Personal Notes on *A graduate course in applied cryptography*

June 5, 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 trasncript 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*

$$(pk, sk) \leftarrow G(), x \xleftarrow{R} X, y \leftarrow F(pk, x)$$

- *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 $x \xleftarrow{R} x$, $y \leftarrow F(pk, x)$ 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 $G() \coloneqq (n, d) \xleftarrow{R} RSAGen(l, e), pk \leftarrow (n, e) sk \leftarrow (n, d)$
- 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 $(n, d) \xleftarrow{R} RSAGEN(l, e), x \xleftarrow{R} \mathbb{Z}_n, y \leftarrow x^e \in \mathbb{Z}_n$ 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 $(n, d) \xleftarrow{R} RSAGEN(l, e)$ and sends (n, e) to bob.
2. Bob computes $x \xleftarrow{R} Z_n, y \leftarrow x^e$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 $\alpha \xleftarrow{R} Z_q$, $u \leftarrow g^\alpha$ and sends u to Bob.
2. Bob computes $\beta \xleftarrow{R} \mathbb{Z}_q$ 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 $\alpha \xleftarrow{R} \mathbb{Z}_q u \leftarrow g^\alpha$ 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 $\alpha \xleftarrow{R} \mathbb{Z}_q$, $u \in g^\alpha$, $v \in g^\beta, w \leftarrow g^{\alpha\beta}$. 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 $\alpha\beta\gamma \xleftarrow{R} \mathbb{Z}_q$, $u \leftarrow g^\alpha, v \leftarrow g^\beta, w_0 \leftarrow g^{\alpha\beta}, w_1 \leftarrow g^\gamma$*
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 $\alpha' \xleftarrow{R} \mathbb{Z}_n$ 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 $\beta' \xleftarrow{R} \mathbb{Z}_n$ 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 $(\mathbb{G}, g) \xleftarrow{R} GGen()$ 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