CS 4501-6501 Topics in Cryptography 16 Mar 2018

Lecture 8, Public-Key Cryptography and Key Agreement

Lecturer: Mahmoody *Scribe: Amar Singh, Jack Prescott, Xuyu Yi*

1 Introduction

Last time, we combined CPA security and MACs to achieve CCA security which provides security against active attacks. All of that was in the *private-key* setting. This time, we start by defining public key encryption schemes formally and key-agreement mechanisms.

2 Defining Public Key Encryption

Definition 2.1 (Public Key Encryption)**.** A public key encryption scheme $\mathcal{SKE} = (\text{Gen}, \text{Enc}, \text{Dec})$ (for all messages) consists of three algorithms:

- Gen is a randomized algorithm, that takes as input the security parameter 1^n and outputs a pair of keys (ek, dk).
  - ek is referred to as the "pubic key" (or encryption key)
  - dk is referred to as the "private key" (or decryption key)
- $\text{Enc}(ek, m, r)$, takes a generated ek, and arbitrary message m (and enough randomness r), and outputs a string c. This is written as $c \leftarrow \text{Enc}_{ek}(m)$, but note that denotes the process is potentially randomized.
- $\text{Dec}(dk, c)$, takes the decryption key dk and ciphertext x, and outputs a message m' or a symbol that denotes failure, $\perp$. This is written as $m := \text{Dec}_{dk}(c)$.

Completeness Requirement: For all m, with probability $1 - \text{negl}(n)$ over (pk, sk) output by $\text{Gen}(1^n)$ and the randomness of encryption ot holds that $\text{Dec}_{dk}(\text{Enc}_{ek}(m)) = m$. In other words, with overwhelming probability we do get back the message that is encysted. In fact, most encryption algorithms decrypt correctly *all* the time, so we might as well define completeness requirement to hold with probability one.

In the following we will focus on defining the security.

2.1 CPA Security

Public key encryption can be defined to have similar security properties like indistinguishability (for one message or multiple massages) or CPA security or even CCA security. Interestingly, it can have even a simpler definition of CPA security that happens to be equivalent to indistinguishability for one message, due to the fact that the encryption key is *public*. First, let's see what an indistinguishability security game would be.

Modified definition of the eavesdropping indistinguishability experiment $\mathrm{PubK}_{\mathcal{A},\Pi}^{EAV}(n)$: for Public Key Encryption

- $\mathrm{Gen}(1^n)$ is run, and keys (ek, dk) are returned.
- Adversary $\mathcal{A}$ is given pk and outputs a pair of messages (m_0, m_1) of equal length.
- A bit $b \in \{0,1\}$ is chosen at random, and a cipher text $c \leftarrow \mathrm{Enc}_{ek}(m_b)$ is given to $\mathcal{A}$.
- $\mathcal{A}$ outputs a bit b'. The output of the experiment is 1 if $b' = b$, otherwise it is 0. In this case, we say that $\mathcal{A}$ succeeds.

Definition 2.2 (Indistinguishable encryptions)**.** A public key encryption scheme Π has indistinguishable encryptions to an eavesdropper if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there is a negligible function $\mathrm{negl}(\cdot)$ s.t.

$$\Pr[\mathrm{PubK}_{\mathcal{A},\Pi}^{EAV}(n) = 1] \leq \tfrac{1}{2} + \mathrm{negl}(n)$$

But something that we can notice right away, is that even the above seemingly "weaker" notion of security, when it comes to *public* key encryption, *cannot* be deterministic. Namely: **No deterministic public-key encryption is indistinguishable secure.** The reason is that the adversary also has the public key, like everyone does. So, if the scheme is deterministic, the adversary can encrypt both of m_0, m_1 himself, and compare them with the challenge ciphertext and find out which one is encrypted. In fact, what is going one is even more deep: the above definition is already equivalent to CPA security, because the only difference between the two definitions is the existence of the encryption oracle $\mathrm{Enc}_{pk}(\cdot)$, which is available to the adversary already: she can encrypt any messages herself!

Definition 2.3 (CPA secure public key encryptions)**.** A public key encryption scheme Π is CPA secure if and only if it is single-message indistinguishable encryptions. So the definition is simpler, but it already carries the heavier weight of guaranteeing CPA security.

As it was the case of private key encryption, CPA security already implies multi-message security as well, so all definitions become the same and equivalent in case of public-key encryption. However, CCA security is still a stronger notion.

2.2 CCA Security

Definition of the CCA indistinguishability experiment $PubK^{cca}_{\mathcal{A},\Pi}(n)$:

- $\text{Gen}(1^n)$ is run, and keys (ek, dk) are returned.
- Adversary $\mathcal{A}$ is given pk and it has access to a decryption oracle $\text{Dec}_{dk}(.)$ It then outputs a pair of messages (m_0, m_1) of equal length.
- A bit $b \in \{0, 1\}$ is chosen at random, and a cipher text $c \leftarrow \text{Enc}_{ek}(m_b)$ is given to $\mathcal{A}$.
- $\mathcal{A}$ continues to interact with the $\text{Dec}_{dk}(\cdot)$ oracle, but cannot request the decryption of c. When they are ready, $\mathcal{A}$ outputs a bit b'.
- $\mathcal{A}$ outputs a bit b'. The output of the experiment is 1 if $b' = b$, otherwise it is 0. In this case, we say that $\mathcal{A}$ succeeds.

Definition 2.4 (CCA Security)**.** A public key encryption scheme Π has CCA Security if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there is a negligible function negl s.t.

$$\Pr[\text{PubK}^{CCA}_{\mathcal{A},\Pi}(n) = 1] \leq \tfrac{1}{2} + \text{negl}(n)$$

We will see RSA encryption later on, which can be combined with "ideal hash" functions to achieve CPA or even CCA secure encryption.

3 Key Agreement Protocols and Their Security

Now that we know the definition of public key encryption, we can talk key agreement protocols as a relaxation of public-key encyrption. Public key encryptio nallows two parties to build a secure channel over a public channel. Key agreement allows doing so by first agreeing on a random key that is hidden from an eavesdropper, and then we can combine it with previously developed privat key encryption schemes to do the same thing. However, the number of messages exchanged here could potentially be larger.

3.1 What is a Key Agreement Protocol?

Intuitively, the public key encryption is aiming at a picture that Bob (or any party) sends a single mesasge to everyone, that is the *encryption key ek* and then gets back an encrypted message c, which is called the cipher-text. Then he uses the decryption key, which he kept secret, to obtain the message m inside c.

Key agreement also tries to achieve the same thing by also relies on the private key setting with more messages exchanged. A weaker goal of key agreement can be agreeing on a random key between Alice and Bob. If a random key can be agreed over a public channel, then all the information we had under the private key setting can be used in this case.

Suppose Alice and Bob are exchanging not just two arbitrary messages but messages with *local* (private) randomness. They come up, at the end, with their own keys k_A, k_B, which have two properties: completeness and security.

Completeness: The probability of $k_A = k_B$ is 1, which means the two keys are equal.

Security Intuition: Suppose we have very good encryption and decryption algorithms that need a private key. A good key is then possible to be plugged into the private key encryption algorithm, and nothing would be revealed about the messages when using it. The informal goal: if we use key k to encrypt and decrypt the messages through any CPA-secure private-key encryption scheme, the combined scheme shall remain "secure". The key point is that this should happen even if the adversary knows the transcript of the interaction between Alie and Bob, namely the messages that they exchanged to agree on a key. We want the actual key to be pseudorandom even if Eve knows the transcript T, namely even if we show the adversary Eve the true key k after seeing T, he can not distinguish it from a completely random key U_n.

3.2 Security of Key Agreement

Security Game (two worlds). Suppose there are two worlds. In world 1, the adversary is given the transcript, which is all messages we are sending, and a completely random "key" U_n of the same length n that is the length of k. In world 2, the adversary is given the transcript $T = (t_1, \dots, t_k)$ and the actual key($k = k_A = k_B$). The security requires that adversary cannot tell these two apart.

Definition 3.1 (Security of key agreement)**.** A key agreement is secure if any polynomial time adversary cannot distinguish the above two worlds by more than negligible.

Theorem 3.2 (Informally stated)**.** *If we use a secure key agreement and a CPA-secure-secret key encryption, even though the adversary knows about the transcript and cipher-text, he is not able to tell the encrypted message. The reason is that the key would be at least pseudorandom (i.e. indistinguishable from truly random) even after knowing the public transcript, so it can substitute a truely random key for private-key encryption schemes as well.*

4 Key Agreement Mechanism Examples

4.1 Number Theory 101: Modular Computation

Definition 4.1 (Group)**.** A group is a set G with an operation $*$ such that for any two elements $a, b \in G$, $ab \in G$. Specifically, the set and operation $(G, *)$ must satisfy the group axioms:

1. Closure. $\forall\ a, b \in G$, $ab \in G$
2. Associativity. $\forall\ a, b, c \in G$, $(a * b) * c = a * (b * c)$
3. Identity Element. $\exists!\ e \in G$ s.t. $\forall\ a \in G$, $e * a = a * e = a$.

4. Inverse Element. $\forall\ a \in G\ \exists\ b \in G$ such that $a * b = b * a = e$ where e is the identity element.

For any n, we call the set $\mathbb{Z}_n$, the set $\{0, \ldots, n-1\}$ with addition operation module n. Namely, $(a+b)$ is their addition mod n, which is in the same set $\{0, \ldots, n-1\}$. It is easy to see that $\mathbb{Z}_n$ is a group under the binary operation $+$ (mod n). The identity element here is 0

Interestingly, if $n = q$ is a prime, then if we remove 0 from $\mathbb{Z}_n$, call it $\mathbb{Z}_n^*$, and consider $*$ to be multiplication mod n, then again $\mathbb{Z}_q^*$ becomes a group (proving the existence of inverse for all nonzero elements is nontrivial. Namely, for all $a \in \{1, \ldots, q-1\}$ there is b, such that $ab = ba = 1 \bmod q$).

Definition 4.2 (Generator)**.** An element $g \in G$ generates the group $(G, *)$ if repeated applications of g on itself (with the operation $*$) produces all the elements in the group. More formally, if $g^k = e$, then the elements in G are $\{g^1, ..., g^k = e\}$.

Theorem 4.3. *For the group $\mathbb{Z}_q^*$, q is prime $\implies \exists\ g \in \mathbb{Z}_q$ s.t. g is a generator.*

For simplicity (and abuse of terminology) we call g simply a multiplicative generator for $\mathbb{Z}_q$ (even though it is technically a generator for the group $(\mathbb{Z}_q^*, *)$.

4.2 Diffie-Hellman Key Agreement

This protocol can be summarized in 5 simple steps:

1. Alice and Bob agree on a finite group $(G, *)$ of order n such that $g \in G$ generates G. In particular, they can use $G = \mathbb{Z}_q^*$ and a generator for it, which as we described above always exists. So, below we will work with this simpler case. Note that the adversary will know g and G. These are public parameters and part of the description of the protocol, and might be even universally fixed as part of the standard of the protocol. Note that the *length* of representation of q is the security parameter, so we have $n \approx \log q$ as the security parameter.

2. Alice picks a number $x \in \{1, \ldots, q-1\}$ at random and sends $a = g^x$ to Bob.

3. Bob picks a number $y \in \{1, \ldots, q-1\}$ at random and sends $b = g^y$ to Alice.

4. Alice gets $b^x = g^{xy}$ as her key, and Bob takes $a^y = g^{xy}$ as his key. Note that the keys would be the same.

Both Alice and Bob can now use the group element g^{xy} as the shared secret key. The group G satisfies the condition for secure communication as long as there does not exist an efficient algorithm for determining g^{xy} given g, g^x, and g^y. This is tightly related to the the hardness of the discrete logarithm problem.

Definition 4.4 (Discrete Log Problem (DLP))**.** Given q, g and $g^x = a \bmod q$, find x. Put differently, compute $\log_x(a)$ in the group $\mathbb{Z}_q^*$.

Definition 4.5 ((Computational) Diffie-Hellman Problem (CDH))**.** Let p be a prime number and g a generator for $\mathbb{Z}_q^*$. The Computational Diffie-Hellman Problem (CDH) is the problem of computing the value of g^{xy} from the known values of g^x and g^y. The conjecture is that this problem is hard and any polytime adversary has negligible chance of doing so for sufficiently large and carefully chosen q, g.

Theorem 4.6. *If the discrete logarithm problem is solvable in polynomial time $\implies$ Diffie-Hellman key exchange is not secure, because CDH becomes easy to solve.*

Proof. If Even can solve the DLP in polynomial time, then she can also compute Alice and Bob's secret exponents x and y from the intercepted values $a = g^x$ and $b = g^y$. It directly follows that it is easy for Eve to compute their shared key g^{ab}. $\square$

The converse is less clear. If Eve has an algorithm that efficiently solve CDH, it is not known whether she can also efficiently solve the discrete log problem.

Efficiency of implementing Diffie Hellman. Note that if we do the exponentiation in the Diffie-Hellaman protocol naively, it would not be efficient at all. In particular, q is a huge number, maybe hundreds of bits, and thus it can be as large as 2^{100}, for such q, x is also a number that could be as large as q, and thus computing g^x could be tricky:

1. Worse way: if we keep multiplying g by itself x times before computing the whole answer mod q, the actual number g^x (without taking mod q) can have trillions of digits, and not even efficient to write it down, let alone do computation on top of it. In particular, g^x could be 2^{100} *digits*, and so as large as $2^{2^{100}}$ itself.

2. Better way: a better, yet not efficient enough, way to do this is to keep multiplying g by itself x times, by simplifying the answer to be mod q after every multiplication. Namely, after the i'th iteration, a variable c would contain $g^i \bmod q$, and then we compute $c \cdot g \bmod q$ to obtain $g^{i+1} \bmod q$. This approach at least does not have the terrible representation length issue that we had before, but still it can take $x \approx 2^{100}$ iterations till getting the solution.

3. Best solution: fast exponentiation. The good news is that we can compute the exponentiation g^x using $O(\log x)$ multiplications. First, note that we can compute $g, g^2, g^4, \ldots, g^{2^k}$ by only k multiplications. Every time, simply multiply the last element with itself to get the next one in this list. Finally, if $2^k > x$, which happens for $k \leq O(\log x)$, we can then pick a subset of $g, g^2, g^4, \ldots, g^{2^k}$, multiply them and get back g^x. This can be done by using the binary representation of $x =$ in basis 2.