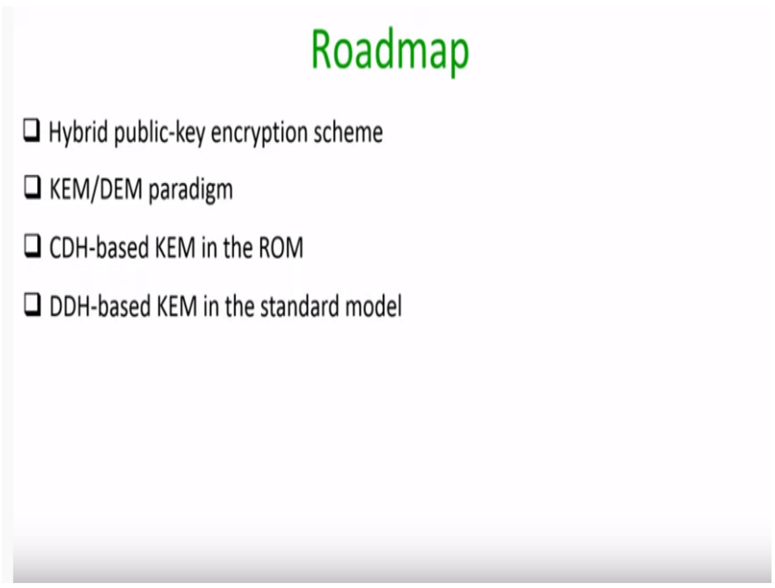
Foundations of Cryptography

Dr. Ashish Choudhury
Department of Computer Science
International Institute of Information Technology – Bangalore

Lecture – 47
Hybrid Public Key Cryptosystem

Welcome to this lecture. The plan for this lecture is as follows.

(Refer Slide Time: 00:36)

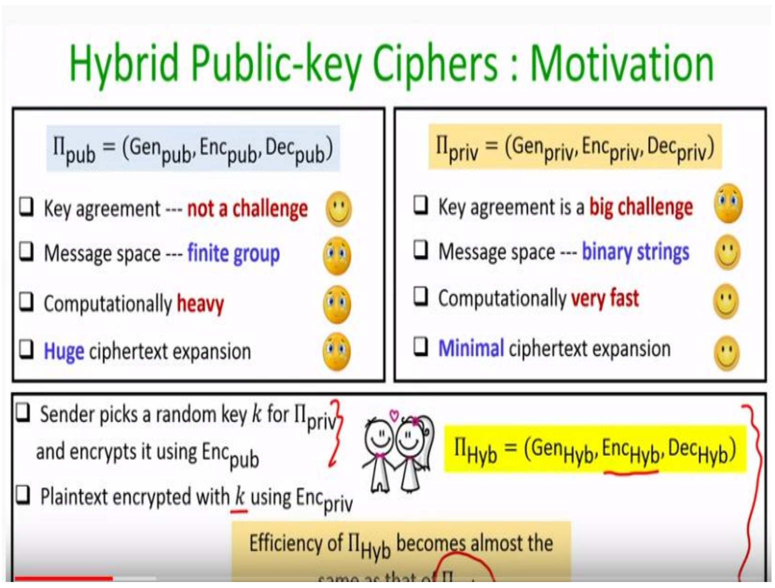


Roadmap

- ❑ Hybrid public-key encryption scheme
- ❑ KEM/DEM paradigm
- ❑ CDH-based KEM in the ROM
- ❑ DDH-based KEM in the standard model

In this lecture, we will introduce Hybrid public-key encryption scheme and we will introduce KEM/DEM paradigm, and we will see an instantiation of CDH-based KEM in the Random-oracle model and DDH-based instantiation of KEM in the standard model.

(Refer Slide Time: 00:49)



Hybrid Public-key Ciphers : Motivation

$\Pi_{pub} = (Gen_{pub}, Enc_{pub}, Dec_{pub})$

- ❑ Key agreement --- not a challenge
- ❑ Message space --- finite group
- ❑ Computationally heavy
- ❑ Huge ciphertext expansion

$\Pi_{priv} = (Gen_{priv}, Enc_{priv}, Dec_{priv})$

- ❑ Key agreement is a big challenge
- ❑ Message space --- binary strings
- ❑ Computationally very fast
- ❑ Minimal ciphertext expansion

- ❑ Sender picks a random key k for Π_{priv} and encrypts it using Enc_{pub}
- ❑ Plaintext encrypted with $\underline{k}$ using Enc_{priv}

$\Pi_{Hyb} = (Gen_{Hyb}, \underline{Enc_{Hyb}}, Dec_{Hyb})$

Efficiency of Π_{Hyb} becomes almost the same as that of Π

So, let's start with the motivation of Hybrid Public-Key Ciphers and before doing that, let us compare the public-key encryption scheme with a secret-key encryption scheme. Imagine, we are given a public-key encryption scheme, Π_{pub}, and we are given a symmetric encryption scheme, Π_{priv}, then if we consider the public-key encryption scheme, the key agreement is not at all a challenge.

That is what the whole motivation for the birth the public encryption process. If I am a receiver, it is suffice for me just to announce my public-key in the public domain. Anyone who wants to encrypt a message and send it to me, it can pick my public key, encrypt the message and send it to me. On the other hand, we had seen that key agreement is the biggest challenge in the symmetric key domain.

Until and unless the common key is established between the sender and the receiver, we cannot use any of the symmetric key primitives. If we consider the public-key encryption scheme, the downside there is the message space has to be a finite group or a finite algebraic structure, which most cases is the group, which is kind of a restriction because, in practice, the message space should be a set of bit string.

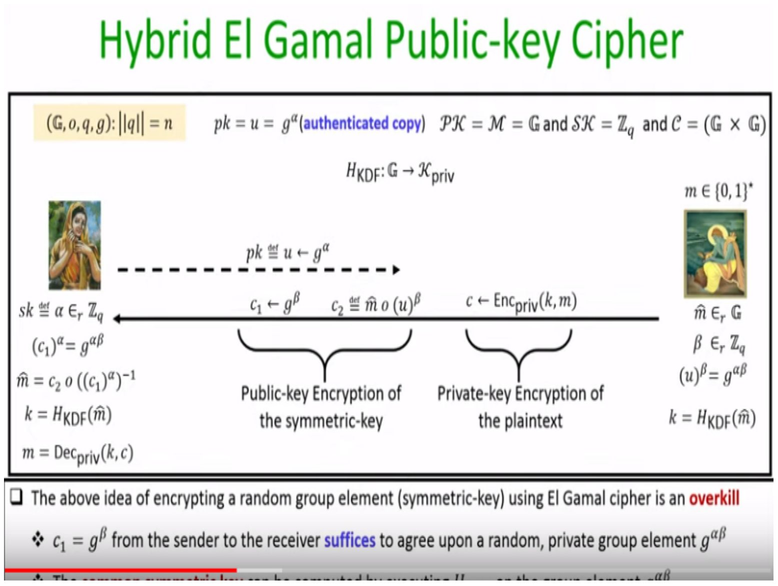
On the other hand, the message space in the symmetric keyword are binary strings. We do not require any sophisticated algebraic structure from the underlying message space for the overall security of my symmetric key encryption schemes. If we consider public-key encryption schemes, they are computationally very heavy because we have to perform modular exponentiations for instance in RSA function and in El Gamal encryption scheme.

Performing modular exponentiations is very computationally expensive and heavy compared to the encryption process, decryption process that we use in symmetric key encryption scheme, which are superfast and performs operations which are of XOR operations. In the same way, the public-key encryption scheme, the cipher text expansion is huge, for instance, if you see the El Gamal encryption scheme, the plain text that you want can encrypt is a single group element, but the cipher text consists of two groups elements and padded RSA, the amount of message, which you end up encrypting is very short compared to the amount of randomness that you are actually padding. On the other hand, in the symmetric key world, cipher text expansion can be as minimal as possible using any of the so called secure modes of operation of block ciphers.

So, now we can see have both pros as well as cons of public-key encryption scheme, symmetric encryption schemes, and what we can do is, we can think of somehow combining these two processes in a generic fashion and obtain a hybrid kind of encryption process where in the hybrid encryption process, which we call as Enc_{Hyb}. The sender picks a random key for the symmetric key encryption scheme and encrypts it using the public-key encryption process, namely it encrypts the random key k, which it has selected and encrypts it using the public key of the receiver and once the plain text is available with the sender, the actual encryption of the plain text happens using a symmetric key encryption process using the random key, which has been picked by the sender. If we do this, what is happening here is basically the entire efficiency of the hybrid encryption process becomes almost that of the symmetric key encryption process because the actual message which we are encrypting gets encrypted using a symmetric key encryption process.

The extra pay load that we are paying here is to encrypt the symmetric key which is used for encrypting the actual plain text using a public key encryption process. However in principle syntactically this whole hybrid process is still will be considered as an instantiation of public key encryption process because the key for the symmetric key encryption which the sender is using is getting encrypted by a public key encryption process. So, that is what is the overall motivation for designing Hybrid Public-Key Ciphers.

(Refer Slide Time: 5:01)



Hybrid El Gamal Public-key Cipher

$(\mathbb{G}, o, q, g): ||q|| = n$ $pk = u = g^{\alpha}$(authenticated copy) $\mathcal{PK} = \mathcal{M} = \mathbb{G}$ and $\mathcal{SK} = \mathbb{Z}_q$ and $\mathcal{C} = (\mathbb{G} \times \mathbb{G})$

$H_{\text{KDF}}: \mathbb{G} \rightarrow \mathcal{K}_{\text{priv}}$

$m \in \{0,1\}^*$

$pk \overset{\text{def}}{=} u \leftarrow g^{\alpha}$

$c_1 \leftarrow g^{\beta}$ $c_2 \overset{\text{def}}{=} \hat{m} \, o \, (u)^{\beta}$ $c \leftarrow \text{Enc}_{\text{priv}}(k, m)$

Public-key Encryption of the symmetric-key
Private-key Encryption of the plaintext

$sk \overset{\text{def}}{=} \alpha \in_r \mathbb{Z}_q$

$(c_1)^{\alpha} = g^{\alpha\beta}$

$\hat{m} = c_2 \, o \, ((c_1)^{\alpha})^{-1}$

$k = H_{\text{KDF}}(\hat{m})$

$m = \text{Dec}_{\text{priv}}(k, c)$

$\hat{m} \in_r \mathbb{G}$

$\beta \in_r \mathbb{Z}_q$

$(u)^{\beta} = g^{\alpha\beta}$

$k = H_{\text{KDF}}(\hat{m})$

- The above idea of encrypting a random group element (symmetric-key) using El Gamal cipher is an overkill
  - $c_1 = g^{\beta}$ from the sender to the receiver suffices to agree upon a random, private group element $g^{\alpha\beta}$

To make my point more clear, let us consider an instantiation of Hybrid El Gamal public-key Cipher. So, let me recall the syntax of Hybrid El Gamal public-key cipher, say Seetha wants to say to set up for her public parameter, her public key and secret key, so the public description, which is available to everyone is the description of the cyclic group, the generator, and the size of the group.

To do the key setup, what Seetha does is, you can imagine that she does her part of Diffie-Hellman key exchange protocol once for all for every potential Ram. For every potential sender who wants to encrypt a message and send to Seetha. So, she picks her random α from the set Z_q namely 0 to q-1, and that is a secret key and a public key, which is g^{α}, which is available in the public domain, and this if you can imagine as Seetha's contribution for the overall Diffie-Hellman key, which she wants to establish with every any potential sender in this world. Now imagine there is a sender, which has a plain text m, which it wants to encrypt and now this plain text is a bit string, it need not be an element of group. So, unlike the El Gamal encryption process where the actual plain text, which sender could encrypt and send to Seetha should be an element of group, now the plain text is a bit string.

Now, to encrypt this plain text m, what the sender does is, it picks a random element from the group which are noted by $\hat{m}$ and it encrypts this message $\hat{m}$ using the El Gamal encryption process. So, it computes the cipher text c_1, c_2 where c_1 can be interpreted as sender's contribution for the Diffie-Hellman key exchange protocol, namely g^{β}, where β is a randomly picked, and c_2 is the actual encryption of this random $\hat{m}$ using the common Diffie-Hellman key $g^{\alpha\beta}$.

Now, that is not the encryption of the message, so till now what sender has done basically the c_1, c_2, that sender has sent to Seetha is an encryption of the random $\hat{m}$, but the goal of sender is to encrypt the plain text m, so what we do here is we assume that apart from the description of the cyclic group, generator, group order and so on, we assume that we have a description of a key derivation function H publicly available and assume that the key derivation function maps the elements from the group g to the key space of the symmetric key encryption scheme, that sender is now going to use. So now, what sender is going to do is, sender knows that by sending c_1, c_2, to Seetha. Sender knows that Seetha also will end up getting the common key $g^{\alpha\beta}$ and assuming the DDH assumption is true in the underlying group, this key

$g^{\alpha\beta}$ is going to be a random element from the group from the view point of a computationally bounded adversary.

So what sender can now do is, it can derive a bit string key for the symmetric encryption process by applying a key derivation function to this random element $\hat{m}$ and that key is used to now encrypt the plain text m, which is a bit string and that is a actual encryption of the plain text m, which sender wants to encrypt. The decryption at Seetha's end happens as follows.

So for decryption, Seetha also needs to recover the same $\hat{m}$, which sender has used to derive the symmetric key k and $\hat{m}$ can be obtained by decrypting c_1, c_2 as per the syntax of El Gamal encryption scheme and once $\hat{m}$ is recovered by Seetha, what Seetha can do, she can also apply the same publicly available key derivation function on $\hat{m}$ and once $\hat{m}$ is recovered, she can decrypt the c component of the cipher text as per the decryption algorithm of the symmetric key encryption process and recover m.

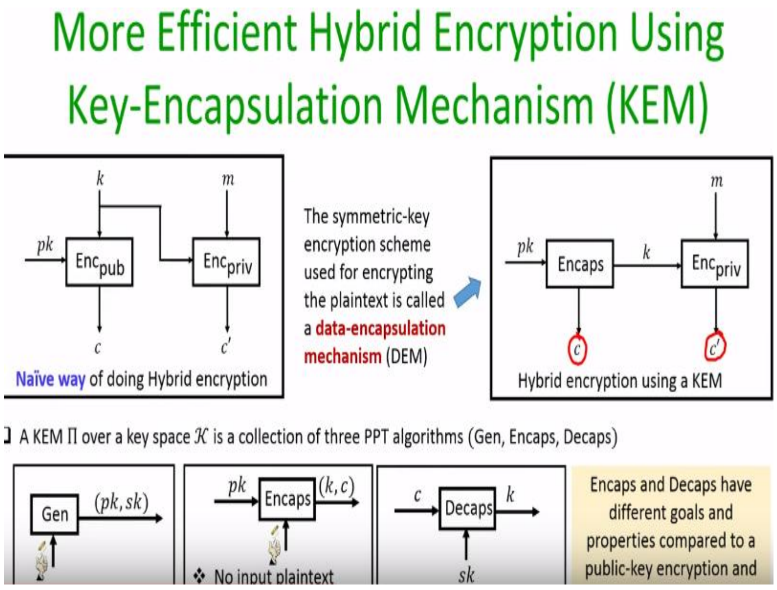
So, that is how you can interpret a hybrid version of El Gamal public-key cipher. So pictorially what is happening is here that as I said earlier c_1, c_2 is the public key encryption of the symmetric key, whereas the c component here is the actual private key encryption of the plain text. However, if you pause here for a moment, the above idea of encrypting the random group element by sender and deriving a key from it is an overkill.

So, what is happening here is sender is using a random group element $\hat{m}$, encrypting it as a whole using El Gamal encryption process, and then deriving that $\hat{m}$ a symmetric key. So that is an overkill here. A close observation tells you that it is suffice for the sender to just send g^{β}. It need not have to send c_2. It is suffice for the sender to just send g^{β}.

Because the sender knows that if it sends g^{β}, which can be reviewed as if it sender is sending his part of the Diffie-Hellman key exchange protocol message, then it knows that by sending g^{β}, both Seetha and Ram will end up agreeing to $g^{\alpha\beta}$ and assuming DDH assumption is true in the underlying group, we know that $g^{\alpha\beta}$ will be random in the view point of the adversary.

Hence the key k can be derived from the $g^{\alpha\beta}$ instead of deriving the key from the random element $\hat{m}$, because if we do this instead of using the approach that we have seen till now, we do a saving here. We need not have to pick the random element $\hat{m}$ and we do not need to encrypt $\hat{m}$, hence we do not need to send c_2, and overall size of the cipher text will get significantly reduced and that will also lead to saving in the bandwidth as well, because if we do not need send the c_2 that means we do not need to send an extra group element to encrypt the message.

(Refer Slide Time: 11:22)



More Efficient Hybrid Encryption Using Key-Encapsulation Mechanism (KEM)

The symmetric-key encryption scheme used for encrypting the plaintext is called a **data-encapsulation mechanism** (DEM)

- A KEM Π over a key space $\mathcal{K}$ is a collection of three PPT algorithms (Gen, Encaps, Decaps)

Encaps and Decaps have different goals and properties compared to a public-key encryption and

k
m
pk
Enc_{pub}
Enc_{priv}
c
c'
Naïve way of doing Hybrid encryption
m
pk
Encaps
k
Enc_{priv}
c
c'
Hybrid encryption using a KEM
Gen
(pk, sk)
pk
Encaps
(k, c)
❖ No input plaintext
c
Decaps
k
sk

Even though we have seen an instantiation of Hybrid El Gamal cipher here, it turns out that this approach is an overkill. So what we are going to do now, as we are now motivated by the discussion here that, it suffices for the sender here to just send g^{β} and derive a key from g^{β}, what we are going to do is we are now going to derive a new primitive, which we call as Key-Encapsulation mechanism or KEM and then we will see that how we can get more efficient Hybrid encryption using this KEM.

So, to make my point clear the naïve way of Hybrid encryption that we have seen now is just in the context of Hybrid El Gamal is as follows. So, at the encryption end what we were doing is the sender was picking a random key for the symmetric key and it was getting encrypted by the public key of the receiver to derive the cipher text c, which can be considered as an encryption of the symmetric key.

And actual encryption of the message was done using the key k and this was kind of a two-stage approach, where we were first choosing our random symmetric key and then using it for the public key encryption process. More efficient approach will be done using a primitive, which we are going to define it soon which we call as key encapsulation mechanism where both these two things are done in a single shot in a single stage.

And the advantage here is that not only it is conceptually simpler, but in many cases it is more efficient and to understand that how exactly KEM is going to be efficient compared to this two stage approach, you can see what we have seen just now in the context of Hybrid El Gamal. The naïve way of implementing El Gamal the sender was picking a random $\hat{m}$ deriving the key and $\hat{m}$ was whole encrypted using the El Gamal encryption process.

But later we argued that the sender need not have to do that. It can carry out Hybrid Encryption even by just sending g^{β} to the receiver. So, this key encapsulation mechanism or KEM over a key space is a collection of three algorithms. It will have a Key Generation algorithm, which will output a public key and a secret key, and it will have an encapsulation algorithm, which takes no plain text as an input.

It has an external input, namely the public key pk, but apart from that it does not take any plain text, and it has an internal randomness and it gives you two outputs. It outputs a key from the key space and it outputs an encapsulation of the key, which we call as c. The de-capsulation algorithm takes an encapsulated key, which I denote as c, and the secret key sk, and it de-capsulates the key, which is hidden there or encapsulated in the c end, and it gives you back the encapsulated key as the output namely key k.

So, I stress here that encapsulation algorithm and the de-capsulation algorithm have completely different goals and properties compared to encryption and decryption algorithm of a public key encryption process. As you can see here, the encapsulation algorithm does not take any plain text. Even without taking any plain text, just based on internal randomness, it gives you an output key k, and an encapsulation of that key k, which I denote as c.

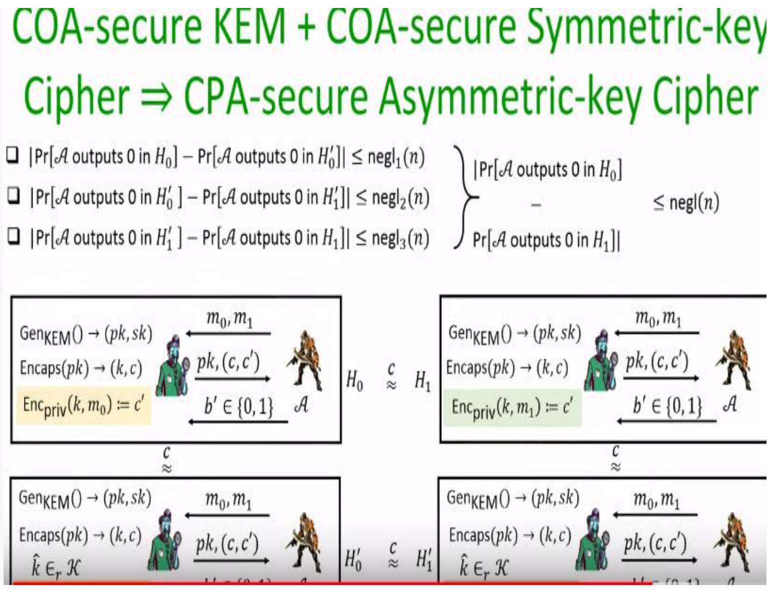
But that c should not be taken as an encryption of k and in the same way, the de-capsulation, it does not take a cipher text for decryption. It takes a c it has to de-capsulate and it has to take out the key k, which is hidden there in the c. So, it turns out that the naïve way of coming up with KEM is to just take any public-key encryption scheme and the encryption encapsulation algorithm can be as follows.

Internally, we generate a random group element and encrypt it as per the public key encryption process that can be the encapsulation process. In the same way, the analog is decapsulation process can be decrypt the group element that we have just encrypted using the public key encryption process, but it turns out that, we can have more efficient instantiation of encapsulation and de-capsulation.

Now, assume for the moment we have such an encapsulation and decapsulation mechanism, then instead of doing the naïve way of Hybrid Encryption, we can do one Hybrid encryption as follows. The sender can run the encapsulation algorithm using the description of the public key and it will obtain the encapsulation of the key and key k, and key k is used to encrypt the plain text using the symmetric key encryption process to obtain the cipher text c**'**.

And overall cipher text will be the encryption of the c**'** plain text along with the encapsulation of the key k, which is used to encrypt the plain text, but the key k will not be a part of this cipher text. So, the symmetric key encryption scheme, which we are now using in this modified way of doing Hybrid encryption is called as a data-encapsulation mechanism or DEM. So, now we can see we have two kind of encapsulation happening. One capsulation is for encapsulating the key and that will be denoted by c, and we will have the encapsulation of the actual data, which we will denote as c**'**.

(Refer Slide Time: 16:30)



COA-secure KEM + COA-secure Symmetric-key Cipher ⇒ CPA-secure Asymmetric-key Cipher

- ❑ $|\Pr[\mathcal{A} \text{ outputs 0 in } H_0] - \Pr[\mathcal{A} \text{ outputs 0 in } H_0']| \leq \mathsf{negl}_1(n)$
- ❑ $|\Pr[\mathcal{A} \text{ outputs 0 in } H_0'] - \Pr[\mathcal{A} \text{ outputs 0 in } H_1']| \leq \mathsf{negl}_2(n)$
- ❑ $|\Pr[\mathcal{A} \text{ outputs 0 in } H_1'] - \Pr[\mathcal{A} \text{ outputs 0 in } H_1]| \leq \mathsf{negl}_3(n)$

$|\Pr[\mathcal{A} \text{ outputs 0 in } H_0] - \Pr[\mathcal{A} \text{ outputs 0 in } H_1]| \leq \mathsf{negl}(n)$

$\mathsf{Gen}_{\mathsf{KEM}}() \to (pk, sk)$
$\mathsf{Encaps}(pk) \to (k, c)$
$\mathsf{Enc}_{\mathsf{priv}}(k, m_0) := c'$
m_0, m_1
$pk, (c, c')$
$b' \in \{0,1\}$
$\mathcal{A}$
$H_0 \overset{c}{\approx} H_1$
$\mathsf{Gen}_{\mathsf{KEM}}() \to (pk, sk)$
$\mathsf{Encaps}(pk) \to (k, c)$
$\mathsf{Enc}_{\mathsf{priv}}(k, m_1) := c'$
m_0, m_1
$pk, (c, c')$
$b' \in \{0,1\}$
$\mathcal{A}$
$\overset{c}{\approx}$
$\overset{c}{\approx}$
$\mathsf{Gen}_{\mathsf{KEM}}() \to (pk, sk)$
$\mathsf{Encaps}(pk) \to (k, c)$
$\hat{k} \in_r \mathcal{K}$
m_0, m_1
$pk, (c, c')$
$H_0' \overset{c}{\approx} H_1'$
$\mathsf{Gen}_{\mathsf{KEM}}() \to (pk, sk)$
$\mathsf{Encaps}(pk) \to (k, c)$
$\hat{k} \in_r \mathcal{K}$
m_0, m_1
$pk, (c, c')$

So, now let us define the COA and CPA security of key encapsulation mechanism, so imagine we have given a key encapsulation mechanism and when I say that it achieves COA security, which actually means CPA security in the public-key domain, then we require the following should be ensured. Imagine there is a receiver who has run the Key Generation algorithm of the $\hat{m}$ KEM.

And it has made the public key available and say there is a sender, which picks the public key, runs the encapsulation algorithm under the public key and obtains the output c, k and the encapsulation of the key c is sent to the receiver and say there is an eavesdropper who eavesdrops the encapsulation of the key, basically we require that even after knowing the description of the encapsulation process, de-capsulation process and by eavesdropping the c, the eavesdropper who is computationally bounded, from its view point the encapsulation c should be independent of the actual key which is encapsulated in c, and this is modeled by a COA experiment played between a computationally bounded adversary and a challenger and the challenger does the following. It runs the Key Generation algorithm, obtains pk, sk and runs the encapsulation algorithm and obtains the output c, k, and now it prepares the challenge as follows.

It tosses a fair coin. If the coin tosses 0, then it picks $\hat{k}$ randomly from the key space of the key encapsulation mechanism. On the other hand, if the coin toss is 1, then the $\hat{k}$ is said to be same k, which is encapsulated in c and now the challenge for the adversary is as follows. The adversary is given the public key, the encapsulation c and $\hat{k}$ and the challenge for the adversary is to identify whether $\hat{\mathrm{k}}$ is related to c or not, whether $\hat{k}$ is the same key, which is encapsulated in c namely b equal to 1 or whether $\hat{k}$ is a totally independent element independent of the key, which is encapsulated in c, namely b equal to zero.

This exactly models the requirement, which we wanted to capture or ensured by a secured key encapsulation mechanism. So the adversary outputs a b, namely it tells whether $\hat{k}$ is generated as per the method b=0 or whether b=1, and the rules of the game here is that we say that the adversary has won the game meaning that the output of the experiment is 1, if and only if b' equal to b, and we say that our key encapsulation mechanism is COA- secure if for every polytime adversary there exist a negligible functions such that the probability of the adversary winning the game is upper bounded by half plus negligible probability or equivalently saying that, the distinguishing advantage of the adversary is negligible, namely it does not matter whether $\hat{k}$ is generated as per method b = 0 or as method b=1, the output of the adversary should be almost the same except with negligible probability in both the cases.

The reason I am calling it COA security as well CPA security because since we are in the public key world, and the public key is explicitly given to the adversary, that the adversary need not have to get any encapsulation oracle service from the challenger. It can run the encapsulation algorithm using the public key pk on its own on any randomness, and generate any kind of c, k.

Our goal is to ensure that even if it has got encapsulation oracle service explicitly, still it should not be able to figure out whether $\hat{k}$ is related to c or not. Now, what we are going to discuss here is a very interesting result. We are going to show that, if you are given a key encapsulation mechanism, which is COA-secure, then combining it with any symmetric key cipher, which is COA-secure, you can obtain a CPA-secure Hybrid encryption process.

So, imagine you are given a key encapsulation mechanism, and you are given a symmetric key encryption process, and say we combine these two primitives to obtain a Hybrid encryption process as follows. So, the Key Generation algorithm of the Hybrid process basically runs the Key Generation algorithm or KEM, and outputs the public key and secret key. The encryption algorithm of the Hybrid process takes a plain text m and the public key pk, and internally it does the following.

It runs the encapsulation algorithm obtains k, c, and using the key, it runs the symmetric key encryption algorithm on the plain text m, and obtains the cipher text c', and the overall cipher text is the encapsulation of the key and the encapsulation of the data, namely c, c' and the decryption happens in the hybrid process analogously, namely if you have the secret key sk and a cipher text c, c'.

Then you first de-capsulate the encapsulation of the key, and obtain the encapsulated key k and then using the key k, you decapsulate the c' component of the cipher text, by running the decryption algorithm of the symmetric key encryption process and output the m as the plain text. What we are going to formally prove is that if your KEM is COA-secure and if your symmetric key encryption process is COA-secure.

Then this overall way of combining and obtaining a Hybrid encryption process is going to give you a public key CPA-secure Hybrid encryption process and this is quite surprising because what we are trying to prove here is that COA-security in the symmetric key world when combined with a COA-secure KEM gives you a CPA security in the public key domain and this is surprising because in the symmetric key world we have seen that COA security and CPS security are not equivalent.

The intuition behind this theorem is as follows. If you see the encryption process of this hybrid encryption scheme, every time I want to encrypt the same message m using the public key pk. I am going to use a different encapsulated key because my encapsulation algorithm is going to be COA-secure or CPA-secure. It will not produce the same k again and again, and since my encapsulated key is going to be different, that means the private key encryption which I am using inside this Hybrid encryption process is using a fresh key for each instance of the encryption process.

So that is the overall intuition behind the proof of this theorem, that means, to instantiate this framework, to instantiate the symmetric key part of encryption and the decryption, we can just use any stream cipher. So, let us try to prove this theorem, and before going into the proof of this theorem, let us first see how exactly the instantiation of the CPA game against this hybrid encryption process will look like.

Imagine, we have an adversary A who is polynomially bounded, then in the CPA game and COA game to be more specific, remember in the public key world, COA and CPA security are same, so the rules of the game will be as follows. The challenger will run the Key Generation algorithm namely the Key Generation algorithm of the encapsulation mechanism obtain the key pair pk, sk, the public key pk will be given to the adversary, which basically gives the adversary encapsulation oracle service on any input of its choice.

Adversary submits a pair of challenge plain text m_0, m_1. One of them is randomly encrypted, decided for encryption. To encrypt the challenge plain text, the challenger runs the encapsulation algorithm under the public key, obtain the encapsulated key k, and encapsulation of the key c, and then using the encapsulated key, it runs the symmetric key encryption algorithm to encrypt the challenge plain text m_b and obtain the cipher text c'.

And now the challenge cipher text for the adversary is the encapsulation of the key and the actual encryption of the message m_b. The challenge for the adversary is to identify whether it is seeing c*, which is encryption of m_0 or m_1, and it will submit its output and our goal is to show the following. Our goal is to show that for any polytime adversary participating in this experiment, the distinguishing advantage of this experiment is upper bounded by some negligible function, and we are going to prove that using Hybrid argument.

The reason we are going to use Hybrid argument is that we are now having two primitives which are involved here. We are using KEM as well as symmetric-key encryption process and hence we cannot directly reduce the overall security of the Hybrid encryption process to the security of any of this underlying primitives. Remember, our goal is to prove this claim, namely the distinguishing advantage of the attacker is upper bounded by some negligible function.

We now to prove this claim, what I do here is I consider two versions of this indistinguishability game, the CPA indistinguishability game depending upon whether the challenger has used message m_0 for encryption or whether the challenger has used message m_1 for encryption. So, we can imagine that this COA game against the Hybrid encryption process can be viewed as a combination of two experiments, which I denote as H_0 and H_1 and each of these experiments can occur with probability 1/2.

In the experiment H_0, it is basically a version of the COA game where the challenger has picked up the message m_0 for encryption and the experiment H_1 is the instantiation of the COA game where the challenger has picked the message m_1 for encryption, and since our goal is to prove this claim namely the distinguishing advantage of the attacker is negligible, our goal is to show that from the view point of the attacker H_0 and H_1, these two experiments are computationally indistinguishable, that means, with almost equal probability, adversary is going to output 0 in experiment H_0 as well as the experiment H_1. We have to prove that what we are going to do is, we are going to introduce two hybrid experiments here, which I denote as H_0' and H_1', and H_0' is almost the same as the experiment H_0. The only difference here is that the message m_0 is now encrypted using a completely random key from the keyspace of the symmetric key encryption process, instead of using the key k, which was encapsulated in the encapsulation c in the experiment H_0. In the same way, if I consider the experiment H_1', it is different from experiment H_1 in the sense, that here a totally independent random key is used for encrypting the message m_1, compared to the experiment H_1 where the key k, which was used to encrypt the message m_1, was encapsulated in the encapsulation c.

So, these are the differences in the experiment H_0' and H_1'. Now, my claim is that experiment H_0 and experiment H_0', they are computationally indistinguishable from the view point of any computationally bounded adversary, namely I claim here that if I take the absolute difference between the probability of the adversary a outputting 0 in the experiment H_0, and outputting 0 in the experiment H_0', it is upper bounded by negligible probability and this is because of COA security of the underlying key encapsulation mechanism.

Namely, I can formally prove that if my underlying key encapsulation mechanism is COA-secure, then it does not matter whether my challenger uses totally independent key from the key space of the symmetric key encryption scheme for encrypting the message or whether it is using a key, which is generated by running a key encapsulation KEM. From the view point of the adversary, that c' could be an encryption produced by both kind of case.

It cannot distinguish a part whether it is seeing c' of type H_0 or whether it is seeing c' of type H_0'. We can formally prove that by reducing the security to the COA security of the underlying KEM. I am not going into the exact proof of that. I will leave those details for you as an assignment. Now, if I consider the experiments, H_0' and H_1', I can say that they are computationally indistinguishable from the view point of any polytime adversary and this follows from the COA -security of the underlying symmetric-key cipher.

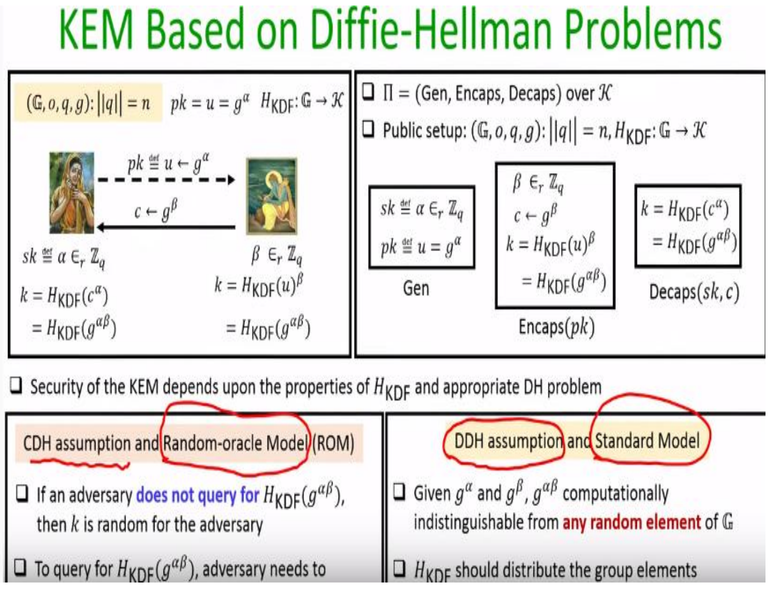
Namely, if my underlying symmetric-key cipher is COA-secure, then it does not matter whether the adversary is seeing an encryption of m_0 or whether the adversary is seeing an encryption of m_1, it cannot distinguish apart. The rest of the information that adversary is seeing namely, pk, c, c' and so on, their distribution are exactly same in the experiment H_0' and H_1'.

So, basically I can say that this experiment H_0' and H_1', they are basically the two versions of the COA indistinguishability game for the underlying Symmetric-key Cipher and if my underlying COA Symmetric-key Cipher is COA-secure, then I know that from the view point of this computationally bounded adversary A, the experiment H_0' and H_1' are same.

Finally using the same argument, which I used to show the computational indistinguishability of H_0, and H_0', I can say that the experiment H_1 and H_1', they are computationally indistinguishable because it follows from the COA security of the underlying KEM. Now, by using the triangle inequality, I can end up showing that if I sum these three inequalities, I end up showing that if I take the difference of the adversary's advantage or adversary's probability of outputting 0 in H_0 and H_1, then that is upper bounded by the summation of three negligible probabilities.

We know that by closure properties of negligible function, summation of any three negligible function in the security parameter is also a negligible function, which ends up showing that experiments H_0 and H_1, they are computationally indistinguishable, which proves that we now have a generic construction of CPA-secure Asymmetric-key Cipher.

(Refer Slide Time: 31:47)



KEM Based on Diffie-Hellman Problems

$(\mathbb{G}, o, q, g): ||q|| = n \quad pk = u = g^{\alpha} \quad H_{KDF}: \mathbb{G} \rightarrow \mathcal{K}$

$pk \overset{\text{def}}{=} u \leftarrow g^{\alpha}$

$c \leftarrow g^{\beta}$

$sk \overset{\text{def}}{=} \alpha \in_r \mathbb{Z}_q$

$k = H_{KDF}(c^{\alpha}) = H_{KDF}(g^{\alpha\beta})$

$\beta \in_r \mathbb{Z}_q$

$k = H_{KDF}(u)^{\beta} = H_{KDF}(g^{\alpha\beta})$

- ❑ Π = (Gen, Encaps, Decaps) over $\mathcal{K}$
- ❑ Public setup: $(\mathbb{G}, o, q, g): ||q|| = n, H_{KDF}: \mathbb{G} \rightarrow \mathcal{K}$

$\boxed{sk \overset{\text{def}}{=} \alpha \in_r \mathbb{Z}_q \quad pk \overset{\text{def}}{=} u = g^{\alpha}}$

Gen

$\boxed{\beta \in_r \mathbb{Z}_q \quad c \leftarrow g^{\beta} \quad k = H_{KDF}(u)^{\beta} = H_{KDF}(g^{\alpha\beta})}$

Encaps(pk)

$\boxed{k = H_{KDF}(c^{\alpha}) = H_{KDF}(g^{\alpha\beta})}$

Decaps(sk, c)

- ❑ Security of the KEM depends upon the properties of H_{KDF} and appropriate DH problem

CDH assumption and Random-oracle Model (ROM)

- ❑ If an adversary does not query for $H_{KDF}(g^{\alpha\beta})$, then k is random for the adversary
- ❑ To query for $H_{KDF}(g^{\alpha\beta})$, adversary needs to

DDH assumption and Standard Model

- ❑ Given g^{α} and g^{β}, $g^{\alpha\beta}$ computationally indistinguishable from any random element of $\mathbb{G}$
- ❑ H_{KDF} should distribute the group elements

So, now let's see an instantiation of KEM based on Diffie-Hellman problems, so the public setup here is the description of generator of a cyclic group, group order and so on and apart from that, we have the description of our key derivation function mapping the group elements to the key space of the underlying symmetric-key encryption process. To do the key setup, basically receiver runs her part of the Diffie-Hellman key exchange protocol.

So she picks a random α as a secret key and a public key is g^{α} and now to do the key encapsulation, basically any sender has to do the following. It will run his part of the Diffie-Hellman key exchange protocol namely pick a random β and compute g^{β} and now map the element $g^{\alpha\beta}$ to an element of the key space of the symmetric encryption scheme by running the key derivation function, so that will be the encapsulation of the key.

The decapsulation happens as follows. What the receiver does is, it runs her part of the Diffie-Hellman key exchange protocol to compute the agreed upon key, namely $g^{\alpha\beta}$ and apply the key derivation function on that derived group element to obtain back the key, which was encapsulated in c. This is basically the same thing, which we have discussed during the instantiation of our Hybrid El Gamal encryption process.

Now, it turns out that security of this KEM depends upon the underlying properties of the key derivation function and appropriate DH problem. If we are in the Random-oracle model, namely if we assume that this key-derivation function is modeled as Random-oracle, and if you assume that the CDH problem is difficult to solve, then we can prove that this KEM is CPA-secure.

And the intuition for that is if the adversary has not queried for the value of the Random-Oracle on the input $g^{\alpha\beta}$, then the resultant output k is random from the view point of the adversary because that is what is the property of Random-oracle. Now, what is the chance that the adversary would have queried for $g^{\alpha\beta}$, given that it only knows the public key g^{α} and encapsulation of the key namely g^{β}.

Well, that is precisely the probability, which it can solve an instance of the CDH problem and that is why if we assume that the CDH problem is difficult to solve, this construction is secure in the Random-Oracle model. On the other hand, if you want to make a stronger assumption, namely if you are assuming that the DDH problem is difficult to solve in your group, then we do not require this key derivation function to be modeled as a Random-oracle.

We can just model special type of function. This is because the DDH assumption says that even if there is a polytime adversary, which has seen the public key g^{α} and encapsulation of g^{β}, the resultant $g^{\alpha\beta}$ could be any random element from the view point of the computationally bounded adversary. Now, it suffice for the underlying key derivation function to be a special type of function, which distributes the group element almost evenly among the elements of the key space of the symmetric key encryption process.

What I mean by that is, if I consider if this is my group g, and if this is my set $\mathcal{K}$, and the number of elements, which map to a candidate element from this key space, the number of group elements which map to this one candidate element from this key space should almost be the same. There should not be any bias in the distribution or in the mapping by which this function H is mapping the elements of the group to the elements of the key space of my symmetric key encryption process.

If I assume that my underlying key derivation function has that kind of smoothing effect, then it is suffice for me to just remain in the standard model and claim the security of this key encapsulation mechanism, just based on the DDH assumption. Now, you can see the importance of assumptions that we are making throughout this course. Same construction here can be proved secure in with weaker hardness assumptions, namely CDH assumption. But with a more powerful properties from the underlying hash function, namely assuming it is acting as a Random-oracle whereas if you do not want to model your underlying hash function as a Random-oracle, you want to model it as a special type of smooth distributing function, then you need to have a more powerful hardness assumption in your underlying group, namely you need to have the DDH problem difficult to be solved in your underlying group.

So, that brings me to the end of this lecture. Just to summarize in this lecture, we have introduced Hybrid encryption process and we have discussed the CPA-security of Hybrid encryption process. We also saw a generic construction of how to combine a CPA-secure or a COA-secure key encapsulation mechanism along with any COA-secure symmetric key encryption process and get an overall CPA-secure Hybrid encryption process. Thank you.