

Foundations of Cryptography
Prof. Dr. Ashish Choudhury
(Former) Infosys Foundation Career Development Chair Professor
Indian Institute of Technology – Bangalore

Lecture – 6
Introduction to Computational Security

(Refer Slide Time: 00:31)

Roadmap

- ❑ Birth of modern cryptography
- ❑ Computational security
 - ❖ Necessary evils associated with computational security
- ❑ Defining efficient algorithms and negligible probability asymptotically

Hello everyone, welcome to lecture 6. The plan for this lecture is as follows. We will discuss about the birth of modern cryptography, namely the approach that we use in modern cryptography. We will also discuss about the notion of computational security and necessary evils which are associated with computational security, and finally, we will define mathematically what do we mean by efficient algorithms and negligible probability.

(Refer Slide Time: 00:54)

Perfect Security : The Impractical Goal

❑ Perfect secrecy is always desirable. But comes with a heavy price

- Key as long as the message
- Fresh key for every instance of encryption

❑ Practical perfectly-secure encryption --- cheating

❑ Modern cryptography follows a different "approach"

- Attempt to get "closer" to perfect secrecy
- Getting rid of the practical limitations imposed by perfect secrecy --- shorter, re-usable key

So, just to recall in the last couple of lectures, we discussed about the notion of perfect secrecy, which is always desirable because that is the strongest notion of secrecy we can achieve. Because the secrecy is achieved against an adversary who is computationally unbounded whose running time is unlimited, but we also discussed that we have to pay a heavy price to achieve perfect secrecy, namely in any perfectly secured encryption process, your key need to be as largest a message and each instance of the encryption needs to use a fresh key.

So these 2 restrictions are kind of impractical, because in practice, we aim to design an encryption process where we can use a short key for encrypting long messages and we would like to have an encryption scheme where the same short key could be used for encrypting sequence of multiple messages. That means, practical perfectly-secure encryption is simply not possible, that is kind of cheating. So if someone claims that his or her encryption process is practical as well as it provides you perfect secrecy, then that is clear cheating.

That brings us to the approach that modern cryptography follows. So the principle that we follow in modern cryptography is that instead of achieving perfect secrecy, we will try to get as closer as possible to perfect secrecy and in return, we achieve two practical goals which we aim for. Namely, we achieve an encryption process where we can use the same short key for encrypting multiple messages. So that is the kind of tradeoff we use in modern cryptography.

(Refer Slide Time: 02:35)

Birth of Modern Cryptography

Consequences :

- Key as large as the plain-text
- Key cannot be re-used



Computationally **unbounded**



Learns **absolutely nothing** additional about the plain-text

So, let us see the approach that we use in modern cryptography. So remember, in the perfect secrecy model, our adversary is computationally unbounded and a secrecy goal in the model of perfect secrecy was that we want to ensure that adversary learns absolutely nothing about the plain text right and then we rigorously formalize this notion, what do we mean by adversary learns absolutely nothing about plain text.

We also discussed that the consequences of this goal, namely the goal of achieving that adversary achieves learns absolutely nothing is that you have to have a key as large as the plain text and you cannot afford to reuse the key right. So, that was the consequences or the restrictions of perfect secrecy.

(Refer Slide Time: 03:33)

Birth of Modern Cryptography

Consequences :

- Key as large as the plain-text
- Key cannot be re-used



Computationally **bounded**



Learns additional info. about the plain-text with a **negligible prob.**

Now, the changes that we are going to do in modern cryptography is as follows. Instead of assuming that our adversary is computationally unbounded or computationally unlimited, we assume that our adversary is computationally bounded techniques, that means we are no longer going to assume that adversary could run his algorithm for breaking the scheme or attacking the scheme for an unlimited period of time. We will see how to mathematically formulate this notion of computationally bounded adversary.


The second relaxation that we are going to make in the model of perfect secrecy is that instead of demanding that adversary learns absolutely nothing about the plaintext, we target to achieve the following goal. We target to achieve that adversary should learn additional information about the plaintext with a negligible probability, that means we are now willing to let adversary learn something about the plaintext, but that is additional information or the probability with which the adversary could learn the additional information is so small, it is so negligible that for almost all practical purposes we can ignore.

(Refer Slide Time: 04:42)

Birth of Modern Cryptography

Implications :

- Relatively much shorter key
- Key can be re-used




Computationally bounded



Learns additional info. about the plain-text with a negligible prob.

Computational security



(Modern Cryptography)

As soon as we make these two relaxations and the model of perfect secrecy, we should hope that we should get the following two implications. Namely, our encryption process should be using short key and the same short key should be usable to encrypt a sequence of messages, and it turns out that indeed if we make these two relaxations in the model of perfect secrecy, we can achieve the two desired goals, namely the same short key can be used for encrypting sequence of long messages and that is what the approach modern cryptography use.

The notion of secrecy that we get by making these two relaxations is what we call computational secrecy, right? Because the security is achieved against an adversary whose computational power is now limited, rather than saying that the adversarial computing power is unlimited, right? So, that is the approach we are going to follow in modern cryptography.

(Refer Slide Time: 05:38)

Computational Security : The Basic Idea

- Two relaxations to the model of perfect-security to achieve key reusability
 - Security preserved only against **efficient adversaries** running in a **feasible/practical** amount of time
 - Adversaries are allowed to break the scheme with **some probability**, which is **so small** that we do not bother
 - ❖ Under certain assumptions, the amount of time required to break the scheme will be of order of **few lifetimes**
 - ❖ Acceptable, as most applications do not require **ever-lasting security**
- The above relaxations are necessary if key reusability is the goal

So the two relaxations that we are going to make is we are now aiming to achieve security only against efficient adversaries, and what I mean by efficient in adversaries is informally those algorithms those adversarial algorithms whose running time is practical or whose running time which takes an amount of time which is feasible and practice. We will mathematically define what exactly we mean by efficient adversaries very soon.

The second relaxation that we are going to now make is to assume that adversaries allowed to break the scheme with some probability, but the probability with which the adversary can break the scheme is so small that we do not bother about such a small probability. Again, we are going to very soon mathematically and rigorously define what exactly we mean by such small error probability. Moreover, as we will see during the course, as the course proceeds, that under certain assumption the amount of time that the adversary will require to break the scheme with that is small probability will be of order of few lifetime.

This is in contrast to what we achieve in perfect secrecy. In perfect secrecy, even if we give the adversary unlimited time, there is absolutely zero probability that he learns anything about underlined plain text, but in the computational security, in model of computational security where our goal is to achieve key reusability is to give enormous amount of time to

the adversary, then there is a chance that adversary will be able to learn something about the underlying message, but that something is going to be so small that for most practical purposes, we can ignore it, right.

Moreover, the amount of time that is going to require for the adversary to learn the message, it is some that a small probability will be of order of few lifetimes. It turns out that this is acceptable for most of the practical applications because in most of the practical applications, we do not require everlasting security. What I mean by this last statement is the following. Imagine you want to have a secure system to maintain the secrecy of your credit card details, right.

So if I have an encryption process, which ensures that it will preserve the privacy of your credit card details right, with significant amount of probability, that means the probability that adversary can learn your credit card details by looking into the encryption of your credit card details with very, very small probability and the amount of time that the adversary will take to learn about your credit card details is of order say 35 years or 40 years, then it is fine because ideally, you will require the secrecy of your credit card details to be maintained only for few years..

You do not require the secrecy or the privacy of your credit card details to be maintained forever lasting, right. So this is acceptable. As it will turn out that above two relaxations that we are making in the model of computational secrecy is absolutely necessary if our ultimate goal is to achieve key reusability. Namely, in the next couple of slides, we are going to discuss that indeed if we want to design the encryption process where our goal is to ensure that the same short key is used to encrypt multiple messages.

Then definitely we need to make the two relaxations that I am discussing here, namely the first relaxation is that we should be willing to let the adversary learn something about the underlying message with a small probability and a second relaxation is that we should target security only against efficient adversaries, right?

(Refer Slide Time: 09:16)

Relaxation I : Security Only Against Efficient Adversaries

❑ Consider an encryption scheme where same key is used to encrypt multiple messages

❑ Consider an adversary, launching a brute-force key-recovery attack in the KPA model

- ❖ Adversary gets access to $(m_1, c_1), \dots, (m_t, c_t)$, where each $c_i \leftarrow \text{Enc}_k(m_i)$
- ❖ Checks if there is some $k \in \mathcal{K}$, such that $\text{Dec}_k(c_i) = m_i$, for each (m_i, c_i)
- ❖ Running time: $\mathcal{O}(|\mathcal{K}|)$ success probability: 1

$|\mathcal{K}| = 2^{256}$

So let us see the necessity of these two relaxations. Namely, the first relaxation is that we should now target security only against efficient adversaries. So, to see that why this relaxation is necessary, consider an arbitrary encryption process, where you are going to use the same key for encrypting sequence of messages and namely, the same key is going to be used for encrypting multiple messages. And imagine we are in the known plaintext attack model KPA attack model.

Just to recall, in the KPA attack model, we assume that adversary sees encryptions of several messages, and it means it knows both the underlying messages as well as their encryptions under an unknown key, where the same key is going to be retained for encrypting for the new messages, right. So imagine I have an arbitrary encryption process whereas say the sender has encrypted message m_1, m_2, m_t and the resultant ciphertext are C_1, C_2, C_t and adversary has got access to this plain text comma ciphertext pairs, right?

It knows that each of the ciphertext in each of these pairs has the encryption of the corresponding plain text under some unknown key k , that the key is not known to the attacker okay. The adversary also knows the description of the encryption process and it also knows that the same unknown key k is going to be retained by the center for encrypting next sequence of messages. So, under this scenario, the adversary can always run what we call as the brute-force key key-recovery attack right.

Idea behind this brute-force key-recovery attack is that what the adversary can do is since it knows the description of the key space, it can try for each candidate key k belonging to

the key space and decrypt each of the ciphertext in his pairs of plain text comma ciphertext pair and see that does there exist a candidate key little k such that each of the ciphertext in his pairs of messages comma ciphertext pair decrypt back to the corresponding plain text under that candidate key little k ?

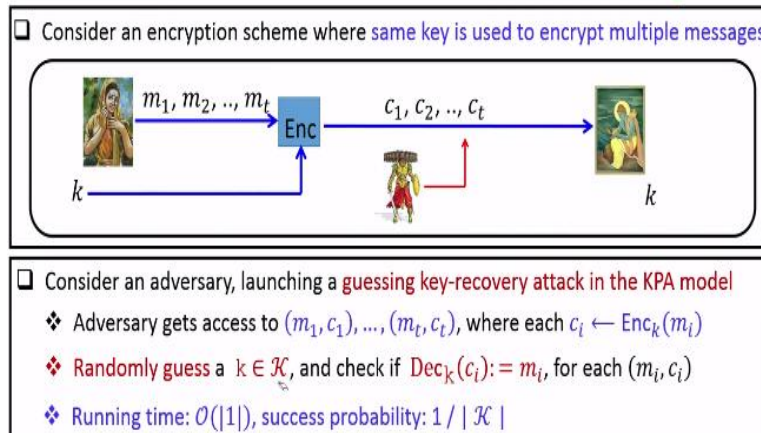
If he could, definitely there is some candidate key little k and as soon as it hit upon that little candidate key little k , it can find out what is the key which sender is going to use for encrypting the next sequence of messages. So you can see that the success probability of this brute-force key-recovery attack is one because definitely there exist some key in the key space which adversary will hit upon when it runs the brute-force key-recovery attack, but if you see the running time of that adversary, the running time of this brute-force key-recovery algorithm is order of the number of candidate keys, maybe the size of the key space.

So if we assume that our key space is significantly large for us, for instance, imagine that the key space is the set of all possible 256 bit strings right? That means imagine my key space is 2 to the power 256. Now this brute forcing over a key space of size 2 to the power 256 is going to take enormous amount of time, that is kind of impractical. That means if I make the relaxation that I am not going to tolerate or I am not bothered about adversaries whose running time is impractical, then this brute-force recovery attack will not be considered as an attack in my attack model.

So that is the necessity of the first relaxation if your goal is to achieve an encryption scheme, where the same key is going to be used for encrypting multiple messages, right, that shows us the necessity of the first relaxation.

(Refer Slide Time: 12:54)

Relaxation II : Allowing the Scheme to be Broken with a Small Probability



Now let us see the necessity of the second relaxation if your goal is to achieve key reusability. The second relaxation is that you should allow the scheme to be broken with a small probability. So again, consider an instance of an arbitrary encryption process where the same key k has been used to encrypt multiple messages in sequence and say the adversary is in the KPA attack model, where it has got access to several message-ciphertext pairs and the key is not known to the adversary.

But the adversary knows the corresponding ciphertext or the encryptions of the corresponding plain text in each of the pairs, right, and the adversary knows that the same unknown key k is going to be retained by the center for encrypting the next sequence of message. Now, the adversary can always launch what we call a guessing attack, right, and the idea behind a guessing attack is that the adversary can simply get a candidate value of key, say k from the key space and check whether that candidate key which he has guessed is indeed the right key or not by performing the following operation.

Namely, it can check whether under that guessed key k , each of the ciphertext c_i gives him back the corresponding plain text m_i , and if it so happens, then he has hit upon the right key. Now, what is the success probability of this attack? The success probability of this attack is one over the key space. What is the running time of the attack or the attacker's algorithm. The running time of the adversary's algorithm is constant because he is now not doing brute-force over the key space, he is just guessing the value of the candidate key, right.

So again, if I assume that my key space is extremely large, that means imagine again for the moment that your key space is order 2 to the power to 256, then the success probability of this attack is 1 over 2 to the power 256, which is very, very small. That means even though adversary has a chance to break the scheme, namely to learn the scheme, the chance that he can learn the key is so small that, namely, it is 1 over 2 to the power 256 that we can ignore it for most practical purpose.

So, this again demonstrate that if key reusability is your ultimate goal, then we have to make the second relaxation in our model, namely, we should be willing to let the adversary break the scheme with a smaller probability and which is so small that we can ignore it, right.

(Refer Slide Time: 15:19)

Key-Reusability : Necessary Evils

Goal: key-reusability

Unavoidable to prevent two extreme attacks on such a system in the KPA model

- ~~◆ Brute force key recovery attack: Running time: $\mathcal{O}(|\mathcal{K}|)$, success probability: 1~~
- ~~◆ Guessing key recovery attack: Running time: $\mathcal{O}(1)$, success probability: $1/|\mathcal{K}|$~~

- Relaxation I : Goal is to achieve security only against efficient adversaries
- Relaxation II : Small probability of a break in the scheme

So, just to summarize the two necessary evils which are associated with our ultimate goal of key reusability is, are the following right. So to prevent there are two possible attacks, two extreme attacks which can always be launched against an arbitrary scheme where the key reusability is the ultimate goal. The first attack is the brute-force key-recovery attack, whose running time is very large, but success probability is 100%.

There is the second extreme attack against such scheme where key reusability is the goal, where the running time of the attacker is very less, it is constant, but the success probability of the attacker is very very small, it is so small that for most practical purposes we can ignore. So now if you see that if we make the two relaxations that I have been talking about, namely the first relaxation where we target to achieve secrecy only against efficient adversaries, then

the brute force attack would not be considered as an attack in our attack model because as I said, the time complexity of the brute force recovery attack will be enormously large.

If I make the second relaxation, namely where I am willing to let the adversary learn or break the scheme with a very small error probability, then the second attack, namely the key recovery attack would not be considered as an attack in our attack model, right.

(Refer Slide Time: 16:42)

Key-Reusability : Necessary Evils

Relaxation I : Security targeted only against efficient adversaries

Relaxation II : Small probability of a break in the scheme

How to mathematically define efficient adversaries ?

How to mathematically define small (negligible) probability ?

So this is the summary of the two necessary evils which are associated with any encryption process, where key reusability is the goal. The first relaxation that you should make in your model is instead of targeting security against a computationally bounded adversary, you should target secrecy only against computationally efficient adversaries. And the second relaxation that you should make in your model is instead of demanding that he absolutely nothing about the underlying plaintext should be revealed.

You should be willing to let the adversary learn something about the underlying plaintext with some small error probability and that probability should be so small that for most practical purposes, you can ignore it off. Now, the challenges how exactly we mathematically define efficient adversaries, namely which algorithms, which adversarial algorithm, which you can say is an efficient adversarial algorithm?

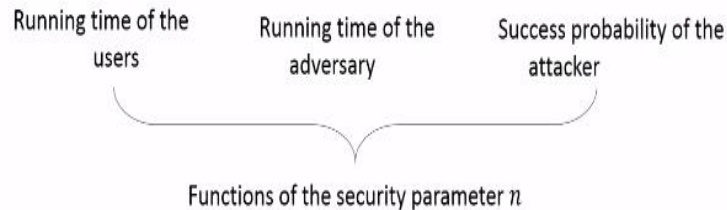
The second challenge here is which quantities you will define or you will call as a small quantity or a small error probability, right. So, what we are going to do here is we are going to mathematically define these two terms in asymptotic notation.

(Refer Slide Time: 17:53)

Defining Efficient Algorithms and Negligible Probability Asymptotically

□ Security parameter n --- publicly known (part of the scheme)

❖ Typically size of secret-key (ex: $n = 128, 256$, etc)



So, people who are familiar with the concept of algorithms, they will know this what exactly we mean by asymptotic notation. So, when I want to measure the running time of an algorithm, there are two approaches by which we can measure the running time of the algorithm. One is the concrete approach, where we compare two algorithms for the same goal based on the exact running time right. So, you have say algorithm 1 algorithm 2 for a task.

You run the algorithm 1 on samples of various size, you run algorithm 2 on samples of various size and then you compare the exact timings of the algorithm 1 and algorithm 2 two, of course over what processor you are given and so on. Based on that, you compare whether algorithm 1 is better or algorithm 2 is better, but the downfall of this approach is even though you get the concrete comparison of algorithm 1 versus algorithm 2, this approach does not take into consideration the underlying computing speed, the progress in the computing speed and so on.

The second approach that we follow in the world of algorithms to compare 2 algorithms is asymptotic notation, where we compare the running time of the 2 algorithms for solving the same task in asymptotic notations, namely in big O notation right, and we compare the running time by measuring the number of basic steps of algorithm 1 and the number of basic steps that algorithm 2 where the number of basic steps are computed as a function of the input size.

Depending upon which algorithm takes less number of steps, we define whether algorithm 1 is better or algorithm 2 is better and you have various asymptotic notations like big O notation, theta notation, omega notation based on which you can compare 2 algorithms. So, when we want to define what we mean by efficient, algorithm negligible probability in the context of cryptography, we are going to follow this latter approach, namely we are going to define these terms asymptotically.

For defining these terms asymptotically we have to introduce something what we call a security parameter which we denote by n . The reason we want introduce this security parameter is that once we introduced the security parameter n , then all the three quantities namely the running time of the users, namely the running time of the key generation algorithm, the running time of the encryption algorithm, the running time of the decryption algorithm.

Similarly the running time of the adversarial algorithm of the attacker, and the success probability of the attacker, all are going to be expressed as a function of the security parameter. Typically in the context of symmetric encryption process, the security parameter is the size of the secret key, which is typically for most practical purposes is 128, 256, and so on right.

(Refer Slide Time: 20:41)

Defining Efficient Algorithms Asymptotically

□ Efficient algorithms --- algorithms with a polynomial running time

- ❖ Algorithm A has a polynomial running time, if there exists a polynomial $p(\cdot)$, such for every input $x \in \{0, 1\}^*$, the computation of $A(x)$ terminates within $p(|x|)$ steps, where $|x|$ denotes the length of the string x

□ Requirement from any cipher (Gen, Enc, Dec)

- ❖ Gen, Enc and Dec should be efficient algorithms
- ❖ Running time of Gen, Enc and Dec should be a polynomial function of the security parameter n

So, let us first define what do we mean by efficient algorithms asymptotically. Informally, we say an algorithm is efficient if its running time is some polynomial function of the security parameter. Namely, if we have an algorithm A , then we say that algorithm A has polynomial

running time if there exists some polynomial function say p such that for every input x of size of cardinality this, the computation of the algorithm A on the input x dominates within polynomial in the number of size of x steps, right.

If that is the case, then we say that our algorithm A has polynomial running time and we will call our algorithm A to be an efficient algorithm. Whereas, if we cannot bound the running time of our algorithm A by some polynomial function in the size of its input, then we say that our algorithm is not efficient okay. That is the definition of efficient algorithm. Now, once we have defined a notion of efficient algorithm, the requirement that we put on any cipher is the following

So, remember we have the correctness requirement, we have the privacy requirement, and apart from that we have now the third requirement from any encryption process. The new requirement is that the running time of the key generation algorithm, encryption algorithm and decryption algorithm should be some polynomial function of this security parameter n . If the running time is not a polynomial function, then we would not consider that algorithm or cipher to be an efficient cipher.

(Refer Slide Time: 22:15)

Defining Negligible Probability Asymptotically

❑ Negligible functions --- functions which are asymptotically smaller than the inverse of every polynomial function

❖ Function $f(n)$ is a negligible function in n , if for every polynomial $p(n)$, there exists some N , such that $f(n) < \frac{1}{p(n)}$, for all $n > N$

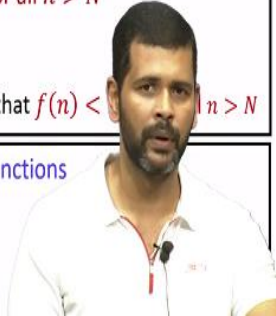
≈

❖ For every constant c , there exists some N , such that $f(n) < \frac{1}{n^c}$, for all $n > N$

❑ Example : 2^{-n} , $2^{-\sqrt{n}}$, $n^{-n \log n}$ are all negligible functions

❑ Function $1/n^{10}$ is not a negligible function

❖ $\frac{1}{n^{10}} \not< \frac{1}{n^{11}}$, for every value of n



Now, let us define the notion of negligible probability as a function of the security parameter. So informally, we say a function of the security parameter is negligible if it becomes almost 0 as the value of your security parameter n tends to infinity or the function of the security parameter will be called as a negligible function if it is asymptotically smaller than the inverse of every polynomial function.

Namely, if f of n is a function, then we will say that f of n is a negligible function if for every polynomial function p of n , there exists some value of n , namely big N , such that f of n is strictly less than the inverse of the polynomial function p of n for all values of n greater than big N , right. If this holds, then we say that our function f of n is a negligible function. Another equivalent definition of this negligible function is if the function f of n is strictly less than n^{-c} for every positive constant c .

Namely for every value of constant c , f of n is strictly less than n^{-c} for all values of n bigger than big N , then we say that our function f of n is a negligible function. The reason that these 2 definitions are equivalent is that any polynomial function p of n , you can always write it as some n^c constant c . So, if f of n is strictly less than the inverse of the polynomial function for every polynomial function, then I can recast it as f of n is strictly less than the inverse of n^c for every constant c right.

So, you can use any of these 2 definitions to prove or disprove whether a given function f of n is a negligible function in n or not. So, here, example a few functions which are all negligible functions. Each of these functions is strictly less than the inverse of every polynomial function, where the value of big N is different for the corresponding polynomial functions. So, even though all these functions are negligible functions, namely if I keep on making the value of small n to be large and as n tends to infinity, each of these candidate functions will become 0 eventually.

However, the rate at which each of these functions approaches to 0 is different, right. So, for instance if I consider the function 2^{-n} and if I consider the second function $2^{-\sqrt{n}}$, then definitely 2^{-n} will approach the value 0 faster compared to the value of the function $2^{-\sqrt{n}}$ and so on. On the other hand, if I consider the function $1/n^{10}$, then it is not a negligible function.

Because the requirement from a negligible function is that the function should be strictly less than the inverse of every polynomial function, but you can easily see that for no value of n , the function $1/n^{10}$ is less than $1/n^{11}$, that is not possible for every value of n . As a result, it violates the definition of negligible probability, right.

(Refer Slide Time: 25:34)

Negligible and Polynomial Functions : Closure Properties

- | |
|--|
| <ul style="list-style-type: none">□ Let $P_1(n)$ and $P_2(n)$ be two arbitrary polynomial functions. Then<ul style="list-style-type: none">➤ $P_1(n) + P_2(n)$, as well as $P_1(n) \times P_2(n)$ are polynomial functions |
| <ul style="list-style-type: none">□ Let $\text{negl}_1(n)$ and $\text{negl}_2(n)$ be two arbitrary negligible functions. Then<ul style="list-style-type: none">➤ $\text{negl}_1(n) + \text{negl}_2(n)$, as well as $P(n) \times \text{negl}_1(n)$ are negligible functions➤ No amplification of a negligible advantage<ul style="list-style-type: none">❖ Ex: Prob. that n fair coin-flips turn out to be $(0, \dots, 0) : 2^{-n}$ (negligible)❖ Even if the experiment repeated polynomial number of times, $(0, \dots, 0)$ will occur in any of these experiments with a negligible prob. |

So, we have defined mathematically what do we mean by efficient algorithm and we have defined which probability you can consider as a small probability. So now the family of negligible and polynomial functions satisfy some nice closure properties. So let us see the closure properties satisfied by the class of polynomial functions. So if you consider two arbitrary polynomial functions, say $P_1(n)$ and $P_2(n)$, then the summation of these two polynomial functions is also going to be a polynomial function, and in the same way, the product of these two polynomial functions is also going to be a polynomial function.

What is the implication of the first closure property? It says that suppose if you have two subroutines a way to interpret the first closure property is the following. Imagine you have two subroutines, where the running time of the first subroutine is some polynomial function in n and the running time of the second procedure is also some polynomial function in n , and if you have a bigger routine, which actually causes these sub-procedures in sequence, then the running time of the bigger algorithm is also going to be a polynomial function.

Namely, a bigger algorithm which causes these two smaller functions in sequence is also going to be considered as an efficient algorithm. That is what is the interpretation of the first closure property. In the same way, you can interpret the second closure property. Interestingly, the class of negligible functions also satisfies the certain closure properties. So, for instance, imagine you are given two arbitrary negligible functions, then it can be proved that the summation of two negligible functions is also going to be a negligible function and you can quickly prove it by contradiction

Assume on contrary that the summation of these two negligible functions is not a negligible functions, namely, the summation of these two functions is not strictly less than the inverse of some polynomial function say P of n , then you can end up showing that either of these two functions is also not a negligible function, which is a contradiction. In the same way, we can prove that if you have a negligible function negligible of $1/n$, then if you multiply the negligible function 1 with some polynomial function, then the result of one function is also going to be a negligible function, and again you can prove it by contradiction.

That means, on contrary, imagine this function which is a product of a polynomial function and a negligible function is not a negligible function, then you can end up showing that the function negligible 1 is actually not a negligible function, which is a contradiction, right. So this is a nice closure property, which is satisfied by the class of negligible function, and the interpretation of this latter property is that there is absolutely no amplification of a negligible advantage. What do I mean by that is the following.

You consider an experiment where I toss n fair coins and say I want to estimate that what is the probability that all the n coins give me the value 0 . Now, since each of the coin is a fair coin, the probability that all these n coins give me the value $0, 0, 0$ is 1 over 2 power n . It turns out that if I run this experiment independently polynomial number of times, then the event at in one of these versions of the experiment, I get the value of all the coins to be all 0 is going again to be a negligible probability.

That means this event is going to occur with very, very negligible probability, that is what is the interpretation of this second closure property that means if you consider it in the context of an adversarial advantage, that means if you have an algorithm and say the adversarial advantage of breaking that algorithm or breaking that experiment is negligible, and if that experiment is repeated polynomial number of times, then again there is absolute no advantage in the winning probability of the adversary, that is what is the interpretation of the second closure property, right.

(Refer Slide Time: 29:31)

Asymptotic Security in Practice

- Need to carefully select n while deploying a scheme, for meaningful security
 - ❖ Consider an encryption scheme, for which an adversary can break the scheme with prob. $2^{40} \cdot 2^{-n}$, by doing computations for n^3 minutes
 - ❖ the scheme is asymptotically secure, as $2^{40} \cdot 2^{-n}$ is negligible
- What value of n should be used while deploying the scheme in practice ?
 - ❖ $n = 40 \Rightarrow$ attacker's success probability will be 1, after doing computation for 40^3 minutes (6 weeks)
 - ❖ $n = 50 \Rightarrow$ attacker's success probability will be $1/1000$, after doing computation for 50^3 minutes (3 months)
 - ❖ $n = 500 \Rightarrow$ attacker's success probability will be 2^{-460} , after doing computation for 200 years

So, even though we have defined mathematically the notion of efficient algorithm, the notion of negligible probability asymptotically, you have to very carefully select the value of n when you are actually deploying a scheme to ensure that the resultant security guarantees that you get is indeed meaningful. What I mean by that is the following. Consider an arbitrary encryption scheme where it is ensured that any adversarial algorithm which runs for n cube minutes or say n cube steps, can break your scheme with probability $2^{40} \cdot 2^{-n}$.

Again, I have not mathematically defined what exactly I mean by break, but intuitively, by break you can understand that it can learn something about the underlying sets, say that is what is the meaning of break for the moment. So the guarantee that my algorithm is giving is, if any algorithm runs for n cube minutes to break it, the success probability of the algorithm is $2^{40} \cdot 2^{-n}$. Now in asymptotic notation, definitely, this is secure because the quantity $2^{40} \cdot 2^{-n}$ is negligible.

That is asymptotic guarantee you are getting. But when I want to deploy the scheme, in practice, I have to replace the value of n by some concrete value because as I said, the value of n is nothing but the size of the secret key. So what value of n you will use? Let us see some implications. If I substitute the value of n to be 40, then the guarantee that I get from this asymptotic notation is the following. Any adversarial algorithm who runs for 6 weeks, namely 40^3 minutes, which is approximately 6 weeks.

The success probability of that attacker's algorithm to break the scheme will be 1, namely 100%. So choosing a value of n equal to 40 is definitely useless because if I operate my encryption process with n equal to 40, then any adversary algorithm which runs for 6 week can break my scheme. So what I can do is, I can run the same algorithm with the larger value of key, say n equal to 50. Then the resultant security guarantee that I get is the following.

Any adversarial algorithm running for 50 cube minutes, which is approximately 3 months have the guarantee that it can break my scheme with probability 1 over 1000. Now, depends upon your underlying context whether 1 over 1000 is good enough security guarantee for you or not, right. If the success probability of 1 over 1000 is not a good security guarantee for you, and what you can do is you can further increase the size of key. Say you operate the same encryption process with a value of key which is equal to 500 bits, then the security guarantee that we obtain is the following.

Any adversarial algorithm running for 200 years has the success probability of 1 over 2 power 460 of attacking the scheme. How large or how small is this quantity, the quantity 1 over 2 power 460 is very very small. Namely, it is a number of seconds which has elapsed ever since the last big bang happened. That means, even though there is a chance that adversary can break your scheme, as you can see, it is so small that you can almost ignore it off if you operate the scheme with n equal to 500, right.

So, what is example demonstrates that when even though asymptotic security guarantee is what we will follow for the rest of the course, when you are actually deploying a scheme whose guarantee is given in terms of asymptotic notation, you have to very carefully choose the value of the security parameter to get a meaningful notion of security , right.

(Refer Slide Time: 33:17)

Asymptotic Security in Practice

(Slide courtesy : Arpita Patra)



User's running also increases 🤔

Adversary's job becomes harder 😊

min —————→ max
 n

So, this asymptotic security, you can imagine that the value of your security parameter is like a knob. As you keep on increasing the value of n , the security parameter, which is the size of the key and adversary's job becomes more and more and more hard, that means, it has to do more and more steps to break your scheme, and if there is a negligible probability that adversary can break your scheme and as you keep on increasing the value of n , then eventually the success probability of the attacker will become 0.

You should be very careful, you should not simply blindly fall increase the value of n because as you keep on increasing the value of n , the user's running time namely the running time of the key generation algorithm, the running time of the encryption algorithm, the running time of the decryption algorithm also increases because remember, the running time of each of these algorithms is also some function of the security parameter n .

So, you have a trade off, you cannot blindly increase the value of n when you are actually deploying a scheme, you have to very judiciously decide the value of your security parameter when you are actually deploying a scheme in practice. So that brings me to the end of this lecture. To just to summarize, in this lecture, we discussed that if key reusability is our ultimate goal, namely if we want to design a scheme where we want to retain the same key for encrypting multiple messages, then we have to make to relaxations to the model of perfect secrecy.

The first relaxation that we have to make is that instead of assuming that our adversary is computationally unbounded, we have to assume that our adversary is computationally

bounded and we have also seen that what do we mean by, how to measure whether the adversary's time is computationally bounded or not. In the same way, the second relaxation that we have to make in our model is instead of saying that adversary should not learn absolutely nothing about the underlying message, we should give the adversary some chance to break your scheme.

That some chance to break the scheme should be so small that for most practical purpose we can ignore it off. So, we have also seen how to mathematically define such a small probability of adversary breaking the scheme. We have seen that these 2 relaxations are kind of necessary evils for any encryption scheme where the goal is to achieve key reusability. Because if you do not make these 2 relaxations, then there are always 2 extreme attacks which are possible, namely the guessing attack and a brute force attack.

The success probability of the guessing attack will be very, very small, but the running time of this guessing attack will be practical, whereas the success probability of the brute force attack will be 100% but the running time of the brute force attack will be extremely large. So we have to definitely make these 2 relaxations. I hope you enjoyed this lecture. Thank you.