  - ❖ Modified key-schedule
  - ❖ Modified S-boxes
  - ❖ Modified mixing permutation

Even a slight modification to DES structure has severe consequences

- ❑ Incorporate a larger key in a black-box fashion without internal modifications
  - ❖ Ex: Double encryption
    - ➢ Let $F: \{0,1\}^n \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ be a secure block cipher (best possible attack 2^n)
    - ➢ Define $F_{new}: \{0,1\}^{2n} \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ as:

$$F_{new_{k_1,k_2}}(x) \stackrel{def}{=} F_{k_2}(F_{k_1}(x))$$

$K = K_1 \| K_2$

So now let us see how we can go about increasing the size of the key. Our goal will be to increase the size of the original DES. And there are 2 approaches for designing a modified block cipher or modified DES where you can operate the modified construction with a larger sized key. The first approach will be to make internal modifications to the existing design to handle larger size key. Namely, you make modifications to the existing key scheduling algorithm, you make modifications to the S boxes, you make modification to the expansion function and mixing permutations and so on.

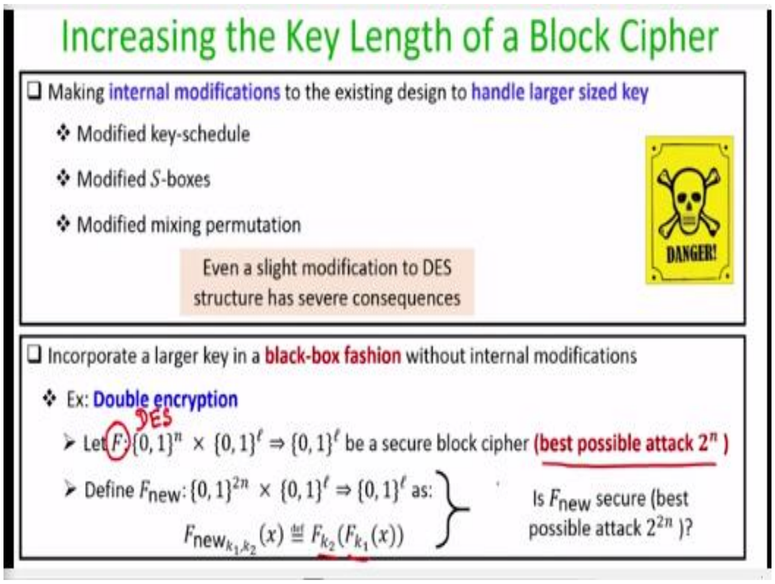
Unfortunately, this approach is not at all recommended, because it has turned out that even a slight modification to the existing DES structure has severe security consequences. So, we cannot afford to make modifications to the internal structure of the DES to ensure that we have a modified DES operating with a larger key size. The second approach which is recommended is to actually incorporate a larger key in a black box fashion without making any internal modifications.

For example, let us see how exactly a double encryption will work and the discussion that we are going to have will be very generic, it is applicable with respect to any block cipher. But for simplicity, you can always consider DES in your mind. So, imagine we have a keyed permutation which is a secure and practical block cipher, whose key size is n bits, block length is ℓ bits and output size is ℓ bits.

And since it is a practically secure block cipher, the best possible attack is of order 2^n. Now imagine I compose this existing construction to obtain what we call as a double encryption block cipher, which now operates with a key of size $2n$ bits. But a block length of ℓ bits and giving an output of ℓ bits. And the way I designed this double encryption block cipher is basically I parse the key k for the double encryption block cipher as 2 independent parts k_1 and k_2 each of n bits.

I operate first the existing block cipher with the k_1 part with the input x and I obtain an output y, which is of ℓ bits. And then treat that y output, as the input for the second invocation for the existing block cipher with the k_2 part of my double encryption block cipher key. So that is the way I am doing a double encryption.

(Refer Slide Time: 22:15)



Increasing the Key Length of a Block Cipher

❑ Making internal modifications to the existing design to handle larger sized key

- ❖ Modified key-schedule
- ❖ Modified S-boxes
- ❖ Modified mixing permutation

Even a slight modification to DES structure has severe consequences

❑ Incorporate a larger key in a black-box fashion without internal modifications

❖ Ex: Double encryption

DES

➢ Let $F: \{0,1\}^n \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ be a secure block cipher (best possible attack 2^n)

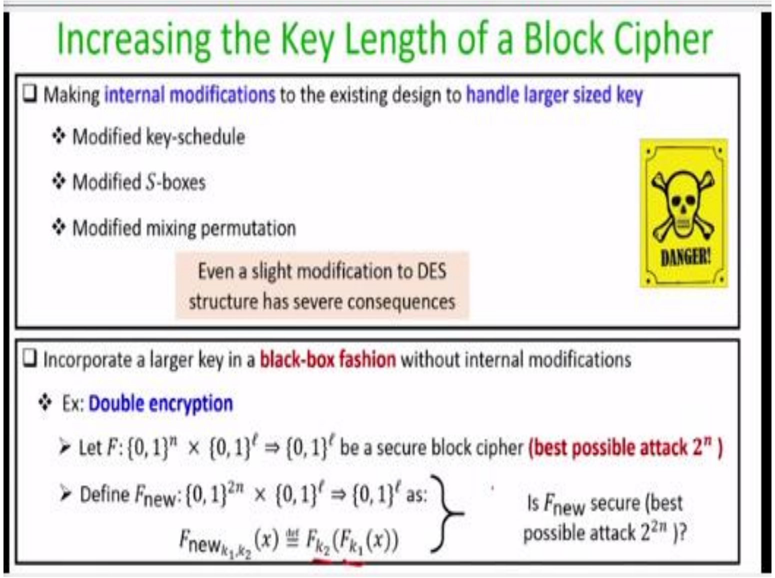
➢ Define $F_{\text{new}}: \{0,1\}^{2n} \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ as:

$$F_{\text{new}_{k_1,k_2}}(x) \stackrel{\text{def}}{=} F_{k_2}(F_{k_1}(x))$$

Is F_{new} secure (best possible attack 2^{2n})?

And as you can see here, I am not making any modification at all in the description or the design of the existing block cipher. The existing block cipher is operated as it is, for instance if your existing F is DES, then what we are basically trying to do here is we are trying to design double DES, where the size of the key will be now 112 bits. The first part will be k_1, the second part will be k_2. And basically what I am doing here is I am just invoking DES twice, and that is what my double DES is considered. So now we have to analyze or argue is the new double encryption block cipher that we have design is practically secure or not?

(Refer Slide Time: 23:00)



Increasing the Key Length of a Block Cipher

❑ Making internal modifications to the existing design to handle larger sized key

- ❖ Modified key-schedule
- ❖ Modified S-boxes
- ❖ Modified mixing permutation

Even a slight modification to DES structure has severe consequences

❑ Incorporate a larger key in a black-box fashion without internal modifications

❖ Ex: Double encryption

➢ Let $F: \{0,1\}^n \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ be a secure block cipher (best possible attack 2^n)

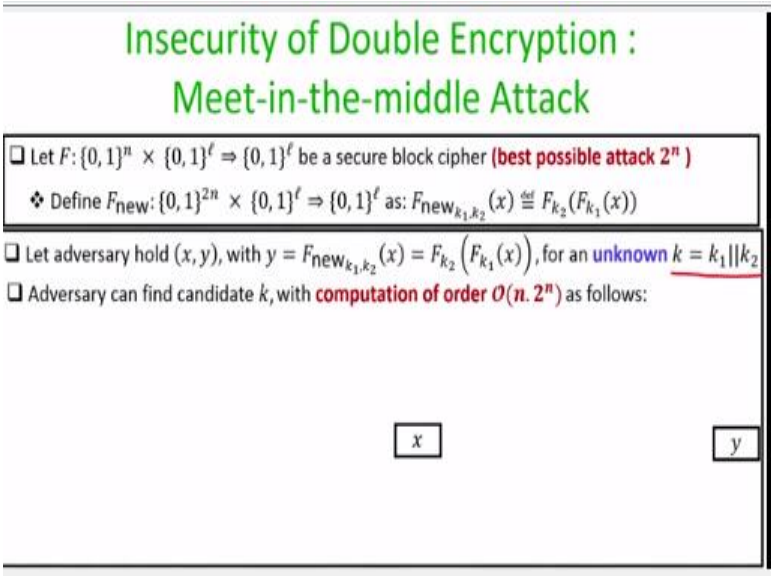
➢ Define $F_{\text{new}}: \{0,1\}^{2n} \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ as:

$$F_{\text{new}_{k_1,k_2}}(x) \stackrel{\text{def}}{=} F_{k_2}(F_{k_1}(x))$$

Is F_{new} secure (best possible attack 2^{2n})?

That means, is the best possible attack of order 2^{2n} or not?

(Refer Slide Time: 23:07)



Insecurity of Double Encryption :
Meet-in-the-middle Attack

❑ Let $F: \{0,1\}^n \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ be a secure block cipher **(best possible attack 2^n)**

❖ Define $F_{\text{new}}: \{0,1\}^{2n} \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ as: $F_{\text{new}_{k_1,k_2}}(x) \stackrel{\text{def}}{=} F_{k_2}(F_{k_1}(x))$

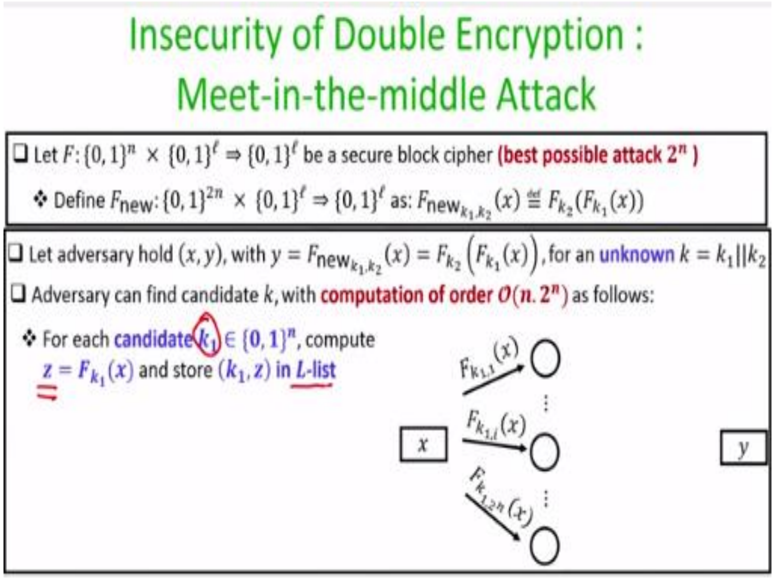
❑ Let adversary hold (x, y), with $y = F_{\text{new}_{k_1,k_2}}(x) = F_{k_2}\left(F_{k_1}(x)\right)$, for an **unknown** $k = k_1||k_2$

❑ Adversary can find candidate k, with **computation of order $O(n.2^n)$** as follows:

Interestingly, the answer is no, and there is a very interesting attack which we call as meet in the middle attack. Where we can show that by launching this meet in the middle attack, adversary can recover the unknown key of the double encryption block cipher with a complexity of order 2^n, which is much much less than 2^{2n}. So imagine that adversary has got several (x, y) pairs.

For simplicity imagine it has got one (x, y) pair, where y is the outcome of the double encryption block cipher with a known input x. But with an unknown key which is a concatenation of k_1 followed by k_2. And what we are going to show is an attack namely meet in the middle attack where an adversary can recover the unknown key with computation of order $n \times 2^n$.

(Refer Slide Time: 24:05)



Insecurity of Double Encryption : Meet-in-the-middle Attack

❑ Let $F: \{0,1\}^n \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ be a secure block cipher (best possible attack 2^n)

❖ Define $F_{\text{new}}: \{0,1\}^{2n} \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ as: $F_{\text{new}_{k_1,k_2}}(x) \stackrel{\text{def}}{=} F_{k_2}(F_{k_1}(x))$

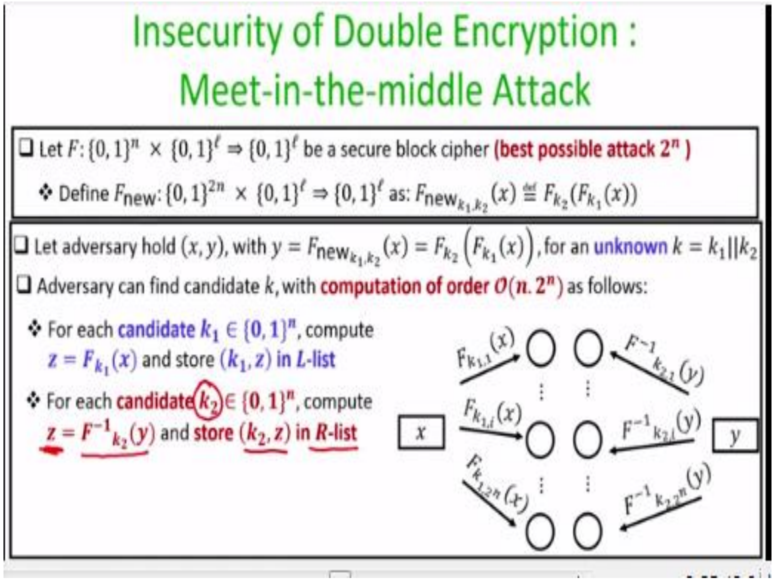
❑ Let adversary hold (x, y), with $y = F_{\text{new}_{k_1,k_2}}(x) = F_{k_2}\left(F_{k_1}(x)\right)$, for an unknown $k = k_1||k_2$

❑ Adversary can find candidate k, with computation of order $O(n.2^n)$ as follows:

❖ For each candidate $k_1 \in \{0,1\}^n$, compute $z = F_{k_1}(x)$ and store (k_1, z) in L-list

The idea here is as follows, basically adversary has to do brute force in 2 opposite directions. In the first direction, what it does is, it performs a brute force over all candidate k_1, and there are 2^n candidate k_1. For each candidate k_1, it operates the existing construction F with respect to the known input x and obtain the intermediate outputs, which it calls the z. And it stores all those (k_1, z) pairs in a list called L-list or left list, that is the first brute force it performs.

(Refer Slide Time: 24:46)



Insecurity of Double Encryption : Meet-in-the-middle Attack

❑ Let $F: \{0,1\}^n \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ be a secure block cipher (best possible attack 2^n)

❖ Define $F_{\text{new}}: \{0,1\}^{2n} \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ as: $F_{\text{new}_{k_1,k_2}}(x) \stackrel{\text{def}}{=} F_{k_2}(F_{k_1}(x))$

❑ Let adversary hold (x, y), with $y = F_{\text{new}_{k_1,k_2}}(x) = F_{k_2}\left(F_{k_1}(x)\right)$, for an unknown $k = k_1||k_2$

❑ Adversary can find candidate k, with computation of order $O(n.2^n)$ as follows:

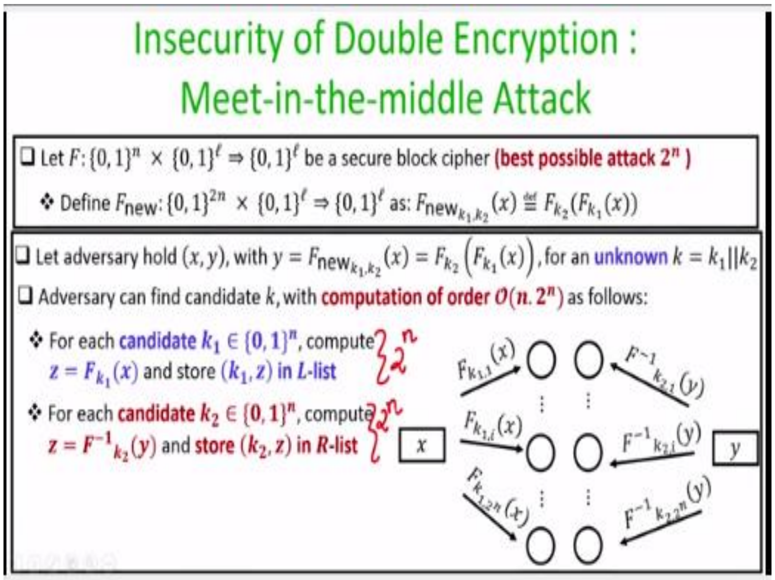
❖ For each candidate $k_1 \in \{0,1\}^n$, compute $z = F_{k_1}(x)$ and store (k_1, z) in L-list

❖ For each candidate $k_2 \in \{0,1\}^n$, compute $z = F^{-1}{}_{k_2}(y)$ and store (k_2, z) in R-list

And what it performs is that it takes the y output which is available with the adversary and it now performs the reverse brute force. Namely it performs the brute force with respect to all candidate k_2. But with respect to the inverse of the function F on the output y, namely it takes each candidate

k_2 and compute the inverse of the output y available to it under that candidate k_2 and obtain the corresponding z and store these (k_2, z) pairs in a list called R-list. So that is the second brute force it is performing.

(Refer Slide Time: 25:25)



Insecurity of Double Encryption :
Meet-in-the-middle Attack

❑ Let $F: \{0,1\}^n \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ be a secure block cipher (best possible attack 2^n)

❖ Define $F_{\text{new}}: \{0,1\}^{2n} \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ as: $F_{\text{new}_{k_1,k_2}}(x) \overset{\text{def}}{=} F_{k_2}(F_{k_1}(x))$

❑ Let adversary hold (x, y), with $y = F_{\text{new}_{k_1,k_2}}(x) = F_{k_2}\left(F_{k_1}(x)\right)$, for an unknown $k = k_1||k_2$

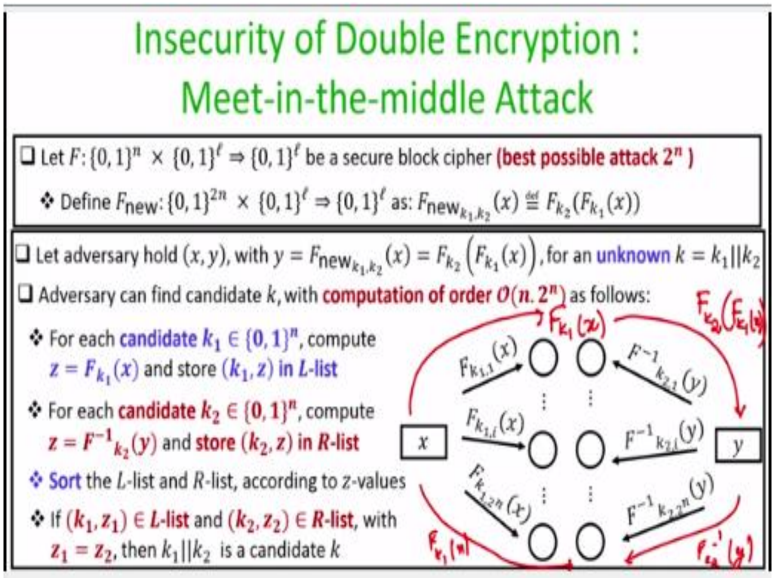
❑ Adversary can find candidate k, with computation of order $O(n.2^n)$ as follows:

❖ For each candidate $k_1 \in \{0,1\}^n$, compute $z = F_{k_1}(x)$ and store (k_1, z) in L-list

❖ For each candidate $k_2 \in \{0,1\}^n$, compute $z = F^{-1}{}_{k_2}(y)$ and store (k_2, z) in R-list

So, the first brute force it performs its complexity is of order 2^n, the second brute force that it has performed parallelly its complexity is of order 2^n. And now, what adversary has to do is, it sorts the left list and the right list right according to the z values.

(Refer Slide Time: 22:51)



Insecurity of Double Encryption :
Meet-in-the-middle Attack

❑ Let $F: \{0,1\}^n \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ be a secure block cipher (best possible attack 2^n)

❖ Define $F_{\text{new}}: \{0,1\}^{2n} \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ as: $F_{\text{new}_{k_1,k_2}}(x) \overset{\text{def}}{=} F_{k_2}(F_{k_1}(x))$

❑ Let adversary hold (x, y), with $y = F_{\text{new}_{k_1,k_2}}(x) = F_{k_2}\left(F_{k_1}(x)\right)$, for an unknown $k = k_1||k_2$

❑ Adversary can find candidate k, with computation of order $O(n.2^n)$ as follows:

❖ For each candidate $k_1 \in \{0,1\}^n$, compute $z = F_{k_1}(x)$ and store (k_1, z) in L-list

❖ For each candidate $k_2 \in \{0,1\}^n$, compute $z = F^{-1}{}_{k_2}(y)$ and store (k_2, z) in R-list

❖ Sort the L-list and R-list, according to z-values

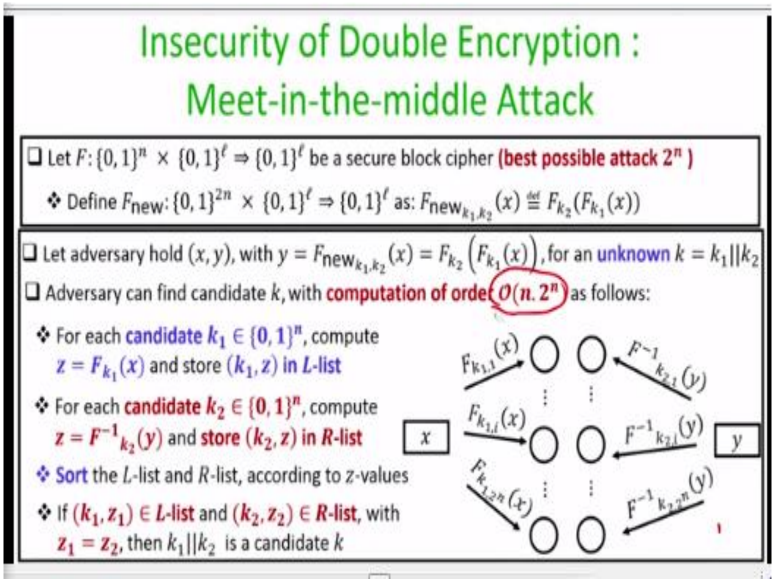
❖ If $(k_1, z_1) \in$ L-list and $(k_2, z_2) \in$ R-list, with $z_1 = z_2$, then $k_1||k_2$ is a candidate k

And once it has the sorted L-list and R-list according to the z values, what adversary basically has to do is, it has to see whether it has a (k_1, z_1) pair in the L-list and a (k_2, z_2) pair in the R-list, such that $z_1 = z_2$ holds. If that is the case, then the candidate k_1 concatenated with the candidate k_2 is the actual candidate k used for the double encryption. Now, what is exactly is the idea of this attack? If you see the construction of the double encryption, to do the double encryption, from x we go to $F_{k_1}(x)$. And then from $F_k(x)$ we would have gone to y, namely we would have obtained $F_{k_2}(F_{k_1}(x))$. Another way to interpret the same construction is that from y, I could have actually performed $F_{k_2}^{-1}(y)$. And from here, the same thing I could have obtained if I would have computed $F_{k_1}(x)$.

So, that is basically adversary is trying to do, adversary brute forcing in the forward direction, it is brute forcing in the reverse direction. And trying to see whether for some candidate k_1 and for candidate k_2, it reaches the same point or not. And definitely there will be one candidate k_1 and one candidate k_2 where the meeting will happen. That meeting point basically gives him the candidate k.

And to obtain the meeting point adversary basically have to perform a computation of order $n2^n$**.** It has to perform 2 brute forces and then it has to do the sorting, and then it has to find down the meeting point. So, as you can see, even though we have now constructed a double encryption block cipher, with a key of size $2n$ bits. And we expect that the best possible attack should be of order 2^{2n}.

(Refer Slide Time: 27:55)



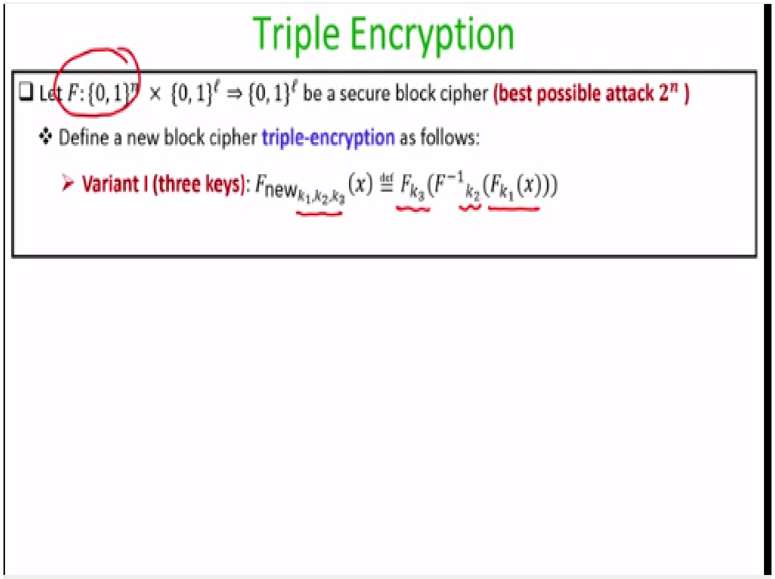
Insecurity of Double Encryption : Meet-in-the-middle Attack

- ❑ Let $F: \{0,1\}^n \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ be a secure block cipher (best possible attack 2^n)
  - ❖ Define $F_{\text{new}}: \{0,1\}^{2n} \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ as: $F_{\text{new}_{k_1,k_2}}(x) \overset{\text{def}}{=} F_{k_2}(F_{k_1}(x))$

- ❑ Let adversary hold (x, y), with $y = F_{\text{new}_{k_1,k_2}}(x) = F_{k_2}\left(F_{k_1}(x)\right)$, for an unknown $k = k_1||k_2$
- ❑ Adversary can find candidate k, with computation of order $O(n.2^n)$ as follows:
  - ❖ For each candidate $k_1 \in \{0,1\}^n$, compute $z = F_{k_1}(x)$ and store (k_1, z) in L-list
  - ❖ For each candidate $k_2 \in \{0,1\}^n$, compute $z = F^{-1}{}_{k_2}(y)$ and store (k_2, z) in R-list
  - ❖ Sort the L-list and R-list, according to z-values
  - ❖ If $(k_1, z_1) \in L$-list and $(k_2, z_2) \in R$-list, with $z_1 = z_2$, then $k_1||k_2$ is a candidate k

It turns out that by doing a computation of only order $n2^n$, the adversary can recover the key. And hence increasing the key size by doing a double encryption is not sufficient.

(Refer Slide Time: 28:03)



Triple Encryption

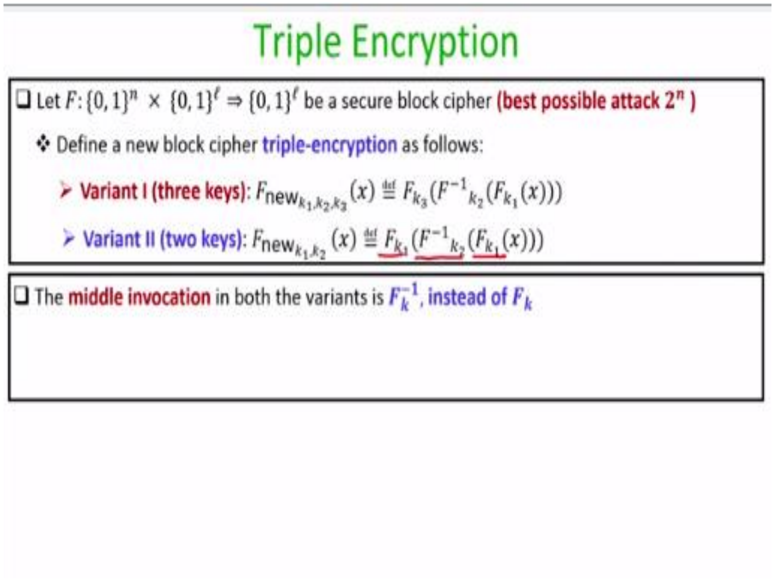
- ❑ Let $F: \{0,1\}^n \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ be a secure block cipher (best possible attack 2^n)
  - ❖ Define a new block cipher triple-encryption as follows:
    - ➤ Variant I (three keys): $F_{\text{new}_{k_1,k_2,k_3}}(x) \overset{\text{def}}{=} F_{k_3}(F^{-1}{}_{k_2}(F_{k_1}(x)))$

So that leads us to what we call us triple encryption. So again, imagine you have an existing, practically secure block cipher. And now we are defining a new block cipher based on triple encryption. We have 2 possible variants possible. In the first variant, we actually operate the existing secure construction with 3 independent keys of size n bits as follows:

$$F_{\text{new}_{k_1,\, k_2,k_3}}(x) \overset{\text{def}}{=} F_{k_3}(F^{-1}{}_{k_2}(F_{k_1}(x)))$$

We first apply the existing construction on x with respect to the k_1 portion of the key, whatever output I obtain, I invert it with respect to the k_2 portion, whatever output I obtain, again I apply the existing construction with respect to the k_3 portion of the key. And that is what going to be the overall output of the triple encryption with respect to the key namely $k_1||k_2||k_3$, that is the first variant.

(Refer Slide Time: 29:05)



Triple Encryption

❑ Let $F: \{0,1\}^n \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ be a secure block cipher (best possible attack 2^n)

❖ Define a new block cipher triple-encryption as follows:

➢ Variant I (three keys): $F_{\text{new}_{k_1,k_2,k_3}}(x) \overset{\text{def}}{=} F_{k_3}(F^{-1}{}_{k_2}(F_{k_1}(x)))$

➢ Variant II (two keys): $F_{\text{new}_{k_1,k_2}}(x) \overset{\text{def}}{=} \underline{F_{k_1}}(\underline{F^{-1}{}_{k_2}}(\underline{F_{k_1}}(x)))$

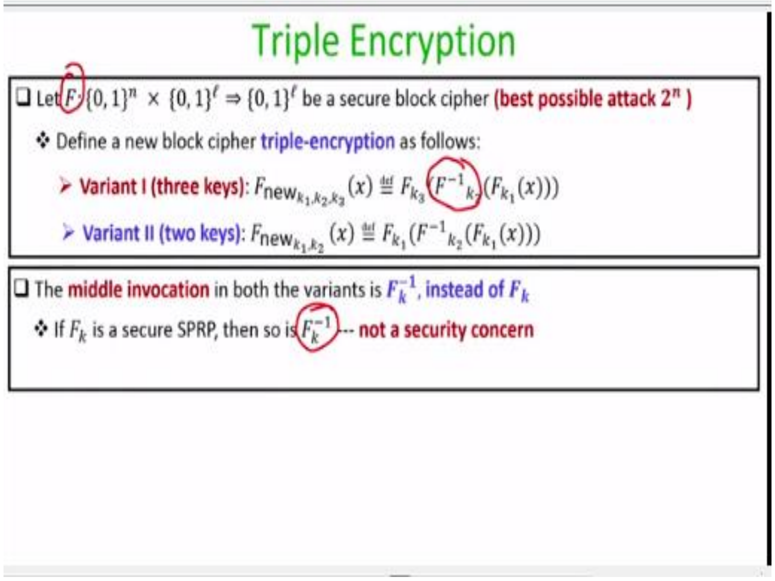
❑ The middle invocation in both the variants is F_k^{-1}, instead of F_k

The second variant actually is going to operate on with respect to only 2 independent keys of size n bits each as follows:

$$F_{\text{new}_{k_1,\,k_2}}(x) \overset{\text{def}}{=} F_{k_1}(F^{-1}{}_{k_2}(F_{k_1}(x)))$$

It will invoke, 3 invocations of the F, the first invocation will be F_{k_1}, the second invocation will be $F_{k_2}^{-1}$ and again, the third invocation will be F_{k_1}.

(Refer Slide Time: 29:2)

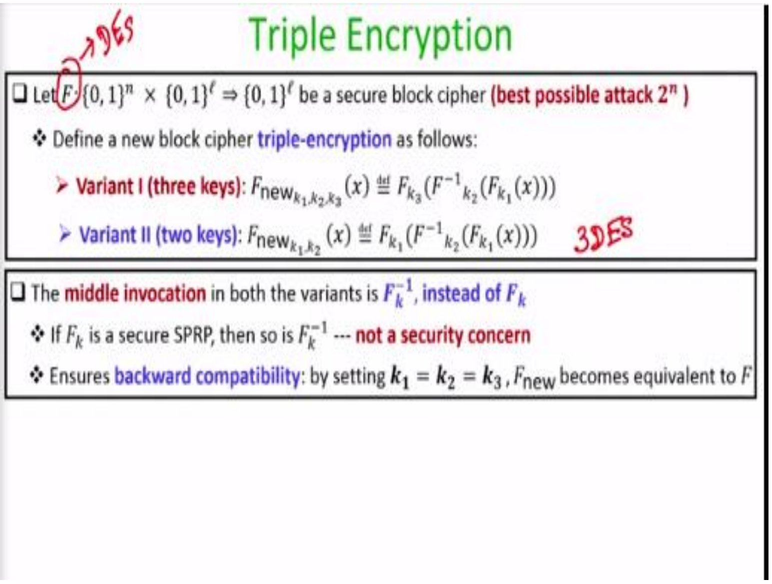


Triple Encryption

- ❑ Let $F: \{0,1\}^n \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ be a secure block cipher (best possible attack 2^n)
  - ❖ Define a new block cipher triple-encryption as follows:
    - ➢ Variant I (three keys): $F\text{new}_{k_1,k_2,k_3}(x) \overset{\text{def}}{=} F_{k_3}({F^{-1}}_{k_2}(F_{k_1}(x)))$
    - ➢ Variant II (two keys): $F\text{new}_{k_1,k_2}(x) \overset{\text{def}}{=} F_{k_1}({F^{-1}}_{k_2}(F_{k_1}(x)))$
- ❑ The middle invocation in both the variants is F_k^{-1}, instead of F_k
  - ❖ If F_k is a secure SPRP, then so is F_k^{-1} --- not a security concern

Now you might be wondering that why the middle invocation in both the variants is F_k^{-1}, instead of F_k. Well, security wise it is not a concern, because if the existing construction F_k is a practically secure strong pseudo random permutation or block cipher, then we can prove that so is F_k^{-1}. So it does not matter whether the middle invocation is F_{k_2} or $F_{k_2}^{-1}$, both are sufficient as far as security is concerned.

(Refer Slide Time: 30:00)



Triple Encryption

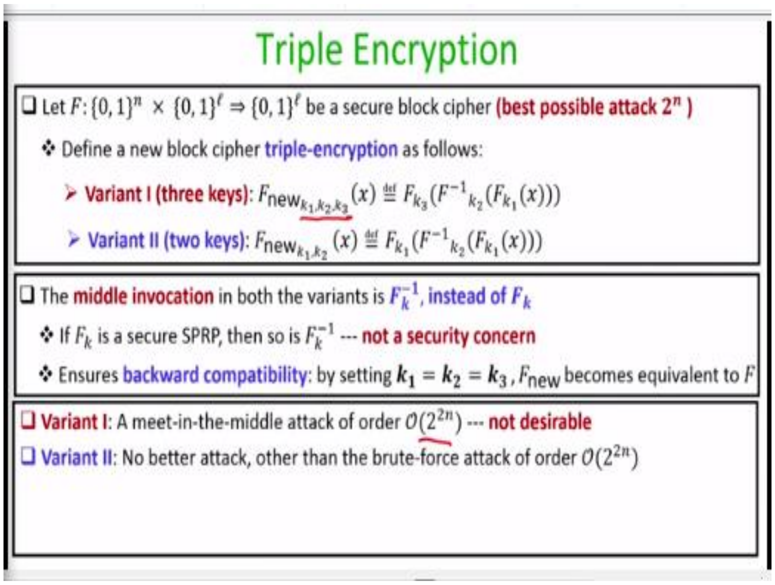
→3DES

- ❑ Let $F: \{0,1\}^n \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ be a secure block cipher (best possible attack 2^n)
  - ❖ Define a new block cipher triple-encryption as follows:
    - ➢ Variant I (three keys): $F\text{new}_{k_1,k_2,k_3}(x) \overset{\text{def}}{=} F_{k_3}({F^{-1}}_{k_2}(F_{k_1}(x)))$
    - ➢ Variant II (two keys): $F\text{new}_{k_1,k_2}(x) \overset{\text{def}}{=} F_{k_1}({F^{-1}}_{k_2}(F_{k_1}(x)))$ 3DES
- ❑ The middle invocation in both the variants is F_k^{-1}, instead of F_k
  - ❖ If F_k is a secure SPRP, then so is F_k^{-1} --- not a security concern
  - ❖ Ensures backward compatibility: by setting $\boldsymbol{k_1 = k_2 = k_3}$, F_{new} becomes equivalent to F

The only reason that the middle invocation is an inverse function and not a forward function is to ensure backward compatibility. Namely, if I ensure that the k_1 portion of the key and the k_2 portion of the key and the k_3 portion of the key for the triple encryption block cipher are all same, then basically the triple encryption based block cipher becomes equivalent to the existing construction. That means, if my existing construction F was DES and my triple encryption based construction is triple DES. Then if I have a software implementation of triple DES and if I want to get an effect of original DES, then I do not have to do any modification in the software implementation of triple DES, what basically I have to do is I take a key of size $3n$ bits where I ensure that the first portion of the key and the second portion of the key and the third portion of the key are all same.

If I use the special type of key and use the software implementation of triple DES, I basically end up getting the effect of original DES. So it is just for backward compatibility that the middle invocation is the inverse function invocation.

(Refer Slide Time: 31:14)



Triple Encryption

❑ Let $F: \{0,1\}^n \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ be a secure block cipher (best possible attack 2^n)

- ❖ Define a new block cipher triple-encryption as follows:
  - ➢ Variant I (three keys): $F\text{new}_{\underline{k_1,k_2,k_3}}(x) \overset{\text{def}}{=} F_{k_3}(F^{-1}{}_{k_2}(F_{k_1}(x)))$
  - ➢ Variant II (two keys): $F\text{new}_{k_1,k_2}(x) \overset{\text{def}}{=} F_{k_1}(F^{-1}{}_{k_2}(F_{k_1}(x)))$

❑ The middle invocation in both the variants is F_k^{-1}, instead of F_k

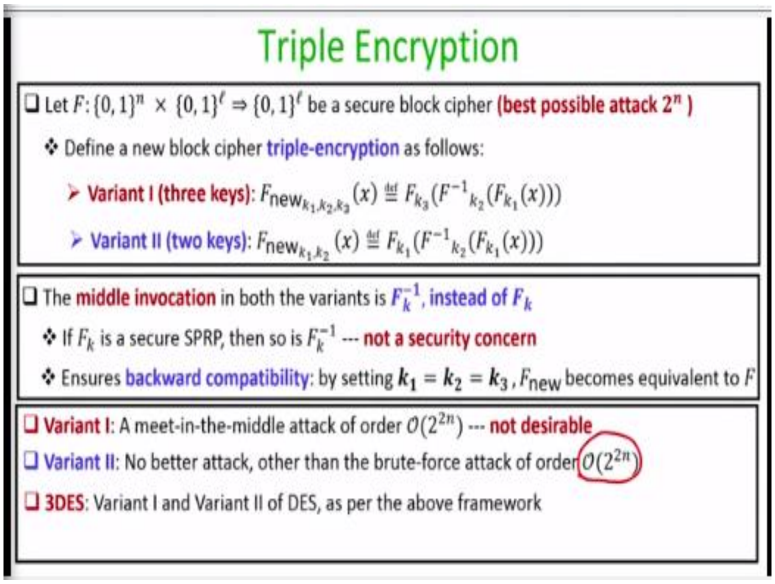
- ❖ If F_k is a secure SPRP, then so is F_k^{-1} --- not a security concern
- ❖ Ensures backward compatibility: by setting $\boldsymbol{k_1} = \boldsymbol{k_2} = \boldsymbol{k_3}$, F_{new} becomes equivalent to F

❑ Variant I: A meet-in-the-middle attack of order $\mathcal{O}(\underline{2^{2n}})$ --- not desirable

❑ Variant II: No better attack, other than the brute-force attack of order $\mathcal{O}(2^{2n})$

Now, it turns out that as far as the security is concerned of this triple encryption based block cipher, for the variant one, there is a meet in the middle attack of order 2^{2n} and that is why it is not acceptable. Because we are operating with a key of size $3n$ bits and we expect that the best possible attack should be of order 2^{3n}. But we can launch a meet in the middle attack whose complexity is of order 2^{2n}, so we are not getting the desired effect.

(Refer Slide Time: 31:47)



Triple Encryption

❑ Let $F: \{0,1\}^n \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ be a secure block cipher (best possible attack 2^n)

❖ Define a new block cipher triple-encryption as follows:

➢ Variant I (three keys): $F_{\text{new}_{k_1,k_2,k_3}}(x) \overset{\text{def}}{=} F_{k_3}(F^{-1}{}_{k_2}(F_{k_1}(x)))$

➢ Variant II (two keys): $F_{\text{new}_{k_1,k_2}}(x) \overset{\text{def}}{=} F_{k_1}(F^{-1}{}_{k_2}(F_{k_1}(x)))$

❑ The middle invocation in both the variants is F_k^{-1}, instead of F_k

❖ If F_k is a secure SPRP, then so is F_k^{-1} --- not a security concern

❖ Ensures backward compatibility: by setting $k_1 = k_2 = k_3$, F_{new} becomes equivalent to F

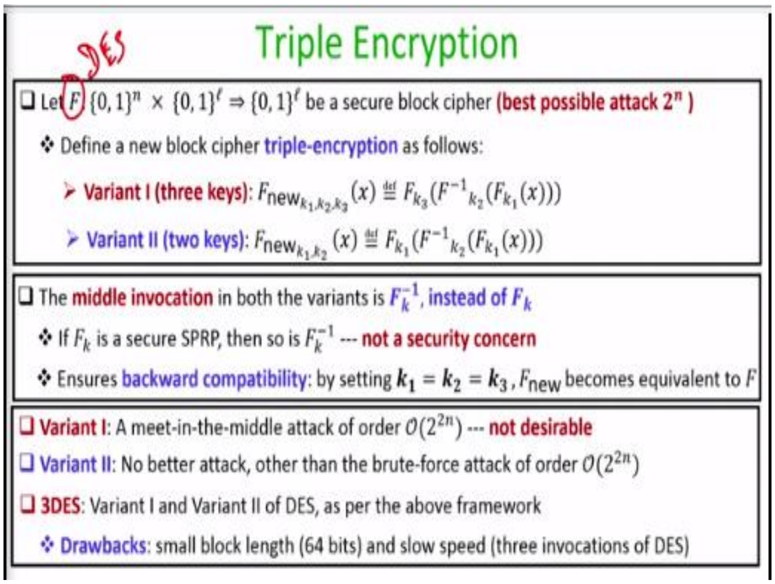
❑ Variant I: A meet-in-the-middle attack of order $O(2^{2n})$ --- not desirable

❑ Variant II: No better attack, other than the brute-force attack of order $O(2^{2n})$

❑ 3DES: Variant I and Variant II of DES, as per the above framework

Whereas for the variant 2, where we are actually using a key of size $2n$ bits, there is no better attack reported other than brute forcing, whose complexity is of order 2 to the power 2^{2n}. And that is what we desire.

(Refer Slide Time: 32:03)



DES

Triple Encryption

❑ Let $F: \{0,1\}^n \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ be a secure block cipher (best possible attack 2^n)

❖ Define a new block cipher triple-encryption as follows:

➢ Variant I (three keys): $F_{\text{new}_{k_1,k_2,k_3}}(x) \overset{\text{def}}{=} F_{k_3}(F^{-1}{}_{k_2}(F_{k_1}(x)))$

➢ Variant II (two keys): $F_{\text{new}_{k_1,k_2}}(x) \overset{\text{def}}{=} F_{k_1}(F^{-1}{}_{k_2}(F_{k_1}(x)))$

❑ The middle invocation in both the variants is F_k^{-1}, instead of F_k

❖ If F_k is a secure SPRP, then so is F_k^{-1} --- not a security concern

❖ Ensures backward compatibility: by setting $k_1 = k_2 = k_3$, F_{new} becomes equivalent to F

❑ Variant I: A meet-in-the-middle attack of order $O(2^{2n})$ --- not desirable

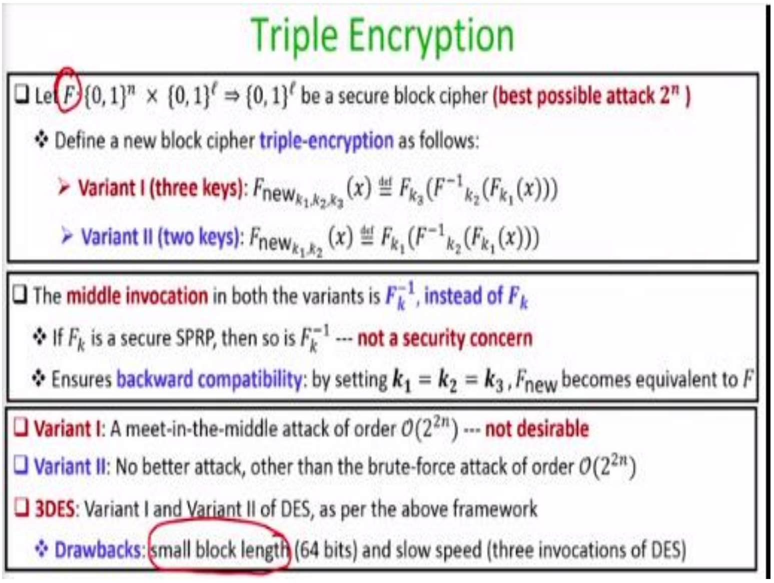
❑ Variant II: No better attack, other than the brute-force attack of order $O(2^{2n})$

❑ 3DES: Variant I and Variant II of DES, as per the above framework

❖ Drawbacks: small block length (64 bits) and slow speed (three invocations of DES)

So, triple DES is basically a variant of DES where the existing F is the DES. And we basically either use variant 1 or variant 2 but preferably variant 2, because that is where we get the maximum possible security. However, it turns out that even though by using triple DES, we actually get rid of the small key size complaint that was there with respect to the original DES,

(Refer Slide Time: 32:31)



Triple Encryption

❑ Let F $\{0,1\}^n \times \{0,1\}^\ell \Rightarrow \{0,1\}^\ell$ be a secure block cipher (best possible attack 2^n)

❖ Define a new block cipher triple-encryption as follows:

➢ Variant I (three keys): $F\text{new}_{k_1,k_2,k_3}(x) \stackrel{\text{def}}{=} F_{k_3}(F^{-1}{}_{k_2}(F_{k_1}(x)))$

➢ Variant II (two keys): $F\text{new}_{k_1,k_2}(x) \stackrel{\text{def}}{=} F_{k_1}(F^{-1}{}_{k_2}(F_{k_1}(x)))$

❑ The middle invocation in both the variants is F_k^{-1}, instead of F_k

❖ If F_k is a secure SPRP, then so is F_k^{-1} --- not a security concern

❖ Ensures backward compatibility: by setting $\boldsymbol{k_1} = \boldsymbol{k_2} = \boldsymbol{k_3}$, F_{new} becomes equivalent to F

❑ Variant I: A meet-in-the-middle attack of order $O(2^{2n})$ --- not desirable

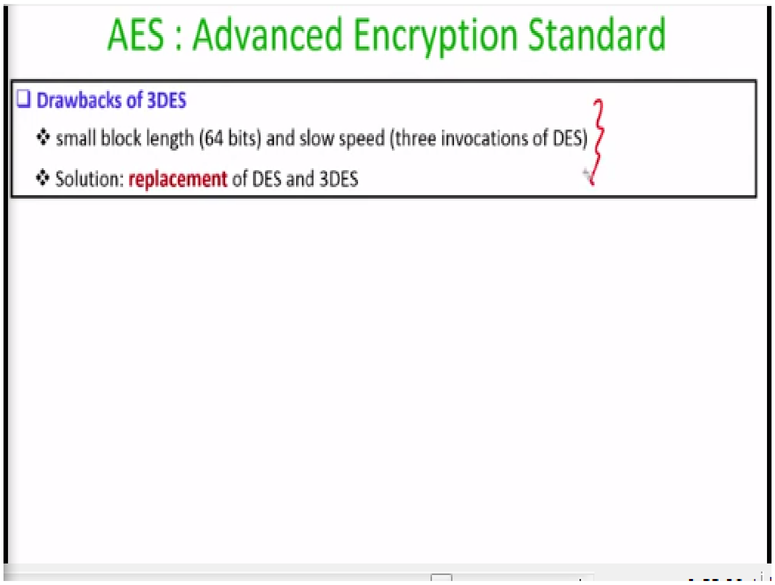
❑ Variant II: No better attack, other than the brute-force attack of order $O(2^{2n})$

❑ 3DES: Variant I and Variant II of DES, as per the above framework

❖ Drawbacks: small block length (64 bits) and slow speed (three invocations of DES)

the complaint with respect to the small block length still remains even in triple DES. Because in the triple DES the block length remains as it is as was the case for the original F, so we are not increasing the block length. Moreover to actually get more security, the overall speed of the triple DES is now slow, because internally it requires 3 invocations of DES.

(Refer Slide Time: 32:54)



AES : Advanced Encryption Standard

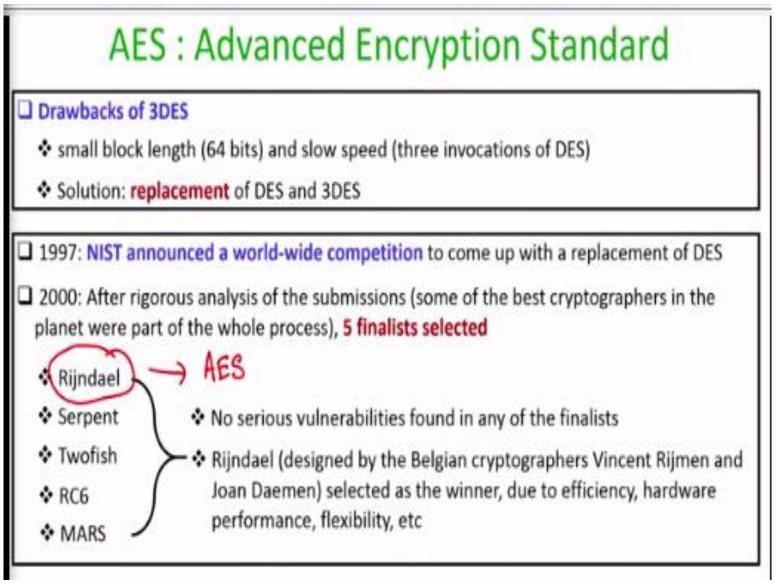
❑ Drawbacks of 3DES

❖ small block length (64 bits) and slow speed (three invocations of DES)

❖ Solution: replacement of DES and 3DES

So, this is the summary of the drawbacks of DES, namely small block length and slow speed. And to get rid of these 2 limitations, the solution will be to replace DES and triple DES and come up with a new practical instantiation of block cipher.

(Refer Slide Time: 33:14)



AES : Advanced Encryption Standard

❑ **Drawbacks of 3DES**

- ❖ small block length (64 bits) and slow speed (three invocations of DES)
- ❖ Solution: **replacement** of DES and 3DES

❑ 1997: **NIST announced a world-wide competition** to come up with a replacement of DES

❑ 2000: After rigorous analysis of the submissions (some of the best cryptographers in the planet were part of the whole process), **5 finalists selected**

- ❖ Rijndael → AES
- ❖ Serpent
- ❖ Twofish
- ❖ RC6
- ❖ MARS

- ❖ No serious vulnerabilities found in any of the finalists
- ❖ Rijndael (designed by the Belgian cryptographers Vincent Rijmen and Joan Daemen) selected as the winner, due to efficiency, hardware performance, flexibility, etc

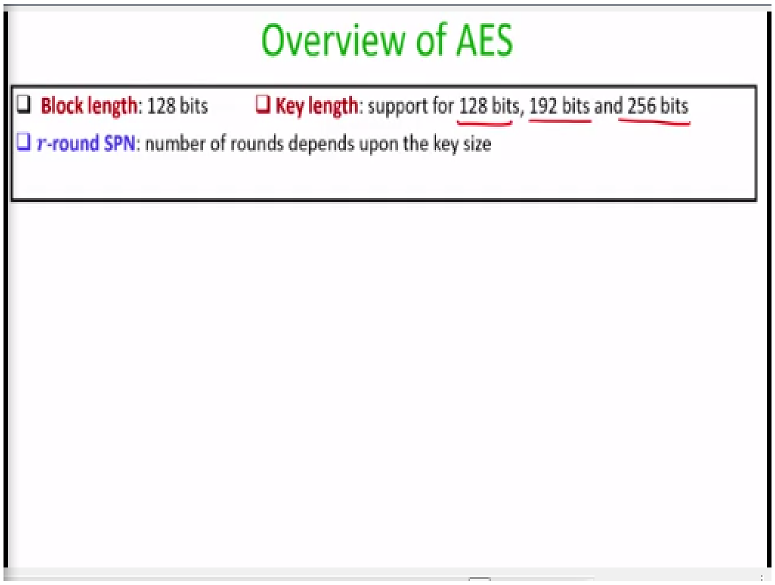
So motivated by that, in 1997 NIST announced a worldwide competition, first of it is kind to come up with a replacement of DES. And after rigorous analysis for 3 years, where some of the best cryptographers in the planet submitted their proposals, each proposal was rigorously analyzed in the public domain, after 3 years of rigorous analysis, 5 finalists were selected: Rijndael, Serpent, Twofish, RC6 and MARS.

And it turns out that in none of these 5 finalists, any serious vulnerabilities were reported. So it is fine if you use any of these 5 finalists as a replacement of DES. But a preference was given for the proposal submitted by the Belgian cryptographers, Vincent Rijmen and Joan Daemen. And it was selected as the winner due to efficiency, hardware performance, flexibility, etc. And the proposal submitted by this Belgian cryptographers eventually becomes the Advanced Encryption Standard, namely the replacement of the DES. So I would like to stress here that why we should believe that AES as indeed a secure block cipher. Because it came out as a result of an open worldwide competition, where actually the submissions came from some of the best cryptographers in the planet.

So this actually supports the argument given by the Kirchhoff based on the Kirchhoff principle where we actually argue that you should not try to achieve security based on obscurity by keeping the description of the design secret. You should always use an encryption process whose design is available in the public domain and which has gone through rigorous security analysis and so on.

And AES is one of those cryptographic algorithms which has gone through 3 years of rigorous security analysis and no security breach has been reported. So, we should feel more comfortable in using schemes like AES which have come out of worldwide competitions.

(Refer Slide Time: 35:36)



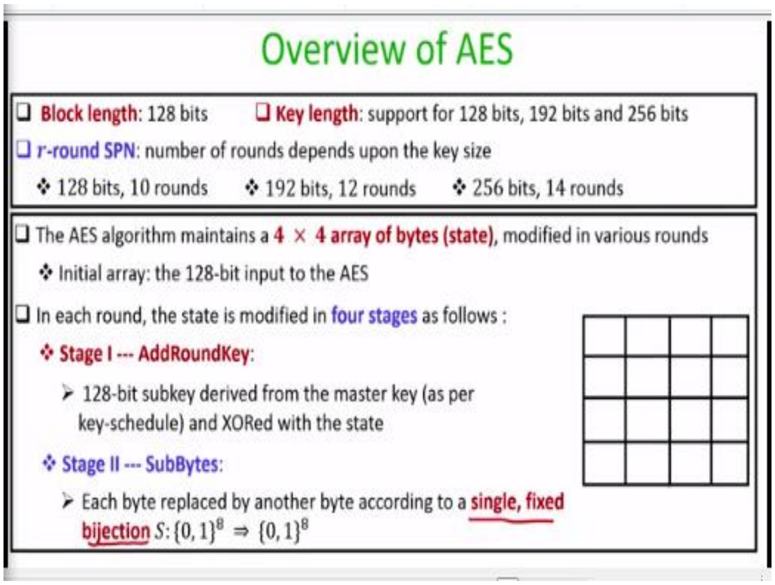
Overview of AES

□ Block length: 128 bits □ Key length: support for 128 bits, 192 bits and 256 bits

□ r-round SPN: number of rounds depends upon the key size

So, what we will do now is we will see an overview of a AES. So its block length is 128 bits and it supports basically 3 versions of key. It has support for 128 bit key size, 192 bit key size and 256 bit key size. And architecture wise, unlike DES which was a combination of SPN and Feistel network, AES is just an r-round SPN where the number of rounds namely the value of r, depends upon the key size.

(Refer Slide Time: 36:07)



Overview of AES

□ Block length: 128 bits □ Key length: support for 128 bits, 192 bits and 256 bits

□ r-round SPN: number of rounds depends upon the key size

❖ 128 bits, 10 rounds ❖ 192 bits, 12 rounds ❖ 256 bits, 14 rounds

□ The AES algorithm maintains a 4×4 array of bytes (state), modified in various rounds

❖ Initial array: the 128-bit input to the AES

□ In each round, the state is modified in four stages as follows :

❖ Stage I --- AddRoundKey:

➢ 128-bit subkey derived from the master key (as per key-schedule) and XORed with the state

❖ Stage II --- SubBytes:

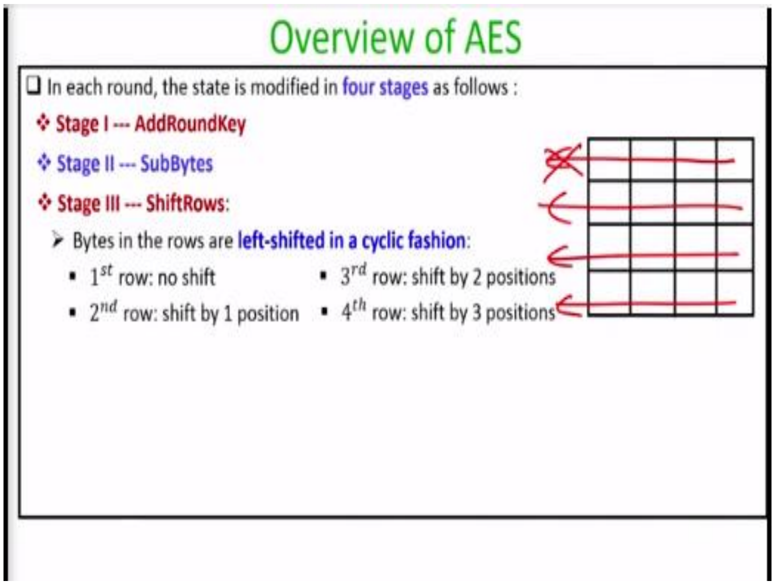
➢ Each byte replaced by another byte according to a single, fixed bijection $S: \{0,1\}^8 \Rightarrow \{0,1\}^8$

If you are operating with 128 bit key size, then the number of rounds is 10, if you are operating with 192 bit key size the number of rounds is 12, if you are operating with 256 bit key size, the number of rounds is 14. And since it is an SPN basically you will be doing r iterations of key mixing followed by substitution, followed by permutation. And then after r iterations, you will be doing a final key mixing.

So basically what AES does is, it maintains a 4×4 array of bytes which is also called as the state and in each iteration in each round the state gets modified. The initial array or the initial state is the 128 bit block input for the AES. And the block input x is parsed as a collection of 16 bytes, so that is why this 4×4 array of bytes. So in each iteration we basically do 4 stages, the first stage is the add round key which is equivalent to the key-mixing step, where a 128 bit sub-key derived from the master key is XORed with the current state.

This is followed by the substitution stage, which is also called as the sub-bytes stagae where each byte in the current state is replaced by another byte according to a single fixed bijection, so this is again another difference with respect to DES. In DES we had 8 different publicly known S boxes, but in AES we have only a single S box, which is applied across all the iteration in all the stages. So once we do the key-mixing, we do the sub-bytes, namely the substitution stage.

(Refer Slide Time: 37:51)



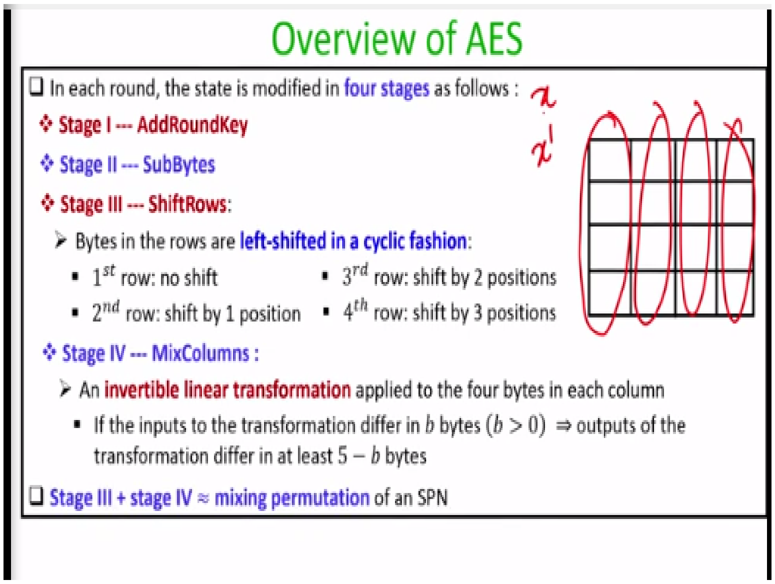
Overview of AES

❑ In each round, the state is modified in **four stages** as follows :

- ❖ **Stage I --- AddRoundKey**
- ❖ **Stage II --- SubBytes**
- ❖ **Stage III --- ShiftRows**:
  - ➢ Bytes in the rows are **left-shifted in a cyclic fashion**:
    - ▪ 1^{st} row: no shift
    - ▪ 2^{nd} row: shift by 1 position
    - ▪ 3^{rd} row: shift by 2 positions
    - ▪ 4^{th} row: shift by 3 positions

Then in stage 3, we would actually perform 2 operations. We first do the shifting of the rows, where the bytes in each of the rows are left shifted in a cyclic fashion. The order of the shift is determined as follows. Basically, we do no shifting in the first row, for the second row, the contents get shifted in a cyclic fashion by 1 byte position. For the third row, the contents get shifted in a cyclic fashion by 2 byte position, and in the fourth row the bytes contents get shifted by 3 positions in the cyclic fashion, so that is publicly known.

(Refer Slide Time: 38:35)



Overview of AES

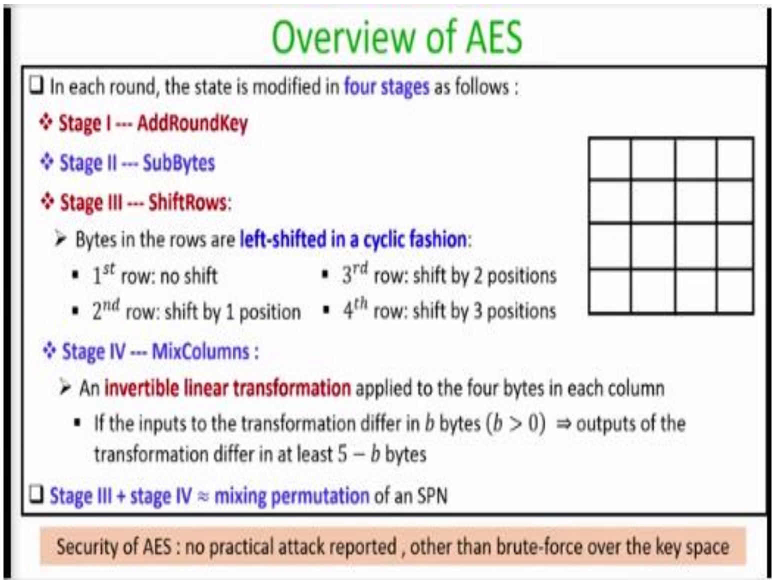
❑ In each round, the state is modified in four stages as follows :

- ❖ Stage I --- AddRoundKey
- ❖ Stage II --- SubBytes
- ❖ Stage III --- ShiftRows:
  - ➢ Bytes in the rows are left-shifted in a cyclic fashion:
    - ▪ 1^{st} row: no shift
    - ▪ 2^{nd} row: shift by 1 position
    - ▪ 3^{rd} row: shift by 2 positions
    - ▪ 4^{th} row: shift by 3 positions
- ❖ Stage IV --- MixColumns :
  - ➢ An invertible linear transformation applied to the four bytes in each column
    - ▪ If the inputs to the transformation differ in b bytes ($b > 0$) ⇒ outputs of the transformation differ in at least $5 - b$ bytes

❑ Stage III + stage IV ≈ mixing permutation of an SPN

And once we have done the shifting of the row bytes, we do actually a mixing of the columns, where an invertible linear transformation is applied to the 4 bytes which are now available in the individual columns. The property of this linear transformation is that if there are two block inputs x and x', such that the 4 bytes of x and x' in a column differs in b bytes where $b > 0$, then the output of this transformation for x and x' will now result in a new column which will differ in $5 - b$ byte positions. And on a very high level, this shifting of the rows and the mixing of the columns gives you the effect of the mixing permutation of an SPN.

(Refer Slide Time: 39:43)



Overview of AES

❑ In each round, the state is modified in four stages as follows :

- ❖ Stage I --- AddRoundKey
- ❖ Stage II --- SubBytes
- ❖ Stage III --- ShiftRows:
  - ➢ Bytes in the rows are left-shifted in a cyclic fashion:
    - ▪ 1^{st} row: no shift
    - ▪ 2^{nd} row: shift by 1 position
    - ▪ 3^{rd} row: shift by 2 positions
    - ▪ 4^{th} row: shift by 3 positions
- ❖ Stage IV --- MixColumns :
  - ➢ An invertible linear transformation applied to the four bytes in each column
    - ▪ If the inputs to the transformation differ in b bytes ($b > 0$) $\Rightarrow$ outputs of the transformation differ in at least $5 - b$ bytes

❑ Stage III + stage IV ≈ mixing permutation of an SPN

Security of AES : no practical attack reported , other than brute-force over the key space

So, I am not going into the internal details of what exactly is this linear transformation, how exactly it operates, why this rows are shifted in a left cyclic fashion and so on. Those details you can find out from any references which are cited in the book by Katz-Lindell. What I can say is that as far as the security of AES is concerned, no practical attack has been reported ever since is designed other than the brute force attack over the key space.

And that makes AES a very good candidate for instantiating a pseudo random permutations in any cryptography constructions based on pseudo random permutation. So, that brings me to the end of this lecture. Just to summarize, in this lecture we had seen the overview of 2 of the highly used and highly popular block ciphers namely DES and AES. And again, I would like to stress that they are not provably secure because there is no mathematical proof which shows that this construction satisfies the indistinguishability based definition of pseudo random permutations.

They just provide you heuristic security in the sense that ever since they are designed no practical attacks have been reported other than the brute force attack on the key space. So that is why we believe they indeed behave like or they give you a good instantiation of pseudo random permutations. Whereas the provably secure constructions based on one way functions, they are mathematically secure. They satisfies the definition of indistinguishability based definition.

The downside is that they cannot be used practically because the running time is enormous compared to the practical instantiations like DES and AES. Thank you.