Principles of Information Security

Assignment 1

Tathagato Roy (2019111020)

March 9, 2022

1 One Way Function

Intuitively a one way function is function which is easy to compute , but hard to invert.Let F be one such function.Then $F(x)$ for any x can be computed in polynomial time but given $F(x)$ to an adversary , it is hard to retrieve x for a random input x.
The formal Definition is : A function $f : \{0,1\}^* \to \{0,1\}^*$ is one way if one of the following conditions hold :
(1) (Easy to compute) There exist an polynomial time algorithm M_f computing f ,that is, $M_f(x) = f(x)$ for all x
(2) (hard to invert) For every probabilistic polynomial time algorithm there exist an negligible function negl such that $Pr[Invert_{f,A}(n) = 1] \leq negl$

1.1 Example of an One Way function

Consider the multiplicative group ${Z_p}^*$ where p is a prime of the form $p = 2q+1$ where q is also a prime , let

$$f(x, g, p) = g^x \bmod p$$

where $x \in \mathrm{Z}_p^*$ and g is a generator of ${Z_p}^*$ Then $f(x, g, p)$ is a one way function. This task of invert the above f is called the discrete log problem.
There is no guarantee that One Way functions actually exist,but we can build strong cryptographic systems and tools assuming that they do.

2 Hard Core Predicates

Intuitively Hard Core Predicate of a one way function f is a function which outputs just a single bit, such that given x, the Hard Core Predicate can be easily computed,but given $f(x)$ it is hard to compute.
The formal Definition is : A function $hc : \{0,1\}^* \to \{0,1\}$ is a hardcore predicate of a function f if hc can be computed in polynomial time and for every probabilistic polynomial time algorithm A , there exist an negligible function

negl such that $Pr_{x \leftarrow 0,1^n}[A(f(x)) = hc(x)] \leq 1/2 + negl$
where probability is taken over the uniform choice of x over $0, 1^n$ and random coin tosses of A .
If the one way function is hard to invert, there must be some information (represented by hc) about the input that is hidden in function f(x) which is hard to extract. Goldreich and Levin showed that for the discrete log problem, $msb(x)$ where x is the input is a hard core predicate of $f(x, g, p) = g^x \bmod p$

3 Pseudo Random Generator

It is not possible to find truly random functions without using a physical source of entropy.As computers are finite state machines, pseudo random generators are actually deterministic and is characterized by is seed value. We reduce our aim from truly random to pseudo random where a pseudo random generator would output a $l(n)$ bit output from a n bit output and a probabilistic polynomial time adversary cannot distinguish whether the output generated is from a uniform distribution with non negligible probability.
Formal Definition : let $l()$ be a polynomial and G be a deterministic polynomial time algorithm such that for any input $s \in \{0,1\}^n$: (1) $|G(s)| = l(n)$, (2) for every n , $l(n) > n$ and for all probabilistic polynomial time distinguisher D, There exist a negligible function negl such that :

$$|Pr[D(r) = 1] - Pr[G(s) = 1]| \leq negl$$

where r is chosen uniformly at random for $\{0,1\}^l(n)$, the seed s is chosen uniformly at random from $\{0,1\}^n$ and the probabilities is taken over random coins used by D and the choice of r and s
We find this definition of pseudo randomness suffices for almost all practical settings. We now construct a pseudo random generator from cryptographic primitives of one way function and hard core predicate

3.1 Construction of PRNG

We want to construct a PRNG f such that $f(s)$ where s is a n bit binary string outputs a $l(n)$ bit binary string where $l(n) = n + 1$ Such a example of f is

$$f(s) = (g^s \bmod p)||(msb(s))$$

where $hc_f = msb$ Using this function we generate a $n+1$ bit size random binary string from n bit input.We can use store the last bit for our final output and use the first n bit as the new seed.Repeating this procedure $l(n)$ times we can concatenate all the random last bit we have stored to create a $l(n)$ bit random string which is the output of the PRNG
The proof that this construction adheres to the definition of Pseudo Randomness is beyond the purview of this report,but the basic sketch idea of the proof is that for an Polynomial Time Adversary to distinguish between a truly uniform distribution and our pseudo random generated distribution it has to solve the discrete log problem in polynomial time.

4 Pseudo Random Function

A Pseudo Random Function is function which maps n bit string input to n bit output.There is little sense in calling a fixed function random or pseudorandom, so we deal with a distribution of functions.
We use keyed functions to simulate the distribution of functions.Consider keyed function F which maps $\{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ where the first input is called the key k.

$$F(k,x) = F_k(x)$$

We say F is efficient given there exist a polynomial time algorithm which can compute $F(k,x)$ given input k and x. Intuitively, we call F pseudorandom if the function F_k (for uniformly randomly chosen key k) is indistinguishable from a function chosen uniformly at random from the set of all functions having the same domain and range. The above is roughly equates to the fact that no probabilistic polynomial time adversary can determine whether they are interacting with F_k or f where f is a uniformly chosen function from the set of the all such length preserving functions of parameter n by querying the outputs for chosen inputs.
Now the formal definition of pseudo random functions : Let $F : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ be an efficient, length-preserving, keyed function,then F is a pseudorandom function if for all probabilistic polynomial-time distinguishers D, there exists a negligible function negl such that:

$$|Pr[D^{F_k(.)}(1^n) = 1] - Pr[D^{f_n(.)}(1^n) = 1] \leq negl$$

where $k \leftarrow \{0,1\}^n$ and n is chosen uniformly at random and f_n is chosen uniformly at random from the set of functions mapping n-bit strings to n-bit strings.

4.1 Construction of a Pseudo Random Function from Pseudo Random Generator

Let G be a pseudorandom generator that maps a n bit string to a $2n$ bit output that is with expansion factor $l(n) = 2 * n$.Let $G_0(x)$ be the first n bit of the output of $G(x)$ and $G_1(x)$ be the last n bit of $G(x)$.For every $k \in \{0,1\}^n$ let :

$$F_k(x_1x_2x_3....x_n) = G_{x_n}(...(G_{x_2}(G_{x_1}(k)))....)$$

The proof for the fact that this construction satisfies the given definition is too extensive for the purposes of the report, but for intuition, one can think what this construction essential does,is create a binary tree of depth n where the root value is k and depending on the input , one can reach a leaf of the root with uniform probability.

5 Chosen Plaintexts Attack (CPA) secure encryption scheme

Chosen Plaintext Attack is a form of attack in cryptosystem where the adversary can ask for the encryptions of plaintexts of their choice in an adaptive manner.We need to design a encryption system that allows us to be secure against such attacks.
The basic idea behind a CPA is that the adversary A asks for encryptions of multiple messages that it chooses in an adaptive manner. Formally A is allowed to interact freely with an encryption oracle, that is "black-box" that encrypts messages. We denote by $A^O()$ the computation of A given access to an oracle O, and therefore denote the computation of A with access to an encryption oracle by $A^{Enc_k()}$.When A queries its oracle by providing it with a message m , the O outputs $c = Enc_k(m)$

5.1 The CPA indistinguishability experiment $PrivK^{cpa}_{A,\pi}(n)$

Consider the Private Encryption scheme $\pi = (Gen, Enc, Dec)$ We know define the experiment $PrivK^{cpa}_{A,\pi}(n)$ 1.A random key k is generated by running $G(1^n)$

2.The adversary A is given input 1^n and oracle access to $Enc_k()$,and outputs a pair of messages m_0, m_1 of the same length.

3.A random bit $b \leftarrow \{0,1\}$ is chosen, and then a ciphertext $c \leftarrow Enc_k(m_b)$ is computed and given to A.c is called the challenge ciphertext.

4. The adversary A continues to have oracle access to $Enc_k()$, and outputs a bit $b^{'}$

5. if $b = b^{'}$output of the experiment 1 and 0 otherwise
The output 1 of the experiment means that the adversary has been able to succesfully break the cryptosystem it has oracle access to.We want to create a system where the probability of success for the Adversary is negligible.

5.2 Construction of CPA secure encryption system

Let F be a pseudorandom function. Consider a private-key encryption scheme for n bit messages as follows

(1) Gen: on input 1^n, choose $k \leftarrow \{0,1\}^n$ uniformly at random and output it as the key.

(2) Enc: on input a key $k \in \{0,1\}^n$ and a message $m \in 0,1^n$, choose $r \leftarrow \{0,1\}^n$ uniformly at random and output the ciphertext $c = \langle r, F_k(r) \oplus m \rangle$

(3) Dec: on input a key $k \in \{0,1\}^n$ and a ciphertext $c = \langle r, s \rangle$, output the plaintext message $m = F_k(r) \oplus s$.
This construction satisfies the neccesary properties for our cryptosystem to be CPA secure.

6 Message Authentication Code (MAC)

Communication over networks require us to send messages over potentially insecure channels.Not only can our adversary read our messages it may be possible that he or she can potential modify our messages.Hence it is imperative for B who has received a message from A to know the message has been indeed sent by A and the message hasn't been corrupted by an adversary or by channel noise.
To solve this issue we will assume that the two communicating parties have a secret key k.Hence this is private key cryptography.
A message authentication code (MAC) is an algorithm that is applied to a message. The output of the algorithm is a tag that is sent along with the message. Security is ensured by requiring that no probabilistic polynomial time Adversary can generate a valid MAC tag for any message with a non-negligible probability.
We now formally define MAC.
A message authentication code or MAC is a tuple of probabilistic polynomial-time algorithms (Gen, Mac, Vrfy) fulfilling the following:
(1).Upon input 1^n, the algorithm Gen outputs a uniformly distributed key k of length n; $k \leftarrow \{0,1\}^n$.

(2)The algorithm Mac receives for input some $k \in \{0,1\}^n$ and $m \in \{0,1\}^*$ and outputs some $t \in \{0,1\}^*$. The value t is called the MAC tag.

(3)The algorithm Vrfy receives for input some $k \in \{0,1\}^n$, $m \in \{0,1\}^*$ and $t \in \{0,1\}^*$ and outputs a bit $b \in \{0,1\}$.

(4).For every n, every $k \in \{0,1\}^n$ and every $m \in \{0,1\}^*$ it holds that $Vrfy_k(m, mac_k(m)) = 1$.

6.1 Construction of MAC from Pseudo Random Functions

We use the following construction for Variable length MAC.
Let $F : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a function such that for every $k \in \{0,1\}^n$, $F_k()$ maps n-bit strings to n-bit strings. Define a variable length MAC as follows:
(1) $Gen(1^n)$: upon input 1^n choose $k \leftarrow \{0,1\}^n$

(2) $Mac_k(m)$: upon input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}$ of

length at most $2^{(n/4-1)}$ first parse m into d blocks $m_1, , m_d$, each of length $n/4$. (In order to ensure unique encoding, the last block is padded with 10^*.) Next, choose a random identifier $r \leftarrow \{0,1\}^{n/4}$.
Then, for $i = 1, ..., d$, compute $t_i = F_k(r||d||i||m_i)$, where i and d are uniquely encoded into strings of length $n/4$ and $||$ denotes concatenation. Finally, output the tag $t = (r, t_1, ..., t_d)$.

(3) $Vrfy_k(m, t)$: Upon input key k, message m and tag t, rerun the MAC tag generation algorithm ,except that instead of choosing a random identifier r, take the identifier r that appears in t. Output 1 if and only if the tag that is received by running Mac with this r is identical to t.

7 Chosen-Ciphertext Secure Encryption

We know consider a more powerful adversary than what we considered in Chosen-Plaintext Attack.We now consider the notion of Chosen-Ciphertext Attack.The adversary can now encrypt any plaintext of its choice and decrypt any ciphertext of its choice.
More formally the adversary now has access to both the encryption and decryption oracle.

7.1 The CCA indistinguishability experiment $PrivK^{cpa}_{A,\pi}(n)$

Consider the private encryption scheme $\pi = (Gen, Enc, Dec)$ The CCA indistinguishability experiment $PrivK^{cpa}_{A,\pi}(n)$ is given by :

1.A random key k is generated by running $G(1^n)$

2.The adversary A is given input 1^n and oracle access to $Enc_k()$ and $Dec_k(.)$,and outputs a pair of messages m_0, m_1 of the same length.

3.A random bit $b \leftarrow \{0,1\}$ is chosen, and then a ciphertext $c \leftarrow Enc_k(m_b)$ is computed and given to A.c is called the challenge ciphertext.

4. The adversary A continues to have oracle access to $Enc_k()$ and $Dec_k(*)$,but it cannot query $Dec_k(.)$ on the challenge ciphertext, and outputs a bit $b^{'}$

5. if $b = b^{'}$ output of the experiment 1 and 0 otherwise
The adversary is succesfully in breaking the encryption scheme π if it manages output the correct bit.Our aim in building a CCA secure cryptosystem is that no polynomial time adversary with oracle access to $Enc_k(.)$ and $Dec_k(.)$ can output the correct with probability less than $1/2 + negl$

7.2 Construction a CCA secure Encryption scheme with CPA secure Encryption scheme and MAC

Let $\pi_E = (Gen_E, Enc, Dec)$ be a CPA-secure encryption scheme and $\pi_M = (Gen_M, Mac, Vrfy)$ a secure message authentication code (MAC), then the following construction is CCA secure encryption scheme .

(1) $Gen'(1^n)$: upon input 1^n , choose $k_1, k_2 \leftarrow \{0,1\}^n$

(2)$Enc'_k(m)$: upon input key $(k1, k2)$ and plaintext message m, compute $c = Enc_{k_1}(m) and t = Mac_{k_2}$ and output the pair (c, t)

(3)$Dec'_k(c, t)$: upon input key $(k1, k2)$ and ciphertext (c, t), first verify that $Vrfy_{k_2}(c, t) = 1$. If yes, then output $Dec_{k_1}(c)$, else output null
The above construction gurantess that no polynomial time adversary has more than a non-negligible change of breaking the encryption scheme even with chosen ciphertext attacks.

7.3 Fixed Length Collision Resistance Hash Function

:
A hash function is basically a function which takes a string of arbitrary size as input and compressed them into a string of smaller size.Let H be a hash function , a collision happens when there exist x , y such that $H(x) = H(y)$.A good hash function is one which has very less collisions.Note from the pigeonhole principle, there is guaranteed to exist atleast one collision, that it is impossible to build a hash function which has no collision and also performs compression.
We introduce a weaker notion of collision resistance , that is a collision-resistant hash function must have the property that no polynomial-time adversary can find a collision in it,that is no polynomial-time adversary should be able to find a distinct pair of values x and y such that $H(x) = H(y)$
We know formally define a Hash Function.
A hash function is a pair of probabilistic polynomial-time algorithms (Gen, H) fulfilling the following:

(1) Gen is a probabilistic algorithm which takes as input a security parameter 1^n and outputs a key s. We assume that 1^n is included in s.

(2)There exists a polynomial l such that H is (deterministic) polynomial time algorithm that takes as input a key s and any string $x \in \{0,1\}^*$, and outputs a string $H^s(x) \in \{0,1\}^{l(n)}$

(3)If for every n and s, H^s is defined only over inputs of length $l'(n)$ and $l'(n) > l(n)$ then we say that (Gen, H) is a fixed-length hash function with length parameter l'

7.4 The collision-finding experiment $Hash - coll_{A,\pi}(n)$

We know describe experiment to find collision in a hash function. We begin by defining an experiment for a hash function $\pi = (Gen, H)$, an adversary A and a security parameter n:

(1) A key s is chosen: $s \leftarrow Gen(1^n)$

(2) The adversary A is given s and outputs a pair x and y.Formally, $(x, y) \leftarrow A(s)$.

(3). The output of the experiment is 1 if and only if $x \neq y$ and $H^s(x) = H^s(y)$. In such a case we say that A has found a collision.

7.5 Definition of Collision Resistance

Formally A hash function $\pi = (Gen, H)$ is collision resistant if for all probabilistic polynomial-time adversaries A there exists a negligible function negl such that

$$Pr[Hash - coll_{A,\pi}(n) = 1] \leq negl(n)$$

7.6 Construction of a Fixed Length Collision Resistance Hash Function

let G be a polynomial-time algorithm that, on input 1^n, outputs a cyclic group G, its order q (with $||q|| = n$), and a generator g. As always, we also assume that the group operation in G can be computed efficiently. Finally, we also require that q is prime except possibly with negligible probability. A fixed-length hash function based on G is given as :
Define (Gen, H) as follows:

(1) Key generation algorithm Gen: On input 1^n, run $G(1^n)$ to obtain (G, q, g) and then select $h \leftarrow G$. Output $s = \langle G, q, g, h\rangle$.

(2) Hash algorithm H: On input $s = \langle G, q, g, h\rangle$ and message $(x1, x2) \in Z_q \times Z_q$, output $g^{x_1} h^{x_2} \in G$.
We can prove that the above construction is collision resistance,that is a polynomial time adversary would find it hard to find collision with non-negligible probability if discrete log problem is hard.
The prime number must be of the form $p = 2 * q + 1$ where q is also a prime.The prime can be generated using brute force or also by Miller Rabin test.
The generator can found using the following idea: factorize $p1$ and use the property that a is a generator if and only if $a^{(p1)/q}! \equiv 1 \bmod p$ for all primes q that divide $(p - 1)$. So we only need to test our candidate a against all the prime factors of $p - 1$
h is randomly sampled from G

8 Merkle-Damgard transform to obtain a provably secure variable collision resistant hash function

Ideally we want our collision resistance hash function to take variable input length.In this section we see how using a fixed length collision resistance hash function we can build variable length collision resistance hash function using something called the Merkle -Damgard Transform.
given a fixed-length collision-resistant hash function that compresses its input by half; that is, the input length is $l'(n) = 2l(n)$ and the output length is $l`(n)$. We denote the given fixed-length collision-resistant hash function by (Gen_h, h) and use it to construct a general collision-resistant hash function (Gen, H) that maps inputs of any length to outputs of length $l'(n)$
Let (Gen_h, h) be a fixed-length hash function with input length $2l(n)$ and output length $l(n)$. Construct a variable-length hash function (Gen, H) as follows:

(1) $Gen(1^n)$: upon input 1^n run the key-generation algorithm Gen_h of the fixed-length hash function and output the key. That is, output $s \leftarrow Gen_h$.

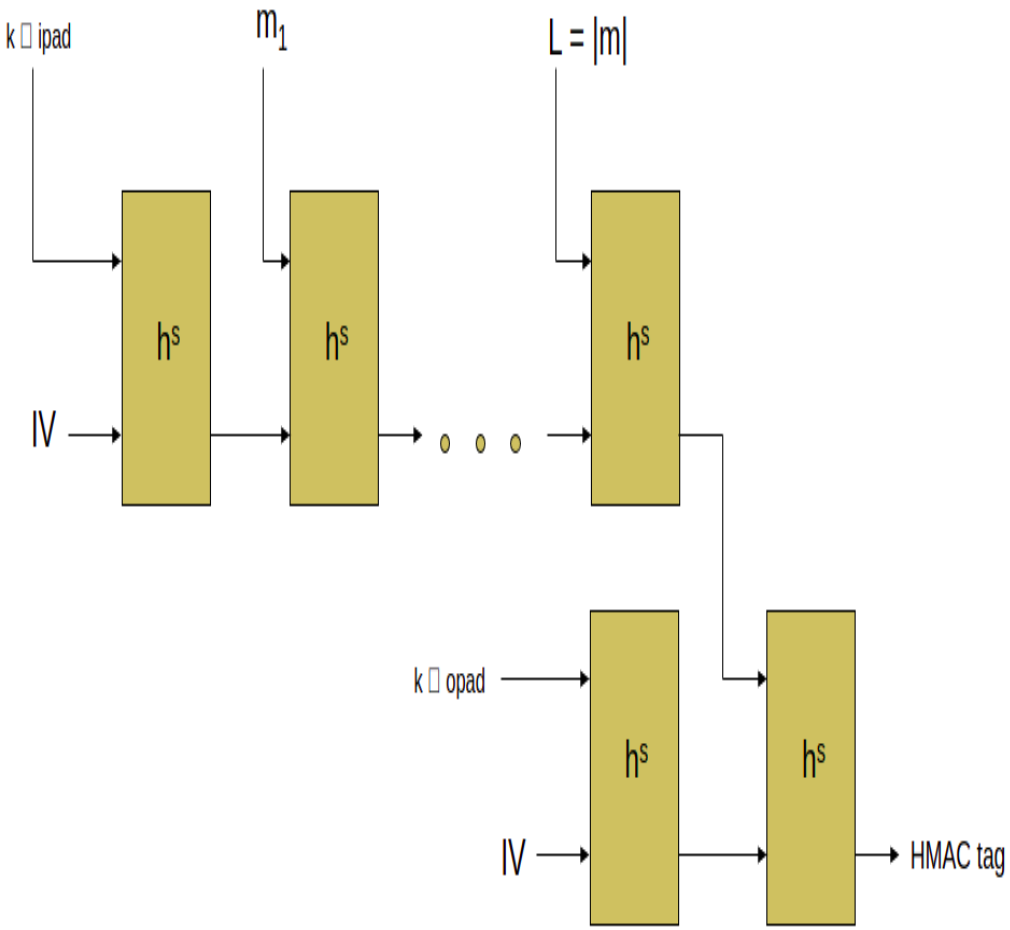
(2) $H^s(x)$: Upon input key s and message $x \in \{0,1\}^*$ most $2^{l(n)}1$, compute as follows:
1. Let $L = |x|$ (the length of x) and let $B = \lceil L/l \rceil$(i.e., the number of blocks in x). Pad x with zeroes so that its length is an exact multiple of l
2. Define $z_0 := 0^l$ and then for every i = 1, . . . , B, compute $z_i := h(z_i||x_i)$, where h^s is the given fixed-length hash function.
3. Output $z = H^s(z_B||L)$
The above construction gives the resulting variable length collision resistance hash function.

9 Construction of HMAC from collision resistant hash function

We now give the construction of HMAC from a variable length collision resistant hash function.
The HMAC construction is as follows:
(1) $Gen(1^n)$: upon input 1^n run the key-generation for the hash function obtaining s, and choose $k \leftarrow \{0,1\}^n$

(2) $Mac_k(m)$: upon input (s, k) and $x \in \{0,1\}^*$ compute
$HMAC^s_k(x) = H^s_{IV}(k \oplus opad||H_{IV}(k \oplus ipad||x))$ and output the result.

(3)$Vrfy_k(m, t)$: output 1 if and only if $t = Mac_k(m)$.



where IV is initialised to 0^n , The string *opad* is formed by repeating the byte 36 in hexadecimal as many times as needed; the string *ipad* is formed in the same way using the byte $5C$