Foundations of Cryptography
Prof. Ashish Choudhury
Department of Computer Science
International Institute of Information Technology - Bangalore

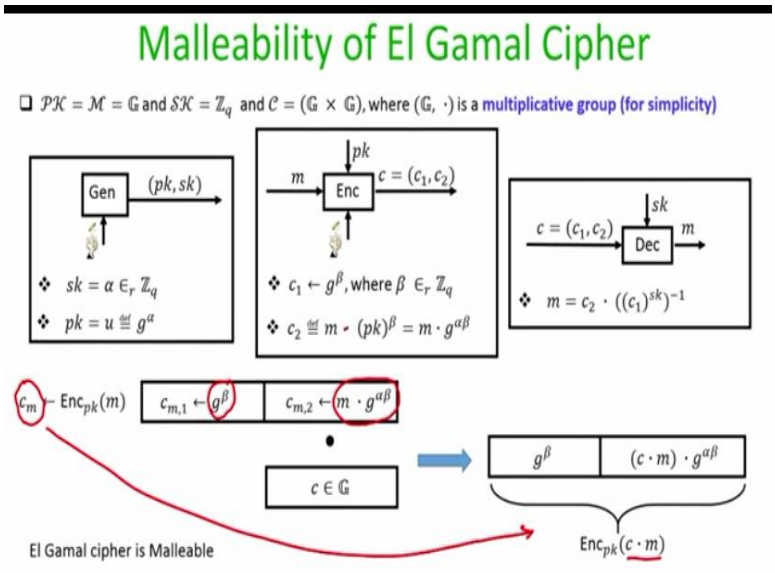
Lecture - 49
CCA Secure Public Key Ciphers Based on Diffie-Hellman Problems

(Refer Slide Time: 00:33)

Hello everyone. Welcome to this lecture. The plan for this lecture is as follows: So in this lecture, we will see the CCA insecurity of El Gamal public key cryptosystem, which we had proved earlier to be CPA secure. Then, we will see an instantiation of CCA secured key encapsulation mechanism based on gap-CDH assumptions. So we will formally introduce what exactly is gap-CDH assumption.

And based on that, we will see a CCA secured instantiation of key encapsulation mechanism and since in the last lecture, we had seen that, if you are given a CCA secured KEM, then combining it with any CCA secure symmetric key cipher, we can obtain a CCA secure hybrid public key encryption scheme. So by doing that, we will obtain a CCA secure version of instantiation of hybrid public key cryptosystem based on Diffie-Hellman problems and the two variants of such an instantiation, namely DHIES and ECIES, we are going to discuss them.

(Refer Slide Time: 01:26)



So let us start with the malleability of El Gamal Cipher. So just to recall, this is a description of El Gamal encryption process. The key generation algorithm outputs a public key and secret key, where public key is some g^{α}, where α is randomly chosen from Z_q and a secret key. To encrypt a plain text, sender first computes g^{β}, where β is randomly chosen from Z_q.

And then the message is multiplied by public key $(pk)^{\beta}$ and $(pk)^{\beta}$ is nothing but a Diffie-Hellman key $g^{\alpha\beta}$. So for simplicity, I am assuming that my underlying group is a multiplicative group and that is why my c_2 is the product of m and the component $(pk)^{\beta}$. To decrypt, the receiver first computes a common Diffie-Hellman key by raising the c_1 part of the cipher text to the secret key and taking its multiplicative inverse and multiply that with the c_2 component of the cipher text.

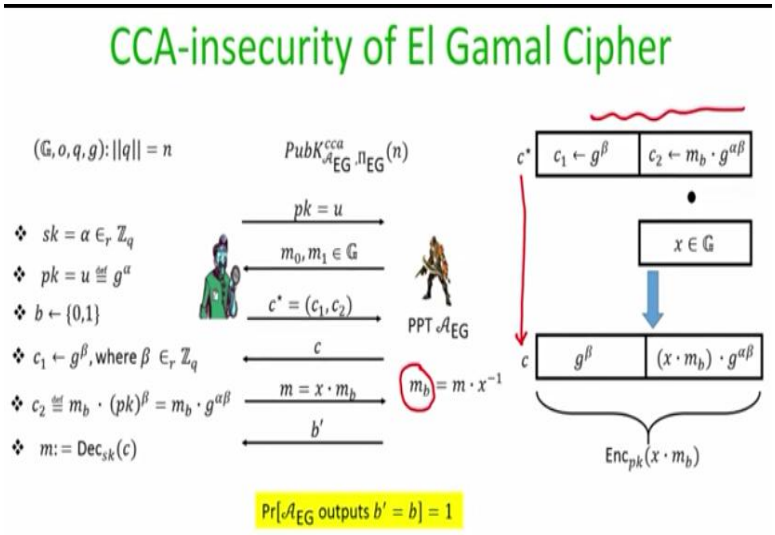
So that the effect of $g^{\alpha\beta}$ cancels out. And we had proved that if DDH assumption holds, then this is CPA secure. So recall that, we also discussed in one of our earlier lectures the homomorphic property of the El Gamal ciphers. So El Gamal ciphers, so what exactly we mean is that, suppose you are given encryption c_m of some unknown message m, then as per the syntax of the El Gamal cipher c_m will consist of two group elements.

The first group element will be some g^{β} (uknonwn β) and the second component will be the unknown plain text m multiplied by $g^{\alpha\beta}$. Now if we take the cipher text and take the second component of the cipher text and multiply it by any value c from the group. Then what we obtain

is basically a new cipher text consisting of two group elements, which can be considered as valid El Gamal encryption of the message c times m.

This means, my El Gamal encryption process is malleable, because just by taking a cipher text of an unknown message m, I can produce a cipher text of a related message namely c times m where c is known to me, but the m is not known to me, by just performing some local operations and it turns out that, if my encryption process is malleable, then we can never hope that it can be CCA secure and I am going to demonstrate that with an example here.

(Refer Slide Time: 04:05)



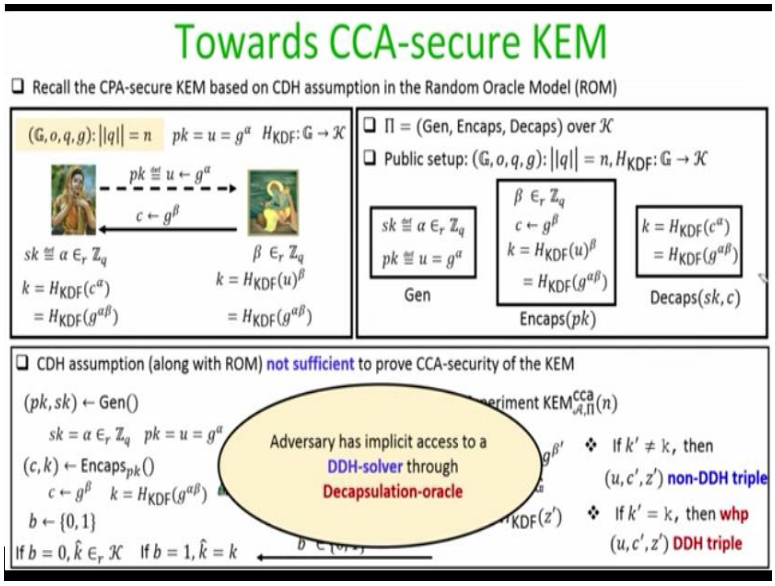
So imagine, there is an arbitrary adversary polytime, which participates in an instance of CCA game against the El Gamal cipher. So as per the rules of the game, the challenger will throw the public key to the adversary, where the public key u is g^{α}, where α is not known. What the adversary does is, it takes an arbitrary pair of plain text from the group. The challenge is prepared by the challenger by picking either m_0 or m_1 for encryption.

And it submits a challenge cipher text c* to the adversary and what adversary now does is, it knows that the challenge cipher text, which consists of two group elements has this structure : $c_1 \leftarrow g^{\beta}$ and $c_2 \leftarrow m_b \cdot g^{\alpha\beta}$, where β is not known to it, the actual m_b is not known to it and $g^{\alpha\beta}$ is not known to it, but what adversary can do is, it can pick some arbitrary x from the group and it can exploit malleability property of the El Gamal cipher text.

And it can now produce some modified cipher text denoted as c, which is now an encryption of the message x times the unknown message m_b and what the adversary now does is, it asks for the decryption oracle service for this modified cipher text c, which is allowed as per the rules of the CCA game. Now the challenger has to decrypt this modified cipher text c and on decrypting it, will throw back x * m_b to the adversary.

Now since the adversary knows the value of x and x belongs to the group, it can compute the multiplicative inverse of the value x and on multiplying it with m, it can completely identify what m_b and hence, it can submit the right b and as you can see that the probability that this adversarial strategy wins the game, CCA game is exactly 1. That means, my El Gamal cipher is not CCA secure. It is only CPA secure. As soon as we encrypt some message using El Gamal cipher and if we are in the malicious adversarial model using the help of the decryption oracle service, adversary can completely find out what exactly is encrypted.

(Refer Slide Time: 06:20)



So that means, now we have to think that how we can design a CCA secure instantiation of El Gamal cipher, so what we are going to do is instead of seeing a CCA secure version of El Gamal cipher, we are going to see a CCA secure variant of hybrid El Gamal cipher and for that what we are going to do is, we are going to instantiate a CCA secure KEM based on Diffie-Hellman problems.

So recall the CPA secure key encapsulation mechanism based on the CDH assumption in the random oracle model, that we had discussed in one of the earlier lectures. So imagine Sita wants to set up the key for her key encapsulation mechanism. So what she does is, she does her part of the Diffie-Hellman key exchange protocol once for all by picking a random α, which is set as her secret key and making g^{α} as her public key.

This can be considered, this message g^{α} can be considered as a message once for all for every potential sender who wants to communicate to her and now if there is a Ram who wants to encapsulate key, then the encapsulation algorithm for Ram will be as follows. Ram will do his part of the Diffie-Hellman key exchange protocol by picking a random β and computing g^{β}, that is the encapsulation. And the key, which Ram obtains or which is considered to be encapsulated in this encapsulation c is basically the output of $(pk)^{\beta}$ evaluated under the key derivation function, which is publicly available, where the underlying key derivation function maps element from the cyclic group g, to the key space of some underlying symmetric key encryption scheme.

To decapsulate c, what Sita has to do is, she has to do her part of the Diffie-Hellman key exchange protocol to obtain the common key $g^{\alpha\beta}$ and obtain the output of the key derivation function on that $g^{\alpha\beta}$ and then she can obtain back the same key k. We had proved that in the random oracle model, if we assume the CDH problem is difficult to solve in the underlying group, then the above key encapsulation mechanism is CPA secured.

But it turns out that the same key encapsulation mechanism is not CCA secure, even if we are in the random oracle model. This is because if we assume that there is a polytime adversary, who participates in an instance of CCA game against the above key encapsulation mechanism, then the game will look something as follows. The challenger will generate the secret key and a public key and using the public key, it will perform an encapsulation to obtain the encapsulation c and encapsulated key k.

And it will prepare the challenge by throwing, by tossing a fair coin and the challenge for the adversary will be u, c, and $\hat{k}$ and the goal of the adversary is to identify whether $\hat{k}$ is generated as

per the method b equal to 0 or as per the method b equal to 1. But since we are in the CCA world, the adversary is now allowed to have access to the decapsulation oracle service.

Namely it can submit any encapsulation c', which is different from c and c the decapsulation of the resultant c'. So what the adversary can now do is, it can arbitrarily pick some element c' from the group and say it is some $g^{\beta'}$ and along with that, it picks some random element z' from the group and not only that, it also computes the output of the key derivation function on that randomly chosen element z' to obtain the key k'.

Now since the adversary is allowed to have the decapsulation oracle service, it can ask for the decapsulation of c'. Since c' is randomly chosen from the group, very likely c' is going to be different from c. Even if it is not different from c, what adversary can do is, it can keep picking random c' and very likely c' will be different from c as soon as it obtains the c' different from c, it asks for the decapsulation of c.

And in response the challenger will output this k, and if you see the syntax of the decapsulation algorithm, this value k is nothing but the output of the key derivation function on the input $g^{\alpha\beta'}$, because if c' is $g^{\beta'}$, then to decapsulate c', the challenger is going to first compute $g^{\alpha\beta'}$ and then map that element $g^{\alpha\beta'}$ to an element of the key space by executing, by computing the output of key derivation function.

Now what the adversary can now do is, it can ask for the decapsulation oracle service for many such c', namely it can perform the same computation that he has done here, polynomial number of time, picking some random c', random z', computing the k' corresponding to picked z' and asking for the decapsulation service of the c' component and now once it is polynomially trained, it can submit its response, whether $\hat{k}$ is generated as per the method b equal to 0 or as per the method b equal to 1.

Now you might be wondering that what extra advantage the adversary here is getting by seeing the response of the decapsulation oracle service. If you see closely here, by seeing the response of the decapsulation oracle service, adversary comes to learn whether the triplet (u, c', z') is a

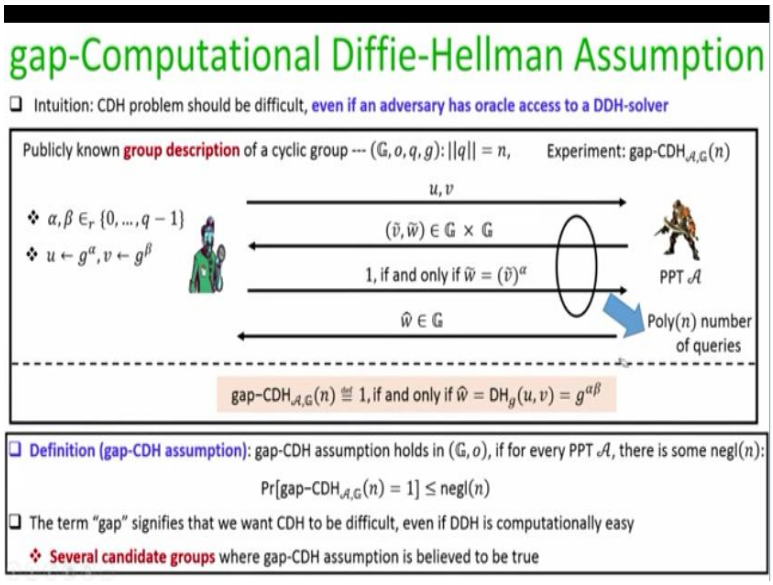
DDH triplet or not. More specifically, if the response of the decapsulation service for c' gives back the key k, which is different from the k', which adversary has computed, then adversary knows that definitely (u, c', z') does not constitute a DDH triplet.

Because contrapositively, if indeed (u, c', z') is a DDH triplet, then basically k would have been equal to k'. On the other hand, adversary also knows that if k' is equal to k, namely the response which comes out as the answer from the decapsulation oracle service corresponding to the adversary's query is same as the k', which she has computed, then with very high probability, the triplet (u, c', z') constitutes a DDH triplet.

This is because if (u, c', z') is not a DDH triplet, that means, this z' is different from $g^{\alpha\beta'}$, then it is very unlikely that the key derivation function on two different inputs namely $g^{\alpha\beta'}$ and some g^{γ}, where γ is different from α times β' gives you the same output and this holds, because we are in the random oracle model, where the key derivation function is behaving like a truly random function.

So that means, what we can now see here is that the way adversary is picking its decapsulation oracle service and doing some local computation then by comparing the response from the decapsulation oracle service, basically adversary has got an access to the DDH solver, which tells him whether a submitted triplet constitutes a DDH triplet or not. That means, just based on the CDH assumption and random oracle model assumption, we cannot prove that this construction of key encapsulation mechanism is CCA secure. Because through the decapsulation oracle service, adversary is now getting access to the DDH solver and that motivates us to now define a new assumption.

(Refer Slide Time: 14:47)



Or a new hard problem, which we call as the gap computational Diffie-Hellmen problem or gap computational Diffie-Hellman assumption and underlying intuition behind this assumption is that we require that the CDH problem should be difficult to solve in my underlying group, even if an adversary has got oracle access to a DDH solver. I stress here that he has got only an oracle access to the DDH solver; it does not have a polytime; it did not know the exact steps of that DDH solver.

So the way we model it here is as follows: we call this experiment as gap-CDH and basically the challenger prepares an instance of CDH problem by picking a random g^{α} and g^{β}, where α and β randomly chosen from the set 0 to q – 1 and the challenge for the adversary is to compute the Diffie-Hellman function of this u, v pair, namely it has to compute $g^{\alpha\beta}$.

But now to model the fact that the adversary has got oracle access to a DDH solver, we allow the adversary to submit polynomial number of queries of the form ($\tilde{v}$, $\tilde{w}$) and in response the challenger outputs 1, if and only if the challenger finds that this $\tilde{w}$ component is indeed this $\tilde{v}^{\alpha}$, namely this pair ($\tilde{v}$, $\tilde{w}$) constitutes a DDH triplet with respect to the u, which adversary has thrown as a challenge.

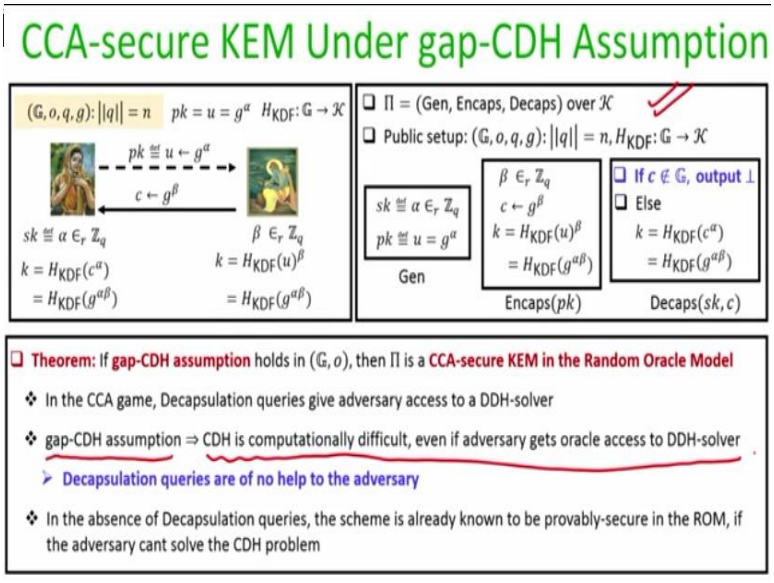
And adversary is now allowed to make polynomial number of such queries here and once it has made polynomial number of queries, the goal of the adversary is to compute the CDH function

of the challenge u, v, which is thrown to the adversary, namely it outputs a group element and definition of the experiment is we say that adversary has won the experiment, which we denote by saying that output of the experiment is 1, if and only if adversary is correctly able to solve the CDH problem, namely $\hat{w}$ is indeed equal to $g^{\alpha\beta}$.

We say that the gap CDH assumption holds in my underlying group g with respect to which this game is played is for every polytime adversary participating in this experiment, the probability that it wins the experiment is upper bounded by some negligible function. You might be wondering that what exactly the prefix gap signifies here. Well, the gap signifies here that we want a CDH problem to be difficult even if the DDH problem is computationally easy to solve in that group.

That means, we want to capture the essence that there is a gap between the difficulty of the CDH problem and the DDH problem here and it turns out that there are several candidate groups, where we believe that the gap CDH assumption holds, namely the gap CDH problem indeed is difficult to solve. For instance, the group based on the points on elliptic curves modulo prime is one such candidate group. You can also take the prime order subgroup of the multiplicative group Z_p*. In both these candidate groups, we believe that the gap-CDH assumption holds for sufficiently large values of the security parameter.

(Refer Slide Time: 18:04)



So now, let us see how we can instantiate key encapsulation mechanism under the gap-CDH assumption and it turns out that the construction is not a new construction. The construction is exactly the same construction, which we had seen, when we discussed a CPA secure instantiation of key encapsulation mechanism under the CDH assumption and DDH assumption. The only difference now is since we are now trying to model an active adversary, whenever receiver receives encapsulation c before going to decapsulate it, it just have to check whether the encapsulation c is the group element or not.

Because ideally, if indeed the encapsulation c is coming from honest party or an honest sender, then as per the steps of the encapsulation algorithm, the c component of the encapsulation should be a group element. On the other hand, if c is actually coming as a decapsulation oracle query, then what an adversary might try to do is, it can ask for the decapsulation of an element, which is not an element of the group and if the receiver does not perform the sanity checking and blindly proceed to decapsulate it and send back the output, then it might lead to a security breach here.

So that is the only additional check we are making here. The rest of the steps for the key generation algorithm, the encapsulation algorithm and decapsulation algorithm are exactly the same, as it was there earlier. Now what we can prove here, if the gap CDH assumption holds in my underlying group, then this instantiation of key encapsulation mechanism is CCA secure in the random oracle model.

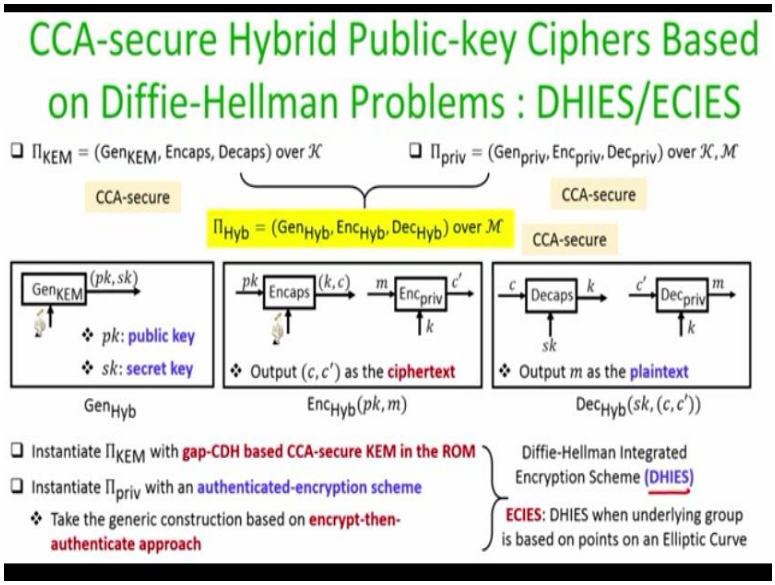
And let us see a very high level overview of the proof. We would not go into the formal details. If you are interested, you can see the book by Katz & Lindell for the formal details. So if we think of any adversary, a polytime adversary which participates in the CCA game against this encapsulation mechanism, then my claim is that the decapsulation, oracle decapsulation queries, basically gives the adversary access to the DDH solver and this we had seen earlier, right.

But if my gap-CDH assumption holds in the underlying group, right, then it means that even if the adversary gets access to the implicit DDH solver, the CDH problem still remains difficult for the adversary to solve. That means, the decapsulation oracle, the decapsulation queries are of no use at all to the adversary. It would not help him to solve the CDH problem and now what

basically we are doing here is basically since the decapsulation queries are of no help for the adversary, then we know already that in the absence of decapsulation queries an instance of CCA game against the key encapsulation mechanism is exactly the same as an instance of the CPA game against the same key encapsulation mechanism, which we had already proved to be CPA secure under the CDH assumption in the random oracle model. That means, we do not have to do add any sophisticated steps in this existing key encapsulation mechanism to make it CCA secure.

What we just have to ensure is that, we instantiate the steps of the encapsulation and decapsulation algorithm in a group where the gap CDH assumption hold and in the gap assumption holds, we are safe because the decapsulation oracle service are completely useless for the adversary and it gets no additional advantage compared to a CPA adversary.

(Refer Slide Time: 21:54)



So that means now we have an instantiation of CCA secure key encapsulation mechanism, so what we are going to do is, we are going to use it to come up with CCA secure hybrid versions of public key ciphers based on Diffie-Hellman problems and there are two well known instantiations of this framework, one is called DHIES and ECIES and idea here is basically to use the generic framework that we had discussed in the last lecture.

Where we have seen that if we are given a CCA secure key encapsulation mechanism and if we are given a CCA secure symmetric key encryption process, then the resultant hybrid encryption

process is CCA secure. What we have to basically now do is, we have to come up with CCA secure instantiation of key encapsulation mechanism. So what we can do is, we can use the key encapsulation mechanism, which is CCA secure based on the gap-CDH assumption, which we had just seen now.

And to instantiate the CCA secure symmetric key encryption, what we can do is, we can use any authenticated encryption scheme, which we had seen during our discussion on the symmetric key encryption process, namely we can use the generic construction based on the encrypt then authenticate approach, which always gives us the guarantee that the resultant scheme is an authenticated encryption scheme.

So remember in the encrypt then authenticate approach, we take a CPA secure encryption process, symmetric key encryption process and a secure message authentication code and combine them using this encrypt then authenticate approach, where we first encrypt the message using the CPA secure symmetric key encryption and the resultant encryption cipher text is authenticated as per the message authentication code to obtain the tag.

And (cipher text, tag) is considered as the overall cipher text and we had rigorously proved that this generic approach always gives us an authenticated encryption symmetric key encryption process and we know that in the symmetric key world, authenticated encryption implies CCA security, because one of the conditions for authenticated encryption scheme to be satisfied is that, it should have the notion of CCA security.

So to instantiate this CCA secure symmetric key encryption, we can use any authenticated encryption scheme. So now, we have instantiation of both the gadgets that we need in our hybrid encryption scheme. If we use these two instantiations, we obtain what we call as Diffie-Hellman integrated encryption scheme or DHIES in short and when we instantiate this DHIES where the underlying group is based on the points on elliptic curve, then the resultant DHIES instantiation is called as ECIES, namely elliptic curve integrated encryption scheme. These are two well known standards, which are used in practice to instantiate, hybrid public key ciphers based on

Diffie-Hellman problems. So that brings me to the end of this lecture. Just to summarize, in this lecture, we have seen instantiations of CCA secure key encapsulation mechanism.

For that, we have introduced a notion of gap-CDH and gap-CDH assumption and we had seen that if we combine a CCA secure KEM based on gap-CDH assumption along with any authenticated encryption scheme in the symmetric key world, which is symmetric key, then the resultant construction gives us CCA secure public key encryption process, which we call as DHIES and ECIES. Thank you.