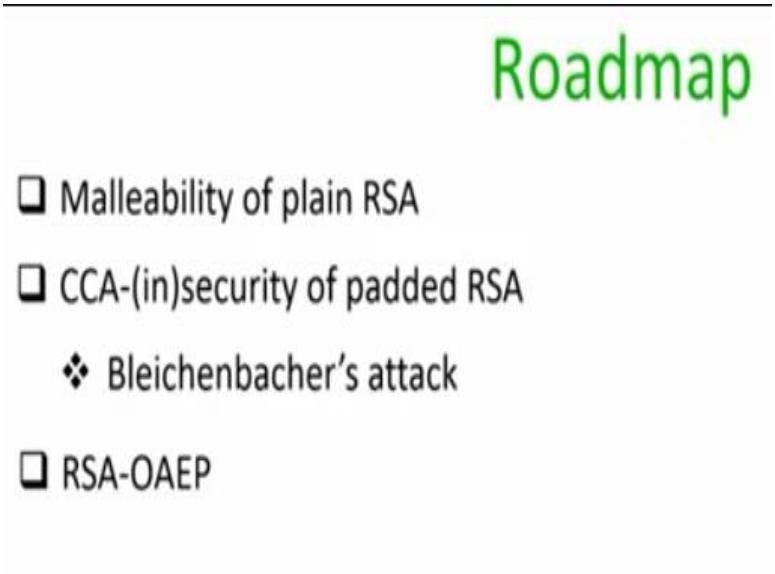
Foundations of Cryptography
Dr. Ashish Choudhury
Department of Computer Science
Indian Institute of Science - Bangalore

Lecture - 50
CCA Secure Public Key Ciphers Based on RSA Assumption

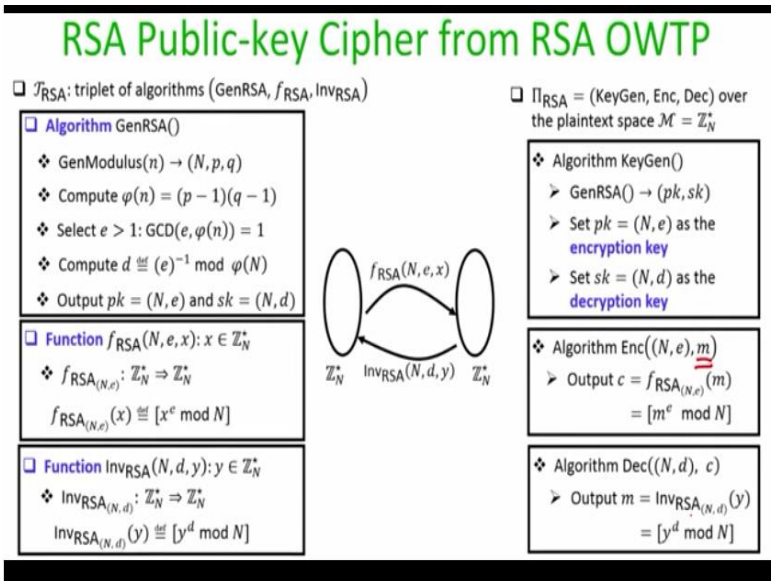
(Refer Slide Time: 00:30)



Hello everyone. Welcome to this lecture. Just to recap in the last lecture, we had discussed CCA secure instantiations of public key ciphers based on Diffie-Hellman problems. In this lecture, our goal is to see CCA secure instantiations of public key encryptions based on RSA assumptions, specifically the roadmap for this lecture is as follows: We will see the malleability of the plain RSA, which will prove that plain RSA is not CCA secure.

And we will consider padded version of RSA and prove its CCA insecurity, namely we will see a very interesting attack, which we call as Bleichenbacher's attack and then we will see a candidate CCA secure instantiation of public key encryption based on RSA assumption, namely RSA OAEP.

(Refer Slide Time: 01:12)



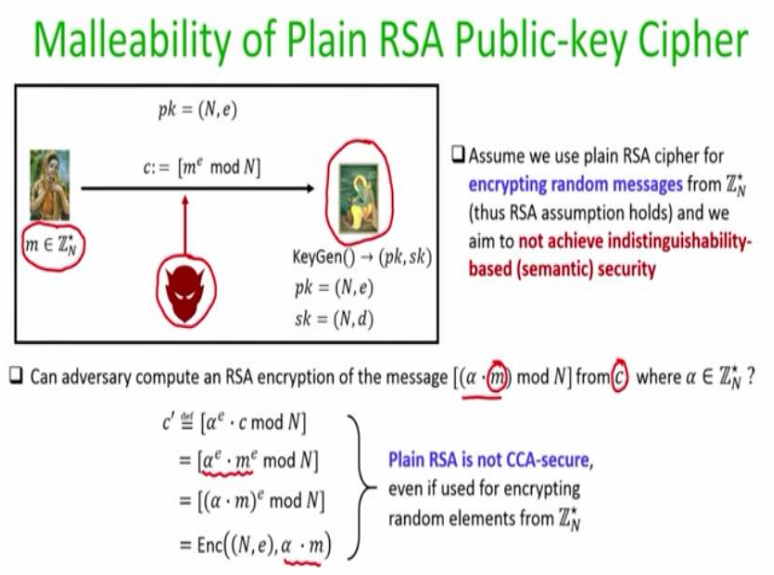
So let us recall the construction of plain RSA public key cipher from RSA one-way trap to permutation. So you have the RSA trapped permutation scheme, which has its key generation algorithm. Basically, the key generation algorithm picks uniformly random n-bit prime number p and q and compute the modulus begin, which is the product of p and q. Then, given p and q, it computes the value of phi of n, namely the size of the group Zn star.

Then, it picks e greater than 1, such that e is coprime to phi of n and then given that e is coprime to phi of n, we can compute the multiplicative inverse of e module of phi of n, which we denote as d and then finally, we set n, e to be the public key and n, d to be the secret key. The RSA function is a mapping from Zn star to Zn star and to compute the value of the RSA function on some input x, we basically compute x to the power e modulo n.

On the other hand, if you want to compute the inverse RSA function, then it is a mapping from Zn star to Zn star operated with the secret key n, d and to compute the value of the inverse function on some input y, we basically compute y to the power d modulo n. then, based on this we had seen an instantiation of public key encryption scheme, which we call as plain RSA cipher, where the key generation algorithm of the plain RSA is basically to run the key generation algorithm of the RSA trapped permutation scheme.

And set n, e to e the encryption key and n, d to be the decryption key. If there is a sender, which wants to encrypt a plain text m belonging to the set Zn star, then using the public key n, e, it can produce RSA cipher text, which is m to the power e modulo n and the receiver which obtains the cipher text c on possessing the secret key n, d can decrypt a cipher text by computing y to the power d modulo n.

(Refer Slide Time: 03:17)



Now what we are going to see is the malleability of plain RSA public key cipher. So remember when we introduced a plain RSA, we saw that RSA encryption definitely does not provide you CPA security, because it is not randomized. If you encrypt the same plain text multiple times using the same public key, you are going to obtain the same cipher text and not only that we cannot reduce the security of RSA encryption directly to the hardness of the RSA problem.

Because RSA problem requires that the underlying m should be a random element from Zn star, but when we are instantiating RSA cipher, the underlying plain text, which sender is encrypting may not be a random element from Zn star and not only that, we also discussed that even if we do not aim for semantic security, namely for us, if suffice if adversary does not learn the entire underlying plain text in its entirety and we are happy with that.

Then, even in that case, there are several possible attacks, which can be lost on the plain RSA public key cipher. So what we are going to now discuss is, we are going to consider the same

requirement, namely say we are happy with all or nothing kind of security. We do not aim for indistinguishability based semantic security. We are happy if adversary does not learn the entire underlying plain text in its entirety.

And we consider a scenario, where in my underlying application, sender is going to encrypt random plain text from the group Zn star using the RSA encryption scheme. What we are going to see here is that, even if we are using RSA for such an application, the plain RSA encryption process is malleable and if it is malleable, then obviously it is not CCA secure. So to demonstrate the malleability of RSA cipher, consider a scenario, where we have a receiver, who has set up its RSA public key and obtained a corresponding secret key.

And imagine, there is a sender, which has picked a random message from the underlying plain text space Zn star and encrypted it as per the plain RSA encryption process and imagine we have now an active adversary who has eavesdropped this cipher text. So the question that we want to now answer is, is it possible for this adversary to compute an RSA encryption of some message alpha times m, where alpha is known to the adversary and alpha is an element of Zn star.

But the adversary does not know the value of underlying plain text m, which is encrypted in the cipher text c. So that is what is the goal of the adversary. So adversary has to do some operation on the cipher text c, which has intercepted and from that, he has to do the computation in such a way that the modified cipher text should correspond to an RSA encryption of the plain text alpha times m and it turns out that it is very easy for an adversary to do that.

So what the adversary can do is, it can compute a modified cipher text c dash, which is alpha to the power public exponent e multiplied by the cipher text c, that adversary has seen modulo m. So here, everything is known to the adversary, which he requires for computing c dash and adversary knows alpha, because it is selected by the adversary. Adversary knows the public exponent e, because that is part of the public key, c is intercepted by the adversary.

And again, the modulus begin is also a part of the public key. So now it is easy to see that if we expand c, the cipher text, which has adversary has seen, that basically is nothing but m to the

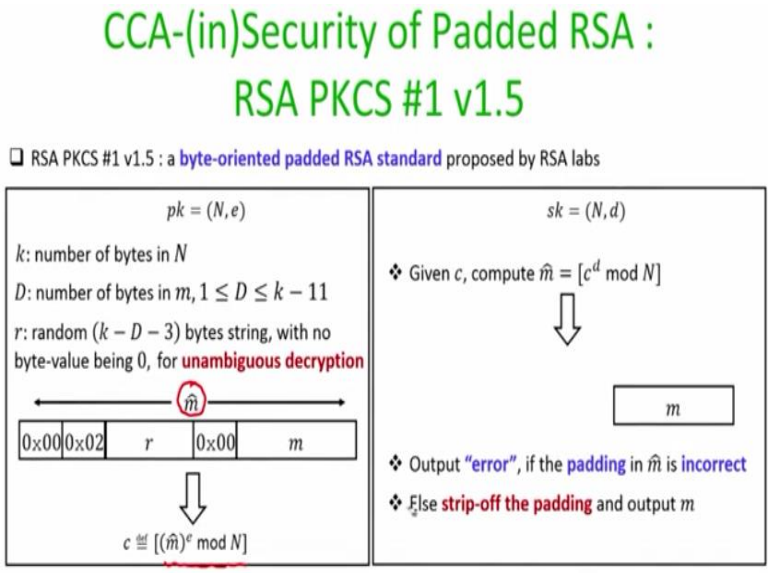
power e modulo N and now I can bunch this alpha to the power e multiplied by m to the power e and collectively write it as alpha times m, whole raised to power e modulo N and what exactly is alpha times m whole raised to e modulo N.

Well, that is basically is the RSA encryption of the plain text alpha times m using the plain RSA cipher. That means, if sender would have encrypted the plain text alpha times m, then the cipher text as per the plain RSA cipher would have been c dash and that means, the RSA encryption process is not CCA secure, even if it is used for encrypting random elements from Zn star, because we have shown that RSA encryption process is malleable.

And as we had done for the case of El Gamal encryption scheme, where we had showed that since El Gamal encryption scheme is malleable, we showed an instance of the CCA experiment, where adversary can completely identify what is encrypted in the challenge cipher text by getting the help of the decryption oracle service. The similar exercise, we can do for the case of plain RSA cipher as well.

In the CCA game, on receiving the challenge cipher text c star, adversary can compute an encryption of alpha times some challenge message m sub b and get the decryption oracle service for this modified cipher text. On decryption, it will learn the plain text alpha times mb and since alpha is known to the adversary, it can find out the message mb, which is encrypted in the challenge cipher text. So the exact details I am skipping, but the overall idea here is that since the plain RSA cipher is malleable, we can never hope that it is CCA secure.

(Refer Slide Time: 08:44)



Now next what we are going to discuss is, the CCA insecurity of even the padded RSA. So remember, plain RSA is not definitely CPA secure, because it is not randomized and during our discussion on the CPA security of RSA encryption scheme, we had discussed a potential way of making the RSA encryption process randomized by concatenating the plain text bit strings by random bit strings and mapping the padded thing as an element of the group.

And then, computing or encryption of the resultant group element as per the RSA function, right. So we are going to see one of those variants of the padded RSA, which we call as RSA PKCS standard 1, version 1.5, which is an instantiation of padded RSA proposed by the RSA lab and it was a very popularly used standard some years back until an attack has been proposed. So the standard is a byte oriented RSA standard and everything is treated here in terms of bytes here.

So what is available in this encryption standard is the public key of the receiver, which is the modulus and a public exponent and imagine that the size, number of bytes in the modulus is k, and imagine the sender's message is m and the standard allows that the sender's message should be anything in the range of 1, 2 up to k – 11 number of bytes. So the maximum number of bytes, which sender can encrypt as per the standard is k – 11.

Of course, message has to have, the plain text needs to have at least 1 byte. So that is the number of bytes, which is allowed as per this encryption standard. Now what this encryption standard

does is, basically it does a padding of the plain text along with some random string and with some publicly known stuff. So let us see how exactly the padding happens. Just to begin with, we put 2 bytes at the beginning here. The first byte represents the value 0.

So this is the hexadecimal representation of the integer 0 and the second byte represents the integer value 2 and this is publicly known. Now next what we do is, we select a random byte string consisting of up to k – d – 3 number of bytes, which is an uniformly random byte string, except that this byte string r should not contain the byte value 0. Other than that, it can contain any byte value, okay.

And the reason we are putting this restriction that this random stuff, which we are inserting here should not have the byte value 0 is to ensure unambiguous decryption. Now once we have inserted the random byte string r, that is followed by again a publicly known byte representing the integer value. So that tells you that why we require that random byte r, which we have inserted should not have the byte value 0.

Because what we are doing here is that, we are actually creating a kind of sealing or boundary for the random byte string r, which we have inserted. On the left hand side, the wall is created by the byte value 0. The left hand side the boundary is created by the byte value 2 and on the right hand side, the boundary is created by the byte value 0. Other than zero, anything else could be present in this byte string r and the number of bytes, which we can put in this byte string r is k – d – 3 number of bytes.

So if we sum up all these things, we get the total number of things that we are actually now including the plain text including the publicly known bytes and including the random byte string, everything sum up to k, okay. So this overall thing is now represented as the padded plain text, which I denote by m hat and now what we do is, we encrypt the padded message m hat by treating it as an element of Zn star.

Remember, even if this padded message m hat is not an element of Zn star, it is fine, because we have seen that the RSA function and the inverse function are mutually inverse of each other,

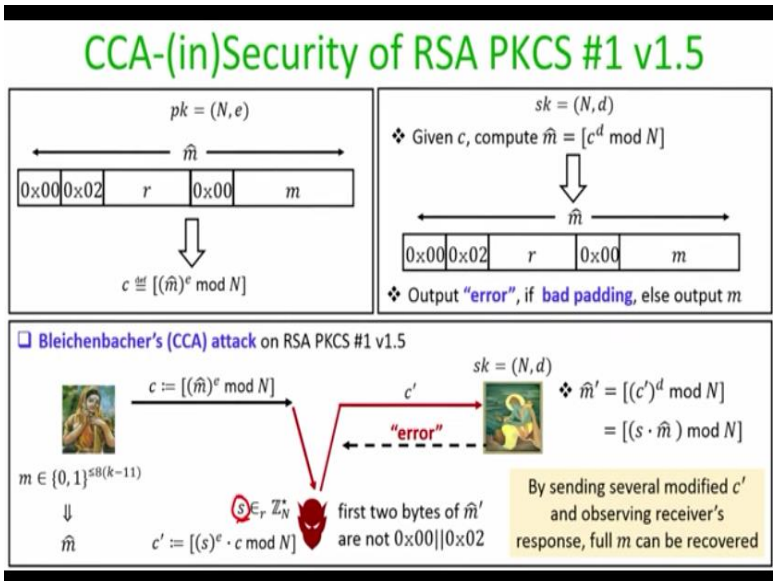
even if the resultant element is an element of Zn rather than Zn star. So it does not matter whether this padded element m cap belongs to Zn star or belong to Zn, we can safely apply the RSA encryption function to encrypt this padded group element m hat and resultant cipher text is c.

So that is the way this byte oriented RSA standard performs the encryption. To do the decryption, what the receiver is going to do is, if it receives a group element or it receives a cipher text c, what it performs is, it computes c to the power d modulo N to recover back the padded group element m hat and since it knows the expected format of m hat, it processes the m hat as a sequence of bytes.

Checks whether the first byte is 0 or not, second byte is 2 or not and then it starts scanning the random byte string, which sender would have potentially added in the message and then it looks for the boundary of the random byte string, namely where it ends as soon as the receiver obtains the byte value 0, that means receiver knows that the byte just before that marks the ending of the random byte string and now what the receiver checks is, whether everything in recovered m hat is in the correct format or not.

If the format in the recovered m hat is not in the same way that the receiver expects, namely if the first byte is not 0 or the second byte is not 2, or there is problem in the formatting of the random byte string, then the receiver outputs error message, else what it does is, it simply strips off the first byte, the second byte, and the random byte string and the auxiliary byte string 0 and the remaining thing is treated as the recovered plain text. So this is how this byte oriented RSA standard operates.

(Refer Slide Time: 15:27)



Now what we are going to show is that, if we implement this, if we use this byte oriented padded RSA standard against a CCA attack, then it is completely insecure and this attack is called Bleichenbacher's attack attributed to the name of the inventor of this attack. So the attack is explained as follows. Imagine, we have a receiver, who has obtained its secret key and it has set up its public key available in the public domain and imagine there is a sender which has message m consisting of k- up to k – 11 number of bytes.

So does the padding of the message, which it wants to encrypt as per the given standard and obtain the padded mapped group element m hat and what it performs is, the encryption of the padded group element as per the RSA function. Now imagine, there is malicious adversary active adversary, who obtain this cipher text and now what this adversary does is, it exploits the malleability of the underlying RSA function.

So what it does is, it picks a random element from the group Zn star, which we call as s and it computes a modified cipher text c dash, which is s to the power e multiplied by cipher text c, which adversary has intercepted modulo N and then it forwards this modified cipher text c dash to the receiver pretending as if this message is coming from the so called sender and wait to see the receiver's response.

So as per the syntax of this padded RSA, the modified cipher text will be decrypted as c dash raised to power d modulo N, which will give the receiver the message s time m hat, not actual m hat, because now the cipher text c dash is not the encryption of m, rather it is an encryption of s time the padded element m hat. So I call that recovered thing as m hat dash and now what the receiver is going to do is, receiver will check whether the padding is correct in the recovered group element m hat prime or not.

If the receiver finds that the padding is not in the correct format, then the receiver is going to throw an error message and if the adversary sees that the receiver is throwing an error message, then the adversary gets the information that the first two bytes of the recovered message, which the receiver has recovered namely m hat prime are not the byte value 0, followed by the byte value 2, because only if the first two bytes of the recovered m hat prime are not 0 and 2, then only the receiver would have thrown the error message.

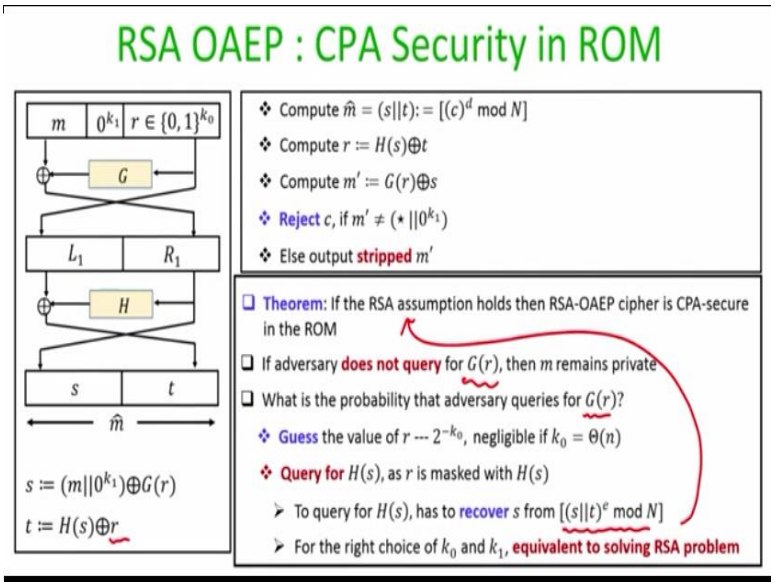
So this is kind of some information, which adversary learns by getting the decryption oracle service from the receiver by making a decryption oracle request for the cipher text c dash and now what the adversary can do is, adversary can send several such modified c dash with different values of s, and by observing the behavior of the receiver for each of the modified cipher text, which adversary has forwarded to the receiver, adversary learns some equations.

Basically, it learns some information about the underlying unknown plain text m and the value s, which adversary has picked and it turns out that by sending sufficient number of such modified cipher text and seeing the response of the receiver, adversary ends up learning the entire original plain text m, which sender had encrypted as per this RSA standard. That clearly shows that this RSA standard is not CCA secure.

Now it is easy to see that attack scenario here, the attack principle here is similar to the padding oracle attack, which we had seen in the context of CBC mode of encryption, when we were discussing the symmetric key encryption scheme. There also, adversary was basically modified the cipher text, forwarding it to the receiver and based on the response of the receiver, whether receiver is throwing back padding error message or not.

Our adversary was learning something about the underlying cipher text, right, underlying plain text. The similar principle now adversary is carrying out in the context of this public key RSA standard.

(Refer Slide Time: 20:01)



So that brings us to the question that do we have any CCA secure instantiation of public key encryption scheme based on RSA function and it turns out that indeed we have one such candidate, which we call as RSA OAEP. So basically OAEP stands for optimal asymmetric encryption padding, because this is a kind of padding, which we performed on the bit string, which we want to encrypt as per the RSA function before actually doing the encryption as per the RSA function and the padding is done as follows:

So the plain text space supported by this RSA OAEP consists of bit strings of length l bits, where l is publicly known and in this padding scheme, we know that description of some publicly known hash functions gNh. So the hash function g maps bit strings of length k0 to bit strings of length l + k1 bits. On the other hand, hash function H maps bit strings of length l + k1 bits to bit strings of length k0 bits.

And we assume that the summation of l and k0 and k1 is strictly less than the size of the modulus begin, which is going to be used to compute the value of RSA function. So the way we do the

encryption in this RSA OAEP encryption process is as follows: Imagine, sender has a plain text m, consisting of l number of bits. So it has to do the padding of this message, as per this OAEP padding to obtain a group element m hat.

And basically this OAEP padding or encoding is nothing but 2 round Feistel network, where in one of the rounds, we use the function g as the round function and in the second round we use the function h. So to begin with, the sender appends the plain text by k1 number of zeros and this is publicly known and this will be considered as the current left half for the Feistel network and the current right half for the Feistel network is said to be random string of length k0 bits.

So that is how we can interpret a current left half and the current right half as per this OAEP padding and now what we do is, we apply the round function g and remember as per the syntax of the Feistel network, the current right half goes and becomes the next left half and next right half is obtained by xr'ting the current left half with the output of the g function on the current right half. So that is how we obtain the next right half.

That completes the round 1 of the OAEP padding and in the second round, what we do is, we use the function h as the round function. So the current right half goes and becomes the next left half and current left half is xr'd with the output of the h function on the current right half and that becomes as the next right half and the concatenation of the next left half and the next right half is treated and interpreted as an element of either Zn star or Zn depending upon whether it is relatively prime to your modulus or not and we call that resultant element as m hat.

So as per the Feistel network nomenclature, it is easy to see that my resultant s part in the element m hat will be xr of G of r and the concatenation of m followed by k1 number of zeros, whereas the t part of m hat is nothing by the xr of the r value with the output of the H function on the input s and we do the encryption by mapping m to m hat as per the OAEP and then computing the output of the RSA function on the element m hat.

To do the decryption, what we do is, we decrypt and obtain cipher text by computing the inverse RSA function and parts are obtained m hat as left half and right half. We know that what should

be the expected left half and what should be expected right half and then we invert the Feistel network. So we know that to compute r, we have to compute the xr of t and H of s. We have already obtained t, H is publicly known, so we can compute H of s.

And once we have obtained r, we can obtain m dash, right, namely the actual left half to begin with, which is nothing but the xr of the s portion with the output of the G function on the input r and then we check whether the recovered left half is of the form bit string of length l bits, which I denote as star, followed by k1 number of zeros or not, because that is what ideally should be the case, if everything would have happened perfectly.

So if the recovered m dash portion, if the recovered left half portion is not of this form, we give an error message that reject the cipher text. It is an invalid cipher text. Otherwise, what we do is, we strip off the k1 number of zeros and whatever is there the number of l bit string that we have recovered in the m dash part is taken as the output plain text. So now let us try to argue about the CCA security of this RSA OAEP encryption process.

So the theorem that we can prove is that if the RSA assumption holds, then this RSA OAEP cipher is a CPA secure cipher in the random oracle model. So the proof is really involved and subtle here, so what we are going to discuss is only the high level ideas of the proof here. The full details of the proof, you can obtain in the manuscript by (()) (26:09).

So the idea behind the CPA security proof is that if any adversary participates in the CPA game against this RSA OAEP encryption process, then our challenge plain text, which has been encrypted in the challenge cipher text remains hidden if the adversary does not query for G of r, because the actual plain text m, which is encrypted in the challenge cipher text would have been xr'ed with G of r and since we are in the random oracle model, the function G is instantiated as a random oracle, where the adversary does not know the code of the function G.

So now the question is what is the probability that adversary indeed queries for G of r. So there could be two ways. One option could be adversary guesses the value of the randomness, which has been used by the challenger and a successful probability of guessing the Rs1 over 2 to the

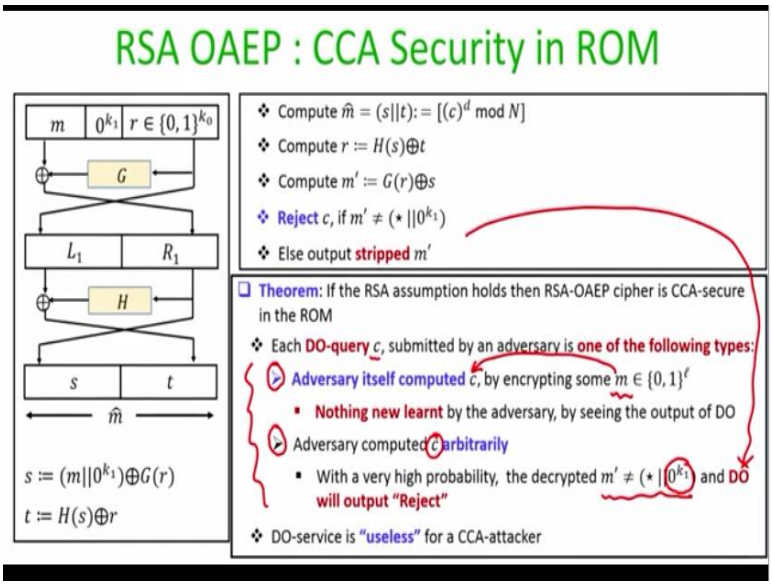
power k0 and it is negligible if we set k0 to be some theta function of a security parameter. On the other hand, the other way, the adversary could have queried for the G function on the input r is that if the adversary would have queried the H oracle on the input s.

Because if you see the structure of the OAEP, r is actually xr'ed with H of s, right. So basically G of r is xr'ed with s, so until and unless r is xr'ed with H of s, so if adversary would have queried for H of s, then probably, we can hope that adversary would have learned r, but to query for H of s, basically adversary has to recover the s portion from the cipher text c. So remember the cipher text c is nothing, but the output of the RSA function on the concatenated input s followed by t.

Sorry, it is output of the RSA function on the input s concatenated with t. So the only way adversary could have queried the H oracle on the input s is, if it would have recovered s from the RSA OAEP encryption of s concatenated with t and it turns out, we can prove formally that if we set k0 and k1 appropriately, then this is nothing but equivalent to solving the RSA problem, but since we are assuming that the RSA assumption holds.

It is very unlikely that adversary will be able to recover s and hence it is very unlikely that it would have queried the H oracle on the input s. So that is the overall idea of the CPA security prove, but formalizing this idea into the exact formal details is really challenging and due to interest of time and since the proof is out of scope of this course, I am excluding the formal details.

(Refer Slide Time: 28:55)



Interestingly, we can prove that this RSA OAEP is also CCA secure in the random oracle model. If we assume that the RSA assumption holds in my underlying group. So if we play the CCA game against this encryption process, then the extra advantage adversary now have is that it has got access to the decryption oracle service. It can submit any cipher text and see the decryption of that cipher text as per this RSA OAEP encryption process.

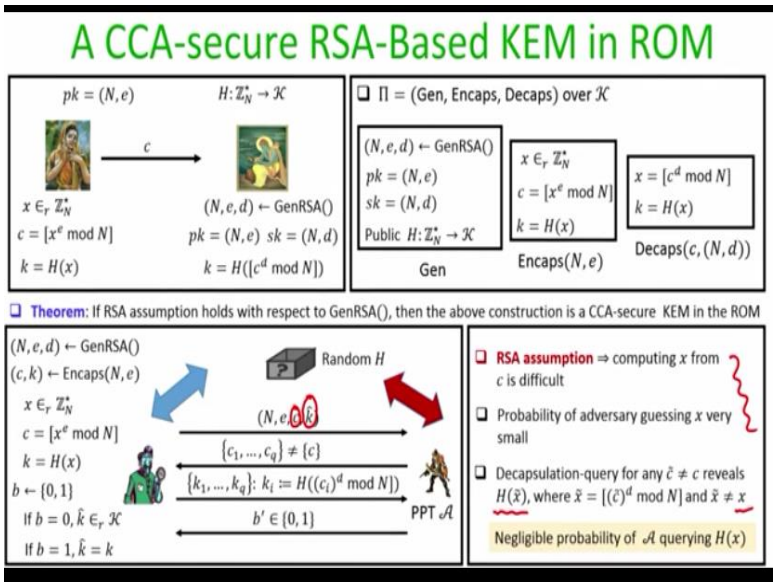
What we can prove is that any DO queries for the cipher text c, which adversary submits can be of one of the following two types. Class 1 can be adversary has itself computed c, by itself picking some message m and encrypting it as per the RSA OAEP standard. If that is the case, then submitting DO query for a cipher text c gives absolutely no new information to the adversary, because adversary will already know what exactly will be the decryption of the cipher text c.

Because it itself has prepared the cipher text c from the plain text m. So it will know the output of the DO oracle on such c's will be the m, which it has picked. The second class of DO queries could be where adversary has computed some random bit string arbitrarily in some random fashion. So what we can prove is that, if that is the case, then with a very high probability that arbitrarily prepared cipher text c, which has been submitted to the decryption oracle.

When decrypted will turn out to something m dash of the form, where the k1 number of zeros are not present after the l number of bits in the recovered m dash and if that is the case, then remember as per the syntax of the OAEP decryption, the decryption oracle will simply reject such cipher text and adversary will learn nothing new at all. So in summary what we can say is that if we use the RSA encryption process.

Then it is very unlikely that the decryption oracle service is going to be useful for the adversary, because the adversary submits decryption oracle query of type 1. It learns absolutely nothing, whereas if it submits decryption oracle queries of type 2, then with very high probability, they are going to be rejected by the decryption service and we are back to the CPA world, where we had already seen a high level overview that why the scheme is CPA secure.

(Refer Slide Time: 31:30)



So now we have a CCA secure instantiation of RSA public key encryption scheme, but what we can do is the RSA OAEP standard can be useful only as long as the sender has a message of size l bits, but if the sender has a message of size more than l bits, then we can do better and go for a hybrid version of RSA encryption scheme, which is CCA secure and the idea here is basically to design a CCA secure key encapsulation mechanism based on the RSA function, whose security we can prove in the random oracle model.

If we can come up with such a CCA secure key encapsulation mechanism, then by generically combining it with any CCA secure symmetric key encryption process, we can obtain hybrid public key encryption scheme based on the RSA function. So here are the details of the CCA secure key encapsulation mechanism based on the RSA function. So imagine receiver is there, who wants to set up its public key and secret keys.

So what it does is, it runs the key generation algorithm of the RSA trapdoor permutation, obtain the public key and a secret key, and it sets its public key and along with this we assume that public description of a hatch function or a key derivation function mapping the elements of Zn star to the key space of the symmetric key encryption scheme is publicly available. So that is what is the key generation algorithm of the key encapsulation mechanism.

Run the RSA key generation algorithm, set up the public parameter, retain the secret key and along with that make the description of the key derivation function public. So that is the key generation algorithm. Now imagine, there is a sender, which wants to run encapsulate a key. So what the sender does is, it picks a random element from the group Zn star, which we call as x and it computes an encapsulation of that x as per the RSA function.

Namely, it computes c, which is x to the power e modulo m and then it maps the group element x to the key space by applying the key derivation function on the group element x and encapsulation c is sent to the receiver. So that is what is your encapsulation algorithm. Now to decapsulate this encapsulation c, what the receiver has to do is, it has to compute the RSA inverse function on the encapsulation c to get back the group element x.

And once it obtains the group element x, it can apply the publicly known hash function of the key derivation function to obtain back the same encapsulated key k. So that is what is your decapsulation algorithm. So on a very high level, it looks very simple and what we can prove is, if the RSA assumption holds with respect to this GenRSA function, then the hour of construction is a CCA secure instantiation of key encapsulation mechanism in the random oracle model.

So let us formally prove that. Imagine we have an adversary participating in the CCA game against the key encapsulation mechanism and since we are in the random oracle model, at the beginning of the experiment, a random function H will be instantiated and both the challenger as well as the adversary will have oracle access to this random function. What the challenger will do, it will run the key generation algorithm, obtain the public key and a secret key.

And then it will run the encapsulation algorithm to obtain a key and its encapsulation and a way this key and encapsulation would have been computed, will be as per the steps of the key encapsulation algorithm, namely it will pick a random group element, compute its output of the RSA function on that x and it will map that x to the key space fancy k and then it will prepare a challenge for the adversary by tossing a fair coin.

If the coin is 0, then a random element from the set fancy k is picked. Otherwise the challenge is said to be the key k, which is encapsulated in c and now the challenge is thrown to the adversary, namely the description of the public key n, e, the encapsulation c and a key k hat and adversary has to find out, whether k hat is generated as per the method b equal to 0 or as per the method b equal to 1.

Since we are in the random oracle model, adversary could ask the output of the H oracle on polynomial number of group elements and then since we are in the CCA world, adversary can ask for the decapsulation oracle service by submitting polynomial number of elements ci's from the group, which should be different from the encapsulation c and as per the rules of the CCA game, the challenger has to decapsulate all such encapsulations.

And each of those response will be nothing, but the output of the H oracle on the input ci to the power d modulo N and then finally the adversary submits its response. So the underlying idea behind a security proof here is the following. If we are assuming that the RSA assumption holds, then computing the exact x from the c is difficult, because that precisely an instance of the RSA problem.

Now since the adversary is given the access to the H oracle, the probability that adversary could guess the x and ask the output of the H oracle on that x is very, very small, right. So that comes from the assumption that RSA assumption is true. It turns out that even though the adversary is now given the advantage of asking decapsulation queries for any c tilde different from c, it reveals nothing about the actual x.

Because the result of every such decapsulation query will be the output of the H oracle on some input x tilde, where x tilde will be different from the group element x, which has been used by the challenger. So overall, we get the conclusion that the probability that the adversary queries for H of x is very, very negligible and if adversary has not queried for H of x, then it does not matter whether we are in the world b equal to zero or whether we are in the world b equal to 1, k hat is uniformly random element from the viewpoint of the adversary, which proves that this key encapsulation mechanism is CCA secure.

So that brings me to the end of this lecture. To summarize, in this lecture, we had discussed the malleability of the plain RSA cipher and we also discussed the Bleichenbacher's attack on one of the versions of padded RSA and we also discussed a CCA secure instantiation of RSA public key encryption scheme based on OAEP padding and finally we saw an instantiation of CCA secure key encapsulation mechanism based on RSA function, whose security is in the random oracle model. Thank you.