Personal Notes on *A graduate course in applied cryptography*

June 22, 2023

Contents

Chapter 1

Notes on Ch. 10

Topic: Introduce tools use used in public-key cryptography, digital signatures, and key exchange.

To begin let us start with a toy problem: Generating a shared secret key between two parties, so that a passive eavesdropping adversary cannot guess the shared key. The adversary only has the ability to lisen, not to generate or alter messages that are sent.

Let us define a key exchange protocol P as a pair of probalstic machines A, B where each take turns sending messages to each other. At the end of the communication period a key k is obtained that is the same between the two.

A protocol transcript T_P is the sequence of messages sent between A, B during a single execution of P. Note A, B are probalistic machines and thus a different transcript is obtained each time. In terms of the adversary, it is important to note that the adversary $\mathcal{A}$ also obtains T_P and thus in the case of security againsts an eavesdropper, our adversary can interpret these messages, but it should not be able to gain information about k from this transcript.

A briefly more formal definition of T_P as a random variable whose a function of the random bits generated by A, B. We again note that our adversary sees this entire transcript with the goal of obtaining the secret key k.

Attack Game 1.0.1. *(Anonymous key exchange) For a key exchange $P = (A, B)$ and a adversary $\mathcal{A}$ the attack game runs as followed:*

1. *The protocol is ran between A and B to generate a shared private key k and a transcript T_P which is also obtained by $\mathcal{A}$.*
2. *$\mathcal{A}$ outputs a guess $\hat{k}$ for k*

We define $\mathcal{A}$ or $AnonKEadv[\mathcal{A}, P]$ as the probability that $\hat{k} = k$.

Definition 1.0.1. *A anonymous key exchange protocol P is known to be secure against an eavesdropper if for all efficient (polynomial times) adversaries $\mathcal{A}$ their advantage is negligible.*

Remark 1. *Why is this a weak form of security?*

1. *Assume the adversary is unable to tamper with the message*

2. *We assume that $\mathcal{A}$ cannot guess the entire bit of k not that it might not learn some information about it.*

3. *The protocol does not guarantee the identity of either A or B.*

1.1 One-Way Trapdoor Functions

We define a one way function as a function $F : X \to Y$ that is easy to compute but hard to convert. A trapdoor is a secret that allows one to efficiently invert the function, but if one does not know this secret the function is hard to invert.

Definition 1.1.1. *(Trapdoor function scheme) Let X and Y be finite sets, We then define a trapdoor function scheme T as defined over (X, Y) as a triple of algorithms (G, F, I) where:*

- *G is a probalistic key generation algorithm that is invoked as $(pk, sk) \leftarrow G()$ where pk is referred to the public key and sk as the secret key*
- *F is a deterministic algorithm that is invoked as $y \leftarrow F(pk, x)$ where again pk is the public key and $x \in X$. The output $y \in Y$.*
- *I is a deterministic algorithm that is invoked as $x \leftarrow I(sk, y)$ where sk is the secret key and $y \in Y$. We say that $x \in X$.*

Lastly, for all outputs pk, sk of $G()$ and for all $x \in X$ we have that $I(sk, F(pk, x)) = x$.

We note that this last property ensures that $F(pk, \dot{)}$ is an injective function. Moreover, when $X = Y$ this function is then also surjective, as it is merely a permutation on the set. Due to this, when $X = Y$ we refer to this as a trapdoor permutation scheme.

Attack Game 1.1.1. *(One way trapdoor function scheme)*

- *The challenger computes*

$$(pk, sk) \leftarrow G(), x \leftarrow\!\!\$\ X, y \leftarrow F(pk, x)$$

- *The adversary outputs $\hat{x} \in X$*

The adversary's advantage in inverting T denoted $OWadv[\mathcal{A}, T]$ is the probability that $\hat{x} = x$

Definition 1.1.2. *A trapdoor function scheme T is one way if for all efficient (polynomial time) adversaries $\mathcal{A}$, their advantage is negligible.*

A natural next question in the perspective of key exchange is how could we use trapdoor one way functions for key exchange. We can define a protocol as followed:

- Alice computes $(pk, sk) \leftarrow G()$ and sends pk to bob.
- After receiving pk bob computes for some $x \leftarrow\!\!\$\ x$, $y \leftarrow F(pk, x)$ and then sends this y to alice
- After receiving y alice computes $x \leftarrow I(sk, y)$

In this case we see that the shared secret key is x.

1.2 A trapdoor permutation scheme based on RSA

As of now, RSA (or close alternatives) is one of the only canidates for a trapdoor permutation scheme.

Let us first briefly recall a probalistic algorithm RSAgen, given some $l > 2$ and odd integer $e > 2$

We say that $RSAGen(l, e) =$

- generate a random l-bit prime p such that $gcd(e, p-1) = 1$
- generate a random l-bit prime q such that $gcd(e, q-1) = 1$ and $q \neq p$
- $n \leftarrow pq$
- $d \leftarrow e^{-1} \mod (p-1)(q-1)$

output (n,d)

Given $RSAGen$ we can now define our RSA trapdoor permutation scheme $T_{RSA} = (G, F, I)$ which is parameterized by fixed values l, e.

- We see that key generation follows as $G() \coloneqq (n, d) \leftarrow\!\!\$\ RSAGen(l, e), pk \leftarrow (n, e) sk \leftarrow (n, d)$
- For a given public key $pk = (n, e)$ and $x \in \mathbb{Z}_n$ we define $F(pk, x) \coloneqq x^e \in \mathbb{Z}_n$
- For a given secret key $sk = (n, d)$ and $y \in \mathbb{Z}_n$ we define $I(sk, y) \coloneqq y^d \in \mathbb{Z}_n$

Theorem 1.2.1. *Let $n = pq$ where p and q are distinct primes. Let e and d be integers such that $ed \equiv 1 \mod (p-1)(q-1))$. Then for all $x \in \mathbb{Z}$ we have $x^{ed} \equiv x \mod n$.*

It is important to note that factoring n is believed to be hard, provided that l is sufficently large. This generally means we chose l to be 1000 to 1500. Additionaly, the only known algorithm to invert T_{RSA} is to first factor n and then compute d. Note that this has not been proved but based on current evidence it is belied that T_{RSA} is one way.

Attack Game 1.2.1. *(RSA) For a given integers $l > 2$ and odd integers $e > 2$ and an adversary $\mathcal{A}$, an attack game goes as:*

- *Challenger and adversary take (l, e) as in put.*
- *Challengers compute $(n, d) \leftarrow\!\!\$\ RSAGEN(l, e), x \leftarrow\!\!\$\ \mathbb{Z}_n, y \leftarrow x^e \in \mathbb{Z}_n$ and sends (n, y) to the adversary.*
- *Adversary outputs $\hat{x} \in \mathbb{Z}_n$*

THe adversary advantage in breaking RSA is the probability that $\hat{x} = x$.

Note n is often called the RSA modulus, e the encryption exponent, d the decryption exponent, (n, y) an instance of the RSA problem, and x a solution to the problem.

Given RSA as defined above let us now consider key exchange utilizing RSA:

The protocol runs as followed:

1. Alice computes $(n, d) \leftarrow\!\!\$\ RSAGEN(l, e)$ and sends (n, e) to bob.
2. Bob computes $x \leftarrow\!\!\$\ Z_n, y \leftarrow x^e$and sends y to alice.
3. Given y, Alice computes $x \leftarrow y^d$

The secret shared by Alice and Bob is x.

1.3 Diffie-Hellman Key Exchange

Let us outline a protocol for key exchange utilizing Diffie-hellman security notations:

First we assume we have some group $\mathbb{G}$ as well as a generator g within it, whose order is q that was generated once for the entire system setup.

1. Alice compute $\alpha \leftarrow\!\!\$\ Z_q$, $u \leftarrow g^\alpha$ and sends u to Bob.
2. Bob computes $\beta \leftarrow\!\!\$\ \mathbb{Z}_q$ and sends $v \leftarrow g^\beta$ to Alice.
3. Alice computes now $w \leftarrow v^\alpha$
4. Bob computes now $w \leftarrow u^\beta$

We see that $w = v^\alpha = g^{\alpha\beta} = u^\beta$

We note that the discrete logarithm is not enough enough to ensure that this is hard, instead it must hold that given $g^\alpha, g^\beta \in \mathbb{G}$ where $\alpha, \beta \in \mathbb{Z}_q$ it hard to compute $g^{\alpha\beta} \in \mathbb{G}$. This is known as computational Diffie-Hellman assumption.

Attack Game 1.3.1. *(Discrete Logarithm) Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in G$. For an adversary $\mathcal{A}$ the attack game is as followed:*

1. *The challenger and the adversary $\mathcal{A}$ takes $\mathbb{G}$ as input. This includes the order q and the generator $g \in \mathbb{G}$.*
2. *Challenges compute $\alpha \leftarrow\!\!\$\ \mathbb{Z}_q u \leftarrow g^\alpha$ and sends $u \in \mathbb{G}$ to the adversary.*
3. *The adversary outputs $\hat{\alpha} \in \mathbb{Z}_q$*

$\mathcal{A}$ advantage in solving the discrete logarithm problem for $\mathbb{G}$ is the probability that $\hat{\alpha} = \alpha$.

Attack Game 1.3.2. *(Computational Diffie-Hellman) Let $\mathbb{G}$ be a cyclic group of prime order q generated by a generator $g \in \mathbb{G}$. For a given adversary $\mathcal{A}$ the attack game is as followed.*

1. *The challenger and adversary take $\mathbb{G}$ as input along with q and the generate $g \in \mathbb{G}$.*
2. *The challenger computes $\alpha \leftarrow\!\!\$\ \mathbb{Z}_q$, $u \in g^\alpha$, $v \in g^\beta, w \leftarrow g^{\alpha\beta}$. and sends the pair (u, v) to the adversary.*
3. *The adversary outputs $\hat{w} \in \mathbb{G}$.*

The advantage of the adversary in solving the computational diffie-hellman problem for $\mathbb{G}$ is the probability that $\hat{w} = w$.

Attack Game 1.3.3. *(Decisional Diffie-Hellman) Let $\mathbb{G}$ by a cyclic group of prime order q with a generator $g \in \mathbb{G}$ and for an adversary $\mathcal{A}$ let us define an attack game as following:*

1. *Challenger and adversary takes $\mathbb{G}$ as input along with the generator g and order q.*
2. *Challenger compute $\alpha\beta\gamma \leftarrow\!\!\$\ \mathbb{Z}_q$, $u \leftarrow g^\alpha, v \leftarrow g^\beta, w_0 \leftarrow g^{\alpha\beta}, w_1 \leftarrow g^\gamma$*
3. *The adversary outputs a bit $b \in \{0, 1\}$*

Therefore for the event that the adversary choses their advantage is the difference between this and w_0 or w_1.

Note that one important property of the discrete log function for a group $\mathbb{G}$ is that it is either hard almost everywhere or almost nowhere. A middle ground where it is easy for some and not for others is NOT possible. This is due to a property called random self-reduction which can be shown to be true for the discrete log function.

Theorem 1.3.1. *Consider a specific cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$. Suppose that $\mathcal{A}$ is a efficient algorithm with the property that if $u \in \mathbb{G}$ is chosen at random then $\Pr[\mathcal{A}(u) = Dlog_q(u) = \epsilon]$ where the probability is over the random choice of u and the random chocies made by $\mathcal{A}$. Then there is an efficient algorithm B, such that for all $u \in \mathbb{G}$, B eithers outputs fail or $Dlog(u)$ and it output the latter with probability e but now is only over the random choices of B.*

Note for a problem with a random self-reduction, if it is hard in the worst case then it is hard on average an important and desirable property for a cryptosystem.

1.4 Attacks on the anonymous Diffie-hellman protocol

.

An attack known as the man in the middle attack works against any key exchange protocol that does not include some form of authentication for its users. It works as followed:

- Alice sends (g, g^α) to Bob. The attack blocks this message from reaching bob, He picks a random $\alpha' \leftarrow\$ \mathbb{Z}_n$ and sends $(g, g^{\alpha'})$ to Bob instead.
- Bob responds with g^β. The attack again blocks this message from reaching alice, picks a random $\beta' \leftarrow\$ \mathbb{Z}_n$ and send $g^{\beta'}$ to alice.
- Now Alice computes key $k_A \coloneqq g^{\alpha\beta'}$ and bob computes $k_B \coloneqq g^{\alpha'\beta}$. The attack knows both k_A and k_B.

Both parties think they shared the key with each other but infact the attack is the only one that knows both keys, as long as the attack continues this attack of blocking messages, it is possible for them to stay undetected allowing communication between bob and alice without them realizing there is a middle man who is interpreting and possibly editing their messages.

1.5 Accumulators

One such way Alice could commit to an order tuple of elements $x_1, \ldots x_n$ and later prove to bob that she knows one value x_i at position i in this set is via a merkle tree. Where each proof can be done as a sequence of $\log_2 n$ hashes. Another way to do this is via an accumulator. This allows alice to commit to an unorder set $S = \{x_1, \ldots x_n\} \subseteq X$ so that she can later prove to Bob that a particular $x \in X$ is either in the set S or not. We call this commitment scheme for unordered sets that allow us to tell if a element is part of a set or not a accumulator. Moreover, a dynamic accumulator is one where we can add elements to the set S. Therefore there is some algorithm that takes as input a commitment to a set S along with an element not in S and outputs a commitment to a new set S' which is the union of the two.

To create a simple dynamic accumulator we use groups of unknown order or (GUO). GUO is a pair $(\mathbb{G}, g)$ where $\mathbb{G}$ is a description of an abelian group and g is an element of g where the size of $\mathbb{G}$ is difficult to deduce from the pair. These can be generated by probabilistic group generation algorithm called $GGen$ that when invokes outputs a pair $(\mathbb{G}, g)$.

In terms of GGen, we say it satisfies strong RSA assumption, where the game for an adversary $\mathcal{A}$ is described as followed. We fun $(\mathbb{G}, g) \leftarrow\$ GGen()$ and send $(\mathbb{G}, g)$ to $\mathcal{A}$. $\mathcal{A}$ outputs (x, e) and wins the game if $x^e = g$ and $e \neq p_{m+1}$. We say that GGen satisfies the strong RSA assumption if no efficient adversary can win the strong RSA game with respect to GGen with non-neglible advantage.

There are a number of candidates for GUO where the strong RSA assumption is believed to hold. One such is the group $\mathbb{Z}_n^\star$ of integers modulo $n = pq$ where $n \leftarrow RSAGEN(l, e)$ for some , e.

Given we have some GUO $(\mathbb{G}, g)$ that is already known for all parties, let us show a construction for an accumulator. We treat $(\mathbb{G}, g)$ as public parameters, additionally we need a collision resistant hash function $H : X \to Primes(L)$ where Primes(L) is the set of the smallest L primes. We need that L to be sufficiently large so that H can be collision resistant.

A GUO accumulator works as follows:

- Alice commits to an unorder set $S = \{x_1, \ldots x_n\} \in X$ where $C(S) \coloneqq e_i \leftarrow H(x_i)$ and $E \leftarrow e_1 \cdot \ldots \cdot e_n \in \mathbb{Z}_n$ we then output $c = g^E$.
- A membership proof for $x \in X$ with respect to c. Alice has (S, x), Bob has (c, x) and Alice wants to convince Bob that x is in S. If $x \in S$ then $\beta = \frac{E}{H(x)}$ is an integer. Alice can prove to bob that x is in S by computing $b \leftarrow g^\beta = g^{\frac{E}{H(x)}}$ and outputting the short proof $\pi_x = b$. Bob verifies the proof $\pi_x = b$ by checking that $b^{H(x)} = c$.
- A non membership proof that for $x \in X$ with respect to c Alice has (S, x) and bob has (c, x), and alice wants to convince bob that x is not in S. Therefore we see that $x \notin S$ that E and $H(x)$ must be relatively prime. A proof of such could be computed by alice and then verified by bob.

In terms of the security of a accumulator, we say it is secure if no efficient adversary can simultaneously prove that $x \in S$ and $x \notin S$ for some commited set S and x. Thus they can not generate a proof of membership and non-membership efficiently .

One other factor of GUO accumulator proofs over Merkle tree proofs is that they can be aggregated together.

Chapter 2

Notes on Ch. 11

In this chapter, the basic problem of encryption is considered, consider that Alice wants to send Bob an encrypted email message, even if they don't share a secret key. This in fact can be done via public key encryption.

Public Key encryption is used in many settings, highlighted below are two of them:

1. **Sharing Encrypted Files** How it works: Alice encrypts a file f with a random key k using a symmetric cipher. The resulting cipher text c_f is stored on the file system. Suppose Alice wants to grant Bob access to the contents of f, she encrypts k under Bob's public pk_b by computing $c_B \leftarrow\!\!\$\ E(\mathsf{pk}_B, k)$. c_B is then stored near c_f as part of the meta data. Now when Bob wants to read f he decrypts C_B using his secret key sk_B to obtain k, then decrypt c_f using k to obtain f. This scheme could be easily extended to other uses, only a single copy of f needs to be stored, and just the seperate metadata is needed.

2. **Key Escrow** Consider a company who wants to be able to open a employees encrypted file when they are unable to provide the code themselves, if the company was using a key escrow this could be done. In this case the company runs a key escrow server where at setup the key escrow server generates a secret key sk_{ES} and a corresponding pk_{ES}. It keeps the secret key to itself and makes the public key available to all employees. When Alice (an employee) then encrypts a file using a random symmetric key k she also encrypts k under pk_{ES} and stores the ciphertext C_{ES} in the meta data section. All files are encrypted this way. Therefore if a manager needs to access some work and Alice is not reachable they could send c_{ES} to the escrow service, the server would decrypted C_{ES} to obtain k and send k to the manager. This could then be used to decrypt c_f and obtain f.

2.1 Basic Definitions

A public key encryption scheme $\mathcal{E} = (G, E, D)$ is a triple of efficient algorithms; a key generated G an encryption algorithm E and a decryption algorithm D.

1. G is a probalistic algorithm that is invoked as $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ G()$ where pk is the public key and sk is the secret key.

2. E is a probalistic algorithm that is invoked as $c \leftarrow\!\!\$\ E(\mathsf{pk}, m)$ where pk is the public key,m is the message, and c is the cipher tex.

3. D is a deterministic algorithm that is invoked as $m \leftarrow D(\mathsf{sk}, c)$ where sk is the secret key, c is the cipher tex and m is the message

4. We require that decryption undoes encryption specifically that for all possible outputs pk, sk of G and all messages that

$$\Pr[D(\mathsf{sk}, E(\mathsf{pk}, m)) = m] = 1$$

Note: Messages are assumed to lie in some finite message space $\mathcal{C}$ and ciphertex in some ciphertext space $\mathcal{C}$. We say that $\mathcal{E} = (G, E, D)$ is defined over $(\mathcal{M}, \mathcal{C})$.

Attack Game 2.1.1. *(semantic security) For a given public-key encryption scheme $\mathcal{E} = (G, E, D)$ defined over $(\mathcal{M}, \mathcal{C})$ and given adversary $\mathcal{A}$ we define two experiments: Experiment b $(b = 0, 1)$:*

1. *Challenger computes $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ G()$ and sends pk to adversary*

2. *Adversary computes $m_0, m_1 \in \mathcal{M}$ of the same length and sends them to the challenger.*

3. *Challenger computes $c \leftarrow\!\!\$\ E(\mathsf{pk}m_b)$ and sends c to adversary. The adversasry outputs a bit $b \in \{0, 1\}$.*

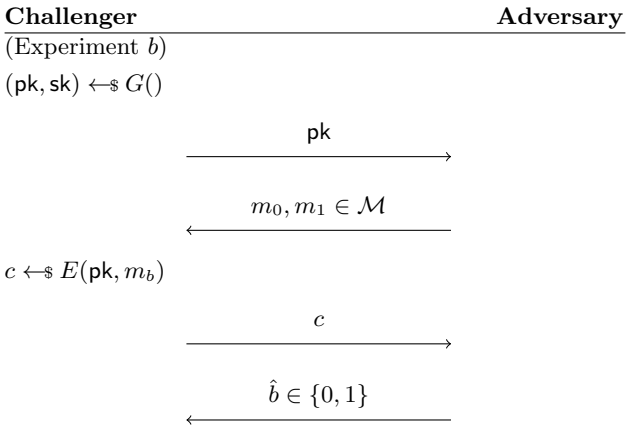

Figure 2.1: Experiment game of Attack Game 2.1.1

If W_b is the event that $\mathcal{A}$ outputs 1 we define $\mathcal{A}$'s advatange is $\mathsf{Adv}^{\mathrm{ss}}_{\mathcal{A},\mathcal{E}}(n) \coloneqq |\Pr[W_0] - \Pr[W_1]|$

Definition 2.1.1. *A public key encryption scheme $\mathcal{E}$ is semantically secure if for all efficent adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathrm{ss}}_{\mathcal{A},\mathcal{E}}(n) = \mathsf{negl}(n)$*

Let us now note one property of any semantically secure public-key scheme that it uses a randomized encryption algorithm. If it did not use a form of randomized encryption for the attack game 2.1.1 we could have the adversary $\mathcal{A}$ simply run the encryption themselves $E(\mathsf{pk}, m_0)$ on one of their messages, they could then compare this message to the cipher text provided by the adversary, they could then determine if they match and break 2.1.1. Therefore we see that such encryption algorithms CANNOT be deterministic.

2.1.1 CPA Security

CPA security of Semantic Security against chosen plaintext attack are two distinct notions of security that can first looked at in the lens of symmetric ciphers, more so they can be looked at in the lens of public key encryption. In the case of symmetric ciphers we see that semantic security does not imply CPA however in the case of public key encryption Semantic security does in fact imply CPA security. A simple reason for this is that in the public key setting adversaries can encrypt any messages they like without having to learning the secret key. This is because they have access to the public key. In the symmetric case this is not possible.

Attack Game 2.1.2. *(CPA Security) A public key encryption scheme $\mathcal{E}$ is called a semantically secure against chosen plaintext against or (CPA Secure) if for all efficient adversaries $\mathcal{A}$ the value $CPAadv[\mathcal{AE}]$ is negligible.*

Theorem 2.1.1. *If a public key encryption scheme $\mathcal{E}$ is semantically secure then it is also CPA Secure.*

Proof Idea:
Suppose that $\mathcal{E} = (G, E, D)$ is defined over $(\mathcal{M}, \mathcal{C})$ and let $\mathcal{A}$ be a CPA Adversary that plays the attack game 2.1.1 with respect to $\mathcal{E}$. Suppose it makes atmost Q queries to its challenger. We can describe relevant hybrid games $j = 0, \ldots Q$ for hybrid game j it played between $\mathcal{A}$ and a challenger who works as follows:

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ G()$
2. Send pk to $\mathcal{A}$.
3. Upon receiving the ith query $(m_{i0}, m_{i1}) \in \mathcal{M}^2$ from $\mathcal{A}$ do the following:
   (a) If $i > j$: $c_i \leftarrow\!\!\$\ E(\mathsf{pk}, m_{i0})$
   (b) Else $c_i \leftarrow\!\!\$\ E(\mathsf{pk} m_{i1})$
   (c) Send c_i to $\mathcal{A}$.

2.1.2 Encryption Based on a trapdoor scheme

Our encryption scheme $\mathcal{E}_{TDF}$ is built of several components.

1. A trapdoor function scheme $T = (G, F, I)$ over $(\mathcal{X}, \mathcal{Y})$
2. A symmetric cipher $\mathcal{E}_s = (E_s, D_s)$ over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$
3. A hash function $H : \mathcal{X} \to \mathcal{K}$.

The message space for $\mathcal{E}_{TDF}$ is $\mathcal{M}$ and the iphertext is $\mathcal{Y} \times \mathcal{C}$ We now describe they key generation, encryption, and decryption algorithm for $\mathcal{E}_{TDF}$.

The Key generation algorithm for $\mathcal{E}_{tdf}$ is the key generating algorithm for T.

For a given public key pk and a given message $m \in \mathcal{M}$ the encryption algorithm runs as follows:

$$E(\mathsf{pk}, m) \coloneqq x \leftarrow\!\$\ \mathcal{X}, y \leftarrow\!\$\ F(pk, x), k \leftarrow H(x), c \leftarrow\!\$\ E_s(k, m) : \text{output } (y, c)$$

For a given secret key sk and a given cipher text (y, c) the decryption algorithm runs as follows:

$$D(\mathsf{sk}, (y, c)) \coloneqq x \leftarrow I(\mathsf{sk}, y), k \leftarrow H(x), m \leftarrow D_s(k, c) : \text{output } m$$

Remark 2. *As it can be seen above utilizing a hash function, a trapdoor function scheme and a symmetric cipher we can build a semantically secure public key encryption scheme. The trap door scheme is used in the encryption step to encrypt the public key along with some random value from $\mathcal{X}$, this value can be inverted by the owner with the secret key to obtain x but is feasibly hard to determine x for a non-holder of sk, this x is hashed to obtained k, which we encrypt using our symmetric cipher encryption function E_s. Therefore we see to decrypt, the holder of the secret key calls the inverter function of T to obtain x from this, they take the hash of x to obtain k, given k, c they can use the symmetric cipher decryption function to decrypt the message.*

It can be proved that $\mathcal{E}_{edf}$ is semantically secure utilizing the correctness property of T and if H is modeled as a random oracle, and E_s is semantically secure.

2.1.3 Instantiating $\mathcal{E}_{TDF}$ with RSA

Suppose we now use RSA to instantiate T in the above encryption scheme for $\mathcal{E}_{TDF}$, the scheme is parameterized by l the length of prime factors of the RSA modulus, and the encryption exponent e, which is a positive odd integer.

$E_{RSA} = (G, E, D)$ with message space $\mathcal{M}$ and ciphertext $\mathcal{X} \times \mathcal{C}$ follows:

1. $\mathrm{G}() \coloneqq (n, d) \leftarrow\!\$\ RSAGEN(l, e), \mathsf{pk} \leftarrow (n, e), \mathsf{sk} \leftarrow (n, d)$ output $(\mathsf{pk}, \mathsf{sk})$
2. For a given public key $\mathsf{pk} = (n, e)$ and message $m \in \mathcal{M}$ the encryption algorithm runs as follows:

$$E(\mathsf{pk}, m) : +x \leftarrow\!\$\ \mathbb{Z}_n y \leftarrow x^e k \leftarrow H(x) c \leftarrow\!\$\ E_s(k, m) : \text{output} : (y, c)$$

3. For a secret key $\mathsf{sk} = (n, d)$ and a cipher text (y, c) decryption runs as follows:

$$D(\mathsf{sk}, (y, c)) \coloneqq x \leftarrow y^d, k \leftarrow h(x), m \leftarrow D_s(k, c) : \text{output} m$$

Theorem 2.1.2. *Assume that $H : \mathcal{X} \to \mathcal{K}$ is modeled as a random oracle, if the RSA assumption holds for paramters (l, e) and $\mathcal{E}_s$ is semantically secure, then $\mathcal{E}_{RSA}$ is semantically secure.*

2.1.4 Oblivious Transfer based on Diffie-Hellman

The question being asked is if some provider such as a newspaper has articles $m_1, \dots m_n \in \mathcal{M}$ and you are interested in reading an article i, is it possible to download article m_i without the newspaper learning what article I downloaded. Besides the obvious, having the newspaper send all the articles and promising to pay for the ones you read, we could utilize oblivious transfers as a solution to this problem. This could be considered a $\frac{1}{n}$ oblivious transfer or OT where the user learns a single item and the sender (newspaper) learns nothing about the specific i chosen just that only 1 was chosen.

It is important to note that OT protocols security can be quite subtle and extra caution should be considered when running OT concurrently.

A secure OT from ElGamal Encryption

This protocol satifies basic security of OT security in the random security model, assuming only CDH. Let $\mathcal{M}$ be the message space for the sender's message, and let $\mathbb{G}$ be a cyclic group of prime order q with a generator $g \in \mathbb{G}$. Suppose H is a hash function $H : \mathbb{G}^2 \to \mathcal{K}$ (modeled as a random oracle), and a semantically secure symmetric cipher (E_s, D_s). Additionally suppose that communication is happening over a secure channel.

Let's now see how this system works:

- Sender choses $\beta \leftarrow\!\!\$\, \mathbb{Z}_q$ and computes $v \leftarrow g^\beta$ and sends v to the receiver
- Receiver choses $\alpha \leftarrow\!\!\$\, \mathbb{Z}_q$, computes $u \leftarrow g^\alpha v^{-i}$ and sends u to the sender.
- For $j = 1, \dots n$ the sender computes:
  1. $u_j \leftarrow u \cdot v^j$ (ElGamal Public Key)
  2. $w_j \leftarrow u_j^\beta$ (Construct ElGamal Cipher Text)
  3. $k_j \leftarrow H(v, w_j)$ (Use randomness β and public key u_j)
  4. $c_j \leftarrow\!\!\$\, E_s(k_j, m_k)$ encryption of m_j

  The sender than sends $c_1, \dots c_n$ to the receiver. Note that all n ElGamal ciphertexts are generated using the same encryption randomness.
- The receiver who has secret key α for the ElGamal public key $u_i = g^\alpha$ decrypts c_i:
  1. Compute $w \leftarrow v^\alpha, k \leftarrow H(v, w)$
  2. Output $m \leftarrow D_s(k, c_i)$

This works as follows: $w = v^\alpha = g^{\alpha\beta}$ and in the decryption stage the key is calculated as $H(v, w)$. In the encryption stage by the sender we see that $k_i \leftarrow H(v, w_i)$ where

$$w_i = u_i^\beta = (uv^i)^\beta = ((g^\alpha v^{-i})v^i)^\beta = g^{\alpha\beta}$$

therefore we can see that the keys are the same. Given the the receiver can only query H once for $1, \dots n$ and that H is modeled as a random oracle it can be seen that the receiver can learn at most one message.

Adapative Oblivious Transfer

The OT protocol presented above while meeting the security needs fails in terms of performance, the newspaper has to rencrypt all the articles in the newspaper under new keys and fresh ciphertexts for each time Bob wants to use a seperate session. In most contexts this is simply to costly. A better system would allow the newspaper to encrypt all the articles once and post the encrypted articles on the web for anyone to download, and will open the article when Bob pays for one. This idea is known as an adaptive OT protocol.

Let F be a secure PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$, suppose that one party, the sender, has the PRF key $k \in K$, another party the receiver has an input $x \in X$. An oblivious PRF protocol is a protocol that lets the receiver learn the output $y = F(k, x)$ without learning anything else than the value of the PRF at any other input. $F(k, \cdot)$ should be able to run over many inputs. nothing about any of the inputs should be learned. More specifcally, we say that $F(k, \cdot)$ for l inputs the receiver cannot learn anything with the sender in l or fewer times.

We can design a Oblivious PRF defined as

$$F'(k, x) := H'(x, H(x)^k)k \in \mathbb{Z}_q, x \in \mathcal{X}$$

where $\mathbb{G}$ is a group of prime order q with generator g.

We say that $H : \mathcal{X} \to \mathbb{G}$ and that $H' : \mathbb{G} \to \mathcal{Y}$ are hash functions modeled as random oracles. The key space is $\mathcal{K} = \mathbb{Z}_q$.

Given our definition F' let us now define a protocol OPRF1 for this. Suppose that the sender has a key $k \in \mathbb{Z}_q$. In each run of the protocol the receiver has an input $x \in \mathcal{X}$ and the sender and receiver interact (over a secure channel) as follows:

1. The receiver chooses $p \leftarrow\$ \mathbb{Z}_q - \{0\}$ and computes $v \leftarrow H(x)^p$ and sends v to the sender.
2. Compute $w \leftarrow v^k \in \mathbb{G}$ and send w to the receiver.
3. The receiver computes and outputs $H'(x, w^{\frac{1}{p}})$

If the receiver and the sender are honest this last step comes downto:

$$w^{\frac{1}{p}} = (v^k)^{\frac{1}{p}} = ((H(x)^p)^k)^{\frac{1}{p}} = H(x)^k$$

We note that this is only partially secure in the prescence of a malicious sender.

Add Secure Version

Adapative OT from an OPRF

We need the following:

1. PRF F defined over $\mathcal{K}_{prf}, \{1, \ldots\}, \mathcal{K}$.
2. A OPRF protocol for a PRF F, which is secure against a malicious sender
3. A symmetric cipher (E_s, D_s) defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ that provides one-time authenticated encryption.

The protocol works as follows:

- Sender begins by getting $k \leftarrow\!\!\$\ K_{prf}$, for $i, \ldots n$ $k_i \leftarrow F(k, i)$, $c_i \leftarrow\!\!\$\ E_s(k_i, m_i)$ send $C = (c_1, \ldots c_n)$ to the receiver.
- When the receiver wants m_i the receiver and send establish a secure channel between them and use the OPRF to compute $F(k, i)$ the sender has k and the receiver has $i \in \{1, \ldots n\}$ when done, the reeceiver has k_i The receiver can then compute $m_i \leftarrow D_s(k_i, c_i)$

Chapter 3

Notes on Ch. 12