PES UNIVERSITY

Applied Cryptography

Unit-2 Symmetric key Cryptography

Lecture Notes 3:

Symmetric Key Cryptography- Computational Secrecy

Recommended Reading:

Katz-Lindell: Chapter 3:3.1

1. **Introduction:**

Last session gave us an idea about why perfect secrecy is not practically feasible and any description of the cryptosystem should be defined using the more practical version of security definition so called as Computational Secrecy.

Computational secrecy is something that is achieved by relaxing certain factors of perfect secrecy and the relaxation comes in the form of

1. Attacker is exposed to tiny probability of getting succeeded in decrypting the cipher.
2. Assuming the attacker is computationally limited.

First let us investigate if this relaxation would be beneficial.

1.1 Justifications that computational Secrecy is the right definition for security:

1. **Probability of Attack:**
   - Considering the probability of attack is somewhere in the range of 2^{-60} . This type of attack if calculated shows that it happens once in 100 billion years.
   - Do we really have to worry about an attack that is going to happen once in 100 billion years? Not really. So the tiny probability what we have mentioned here might always fall in this range and something value lesser than this. Not really we need to concern about these types of attacks that is going to happen once in a while.
2. **Computational Power:**
   - Coming to computationally restricted, how much time an attacker takes to decipher owing different computational power. Well, to get a sense of the kind of attackers we're going to consider,

- Well, to get a sense of the kind of attackers we're going to consider, let's look at an attacker who performs a brute force search over the key space. That is trying each key in the key space one by one, decrypting some observed cyber text with each possible key.
- And let's assume that the attacker can test one key per clock cycle of the underlying processor. This is an overly optimistic assumption, but we'll make it for simplicity in our calculations.
- If we have such an attacker running this attack on a typical desktop computer. Then that attacker can search through about 2 to the 57 keys per year. That's the number of clock cycles in a typical desktop computer if the computer is running over the course of a year. So this would mean that the attacker is taking their computer. Doing this brute-force key search, testing one key per clock cycle and using that computer for nothing else. If the attacker, instead, uses a supercomputer, they could test about 2 to the 80 keys per year. If that attacker was running an exhaustive search on a supercomputer since the time of the Big Bang.They could test about 2 to 112 keys in total.
- So, restricting our retention to attackers who can try at most 2 to 112 keys is a very reasonable assumption. We don't need to worry about attackers who can task 2 to 256 keys. Simply because there's no way that such an attacker can possibly exist in our physical universe. Just by way of comparison I wanted to let you know that in modern encryption schemes. The key space is at least 2 to the 128 keys or more. So this means that if you had an attacker running on a super computer since the time of the Big Bang. And doing nothing else except trying to find the key that you're using in such an encryption scheme. They still would not have been able to do an exhaustive search over the entire key space. That seems like a pretty good security assurance to me.

1.2<u>Perfect Indistinguishability:</u>

- To relax the definition of perfect secrecy, we're going to start with a definition of security called perfect indistinguishability.
- Let's fix an encryption scheme defined by three algorithms Gen, Enc, and Dec, as usual.
- And we'll refer to that encryption scheme by pi.

- Let's assume we have some two messages m0 and m1. And we choose one of those messages at random, and encrypt it using a uniform key to give us some ciphertext c. We give that ciphertext c to an attacker. And that attacker then tries to determine from that ciphertext which of the two messages, m0 or m1, was the one that was encrypted. We'll say that the scheme pi is secure. In the sense of perfect indistinguishability. If no adversary A can correctly guess whether of m0 or m1 one was encrypted with probability any better than one-half. More formally, let's let pi be an encryption scheme as before, and let A be an attacker, i.e, an eavesdropper. We're going to define a randomized experiment that we'll call PrivK for private key and that depends on both A and pi. It's an experiment that depends on some encryption scheme pi that we fix and some attacker A that we've also fixed.

The experiment proceeds as follows:

1. First, A gets to output any two messages m0 and m1 in the message space of its choice.
2. Then we generate a key using the key generation algorithm of the encryption scheme.
3. We choose a uniform bit B that's going to act as a selector bit selecting one of the two messages m0 or m1
4. . And the, we encrypt the message m sub busing the key k that we just generated. This gives us a ciphertext c. The ciphertext c here is going to be called a challenge ciphertext. We give the ciphertext to a. And A then outputs a bit, b prime, which is meant to represent its guess as to which of the two messages was encrypted.
5. We'll say that A succeeds in a given run of the experiment. If its guess, b prime, is equal to the value b corresponding to the message that was actually encrypted. And we'll define the experiment to evaluate to 1 or to output 1 if and only if A succeeds.
6. Notice that it's easy for any attacker A to succeed with probability one-half. If an attacker outputs a random guess for its bit b prime, it will be correct with probability exactly half. In fact, no matter what the attacker does if, if, it would be correct with probability one-half. It's easy to see that it's possible for an attacker to succeed with probability one-half. All

the attacker must do is to output a random guess b prime and it will then succeed with probability exactly half.

Well say if the encryption scheme pi is perfectly indistinguishable. If for all attackers A it holds that attacker succeeds with probability exactly half. There's no possible attacker a that can do any better than one-half. We claim that pi is perfectly indistinguishable if and only if it's perfectly secret. This means that perfect indistinguishability is just an alternate way of defining perfect secrecy. The claim is not very difficult to prove, but it's a bit technical and messy and so I'm not going to prove it here. You'll have to take me on faith. But the point is that we've defined this notion of perfect indistinguishability via an experiment that explicitly mentions an attacker A. That's different from what we had for the case of perfect secrecy.

But nevertheless, this definition that we get by defining that experiment, this definition of perfect indistinguishability. Is exactly equivalent to our prior definition of perfect secrecy. We're now going to define our notion of computational secrecy. By starting with our notion of perfect indistinguishability that we just defined and relaxing that. There are two approaches to relaxing that definition. One of which is called the concrete approach, or the concrete security approach. And the other of which is the asymptotic approach, or the asymptotic security approach.

1.3Concrete Approach:

Recall that to define perfect indistinguishability. We defined an experiment and said that an encryption scheme pi was perfectly indistinguishable. If for all attackers A, the probability that the attacker could succeed in the experiment was exactly one-half. In the concrete version of computational indistinguishability, we're going to relax that in the following way. Now we'll say that a scheme pi is t epsilon indistinguishable. If for all the attackers A running at time at most t. It holds that the probability that the attacker succeeds in the same experiment as before is at most one-half plus epsilon. Notice that we take into account both of the relaxations we've talked about previously. Right, we restrict our retention here, only to attackers A, running in time at most t. Rather allowing all attacker regardless of the running time is definition of perfect indistinguishability. And, moreover, we define the to be cured as for all such A. The probability that A can succeed in the experiment is at most one-half plus

epsilon. So, we're allowing an additional epsilon slack in the probability with which the attacker can succeed. There are a couple of nice advantages to this notion of concrete security.

2 First of all, the parameters t and epsilon correspond very closely with what we ultimately care about in the real world. In the real world, we have some assumed time bound with which our adversaries can run. Right?

3 So, going back to the example from earlier, we may be concerned with attackers who can run for time, at most, 100 million years. And we have this parameter Epsilon, which, which tells us, essentially, the probability with which security can fail. So, the attacker can succeed with probability one-half, just like in the case of perfect indistinguishability, plus this extra parameter epsilon. Which determines the little bit of slack with which we're allowing our attacker to succeed beyond what it can do. What it could do if our scheme were perfectly secret. So concrete security is very nice in that respect because it maps very cleanly on to things we ultimately care about in practice.

Nevertheless, there are some drawbacks to the concrete security approach. For one thing, the concrete security approach doesn't really lead to a clean theory. It's a little hard to go into the details of why that's the case. You may have a better sense of why this is the case after the, after you take the rest of this course. But one thing we can immediately observe is that the parameter t is very sensitive to the exact computational model. Right, if we measure t in terms of years, then we have to be concerned with what kind of computer the attacker is running on. Whether the attacker is running, computers in parallel. Whether the attacker is using, a particular operating system. Whether they're using a particular type of machine, a particular programming language. And all these details will impact ultimately the values of t that we care about. Now this is actually a problem that we must deal with in the real world. In the real world we have to worry about some particular attacker. And we may actually want to know what machine they're running on, what programming language they're using. And whether they're running sequentially or whether they're running a bunch of mis, machines in parallel. Nevertheless, from a theoretical point of view, it makes things rather messy. Furthermore, we'd like to have schemes where users are not bound to some particular choice of t and epsilon. But where they can instead adjust the

levels of security. That is adjust the parameters t and epsilon that the scheme can provide. To some levels that are comfortable for them. And we'll see that in the asymptotic notion of security we can actually achieve that.

1.4 Asymptotic Approach

For the asymptotic relaxation, we're going to do something a little bit more complicated. What we're going to do, is we're going to introduce the notion, of a security parameter n. The security parameter, is a positive integer, and we can view this as allowing the parties to choose the level of security, that they want for the scheme. And for now, you can view n, as denoting the key length. That is the length of the key that the parties share. The security parameter, or the key length, is going to be fixed by the honest parties, at the time, the system is initialized. That is at the time they choose and share their key. We're going to assume further, that the attacker, knows n. The attacker knows the security parameter. The attacker knows, the length of
the key that the parties are sharing. Now the important point, or the use of the security parameter, is that we're now going to view the running times of all parties. That is, the honest parties as well as the attacker, as well as the success probability of the adversary, as functions of n.

So we're going to look at how the running time of the parties changes, as n varies. And also look at the how the success probability of the attacker varies, as n is increased. So for computation indistinguishability, we're now going to relax the notion of perfect indistinguishability as follows. We're now going to allow security to fail, with probability negligible in n. And we're going to restrict our attention to attackers, running in time polynomial in n.

For the asymptotic relaxation, we're going to do something a little bit more complicated. What we're going to do, is we're going to introduce the notion, of a security parameter n. The security parameter, is a positive integer, and we can view this as allowing the parties to choose the level of security, that they want for the scheme. And for now, you can view n, as denoting the key length. That is the length of the key that the parties share. The security parameter, or the key length, is going to be fixed by the honest parties, at the time, the system is initialized. That is at the time they choose and share their key. We're going to

assume further, that the attacker, knows n. The attacker knows the security parameter. The attacker knows, the length of
the key that the parties are sharing. Now the important point, or the use of the security parameter, is that we're now going to view the running times of all parties. That is, the honest parties as well as the attacker, as well as the success probability of the adversary, as functions of n.

So we're going to look at how the running time of the parties changes, as n varies. And also look at the how the success probability of the attacker varies, as n is increased. So for computation indistinguishability, we're now going to relax the notion of perfect indistinguishability as follows. We're now going to allow security to fail, with probability negligible in n. And we're going to restrict our attention to attackers, running in time polynomial in n.