Cryptography and Cryptanalysis

Summer of Science 2020 - Topic Report

Dhruv Arora
First Year Undergraduate,
Department of Computer Science and Engineering

1 Introduction and History

DEFINITION (Cryptography)**.** *The scientific study of techniques used for securing digital information, transactions and communications is called cryptography.*

Historically, cryptography was studied as an art. Due to heavy application and need for schemes to be secure, it has turned into a science.

1.1 Historic Background

Most of these schemes are not secure at all w.r.t the modern definitions of security. Their study reveals the importance of formal definitions and proofs of security. Since these are artistic schemes, they are based in the historic private-key setting.

1.2 Private Key Setting

DEFINITION (Private Key Setting)**.** *In a private key setting, two communicating parties have some secret information conveyed in advance called a* **`key`***. The secrecy of this key is of importance to the cryptographic application.*

Historical cryptographic application was limited to encryption schemes (ciphers) as the sense of message authentication wasn't developed then.
All these ciphers have a common syntax :

1. $Gen()$
   It is run once to generate a key k according to a specified probability distribution which is then shared by both parties.

2. $Enc_k()$
   It is run whenever a message is to be sent. It takes as input the key k and a message m and outputs $c \leftarrow Enc_k(m)$, the ciphertext.

3. $Dec_k()$
   It is run when a message is received. It takes as input the key k and ciphertext c. It outputs the plaintext $m := Dec_k(c)$.

The sets (spaces) are generically defined:

- $\mathcal{K}$: The key-space. I.e. the set of all possible keys output by $Gen()$.
- $\mathcal{M}$: The message-space. I.e. the set of all possible plaintexts.
- $\mathcal{C}$: The ciphertext-space. It is completely defined by the $\mathcal{K}, \mathcal{M}$ and $Enc_k()$.

A scheme under the private key setting is completely defined by specifying $Gen(), Enc_k(), Dec_k()$ and $\mathcal{M}$ ($\because \mathcal{K}$ is completely defined by $Gen()$).
Also, for a scheme to be valid, $Dec_k(Enc_k(m)) = m$ $\forall k$ output by Gen and $\forall m \in \mathcal{M}$.
The security of a cipher is dependent on the secrecy of the key k.
Kerckhoff's first principle states that

> *The secrecy in an encryption scheme should be based solely on the secrecy of the key.*

Attacks and Adversaries Encryption schemes are essentially a protection against adversaries who try to *break* the scheme by gaining access to information. The definition of security of a scheme thus consists of what encompasses a break. Achieving such a break is also dependent on the power of an adversary. Depending in this power, attacking adversaries are classified as:

1. *Ciphertext-Only Attack* :
   The adversary intercepts a ciphertext or a group of ciphertexts.
2. *Known-Plaintext Attack* :
   The adversary knows a set of pairs $(c, Dec_k(c))$.
3. *Chosen-Plaintext Attack* :
   The adversary can get access to a few ciphertexts for (encryptions of) his chosen plaintexts.
4. *Chosen-Ciphertext Attack* :
   The adversary can get access to a few plaintexts for (decryptions of) his chosen ciphertexts.

All historic schemes are completely broken under the weakest, i.e. ciphertext only attacks.

1.3 Historic Ciphers

1.3.1 Caesar's Cipher

$\mathcal{K} = \{3\}$, $\mathcal{M}$ = *Set of all strings comprising of lowercase english alphabets*
Assume that $int(x)$ maps the characters {`a,b,...,z`} to {0,1,...,25} in order. And $char(y)$ is the inverse mapping of $int(x)$.

1. $Gen()$: Outputs 3 with probability 1.
2. $Enc_k(m)$: For all characters x in m, performs $x = char((int(x) + k)\%26)$.
3. $Dec_k(c)$: For all characters x in c, performs $x = char((int(x) - k)\%26)$.

For example :

`thisisaplaintext` corresponds to ***wklvlvdsodlqwhaw***

It is easy to verify that the scheme is valid.
The scheme is easily broken $\because |\mathcal{K}| = 1$, the key is publically known, therefore, anyone can decrypt any ciphertext.
An equivalent version is called the ROT13 cipher. The only difference is that $\mathcal{K} = \{13\}$.
These type of ciphers have application in hiding information on public domains that everyone would not like to see.

1.3.2 Shift Cipher

It is an extension to the Caesar's Cipher.

1. $Gen()$: $k \leftarrow \mathcal{K} = \{0, 1, \cdots, 25\}$.
2. $Enc_k(m)$: For all characters x in m, performs $x = char((int(x) + k)\%26)$.

3. $Dec_k(c)$: For all characters x in c, performs $x = char((int(x) - k)\%26)$.

For example, if the key were 8:

`thisisaplaintext` corresponds to ***bpqaqaixtiqvbmfb***

It is easy to verify that the scheme is valid.
The scheme can be easily broken by `Exhaustive Search`. There are only 26 possible values for the key. Each can be used to decrypt a ciphertext and the value for which the plaintext makes the *most sense* is the key. There is another approach which doesn't make use of the *sense* of a message. This is called a `Statistical Attack`.

If the message space comprises of sentences in English language, for fairly long texts, the relative probability of occurences of different characters is known.
Define p_i to be the statistical probability of occurence of $char(i)$ and q_i be the probability of occurence of $char(i)$ in the given ciphertext. It is known that :

$$\sum_{i=0}^{25} p_i^2 \approx 0.065$$

We define I_k to be:

$$I_k = \sum_{i=0}^{25} p_i \cdot q_{(i+k)\%26}$$

The, the $k \in \mathcal{K}$ for which I_k is closest to 0.065 is the expected key.

This encompasses an exhaustive search as well. Thus,

A secure encryption scheme must have a key space $\mathcal{K}$ large such that it is not vulnerable to Exhaustive Searching.

1.3.3 Mono-Alphabetic Substitution Cipher

$\mathcal{K}$ = The set of all bijections on the set of characters of English Language.

1. $Gen()$: Outputs a key k randomly from the set $\mathcal{K}$.
2. $Enc_k(m)$: Replaces all characters m^i in m by $k(m^i)$.
3. $Dec_k(c)$: Replaces all characters c^i in c by $k^{-1}(c^i)$.

As an example, consider the randomly generated mapping :
{'a': 's', 'b': 'z', 'c': 'e', 'd': 'b', 'e': 'y', 'f': 'u', 'g': 'j', 'h': 'a', 'i': 'c', 'j': 'w', 'k': 'd', 'l': 'q', 'm': 'o', 'n': 'x', 'o': 'h', 'p': 'm', 'q': 'f', 'r': 'r', 's': 'g', 't': 'l', 'u': 'n', 'v': 'v', 'w': 'p', 'x': 'i', 'y': 'k', 'z': 't'}. Under this,

The sentence `thequickbrownfoxjumpsoverthelazydog` translates to
layfncedzrhpxuhiwnomghvyrlayqstkbhj

This scheme has a $|\mathcal{K}| = 26!$, and thus, it is secure against exhaustive searching.
It can be easily broken by launcing a `Statistical Attack`. This involves creating a frequency table for the alphabets. This very nearly matches the statistical english table, and a proposed the mapping is obtained. It is almost certain that this mapping is erroneous, but still a big part of the ciphertext can be deciphered with it. The rest can be done by human analysis.

1.3.4 Vigénere Cipher

$\mathcal{K}$ = The set of all strings of arbitrary length.

1. $Gen()$: Generates a random key k from $\mathcal{K}$.
2. $Enc_k(m)$: The key is placed under the plaintext m and repeated to match the length of m. Then, the characters of m and k are added together to get c. Formally,
   Let m_i be the i^{th} character of m. Then, $c_i \leftarrow m_i + k_{i\%len(k)}$.
3. $Dec_k(m)$: The key is placed under the ciphertext c and repeated to match the length of c. Then, the characters of k are subtracted from c to get m.

For example, say k is $cafe$. The sentence $tellhim$ translates to:

`tellhim`
cafecaf
`wfrqkjs`

The scheme is easily broken by `Statistical Attacks` again. There are two cases:

- Known Key Length (τ) :
  Let keys be $\{k_1, k_2, \cdots, k_\tau\}$. Launch statisitical attacks on $\{c_i, c_{i+\tau}, \cdots\}$ to get k_i.
- Unknown Key Length :
  First estimate τ, and then use the approach described above.
  - Kasiski's Method : It searches for repeated bigrams ang trigrams in the ciphertext. The min distance b/w two of them is a multiple of τ.
  - Index Of Coincidence : Assume a $\tau = 1, 2, 3, \cdots$. For the dataset $\{c_1, c_{1+\tau}, c_{1+2\tau}, \cdots\}$, calculate the frequency distribution q_i. Return the τ for which

    $$\sum_{i=0}^{25} q_i^2 \approx 0.065$$

    For a fairly large dataset, if τ is incorrect, the matching will be close to uniform random, and thus, we expect $q_i \approx 1/26$. Thus, for an incorrect τ,

    $$\sum_{i=0}^{25} q_i^2 \approx 0.038$$

2 Perfect Security

2.1 Definitions of Security

A definition for security for an encryption scheme consists of :

1. What is considered a breach of the securiy?
2. What are the powers of an adversary?

The definition only specifies the powers of the adversary, his tactics can be arbitrary.

Most of the times, the correctness of a scheme relies on some unproven assumptions. Such assumptions are stated along with the scheme and are extensively tested.

2.2 Perfect Secrecy

The definition of perfect secrecy involves an eavesdropping adversary who intercepts one message.

DEFINITION (Pefect Secrecy)**.** *An encryption scheme* $\Pi = (Gen, Enc, Dec)$ *over a plaintext space* $\mathcal{M}$ *is perfectly secure if* $\forall$ *probability distributions over* $\mathcal{M}$*, any message* $m \in \mathcal{M}$ *and any ciphertext* $c \in \mathcal{C} : Pr[C = c] > 0$*,*

$$Pr[M = m|C = c] = Pr[M = m]$$

This is a formal way of saying that *knowing the ciphertext c, does not help in predicting m.*
There are two alternate definitions that can be proven equivalent to the above.

DEFINITION (Perfect Indistinguishability)**.** *An encryption scheme* $\Pi = (Gen, Enc, Dec)$ *over a plaintext space* $\mathcal{M}$ *is perfectly secure iff* $\forall$ *probability distributions over* $\mathcal{M}$*, any messages* $m_0, m_1 \in \mathcal{M}$ *and any ciphertext* $c \in \mathcal{C} : Pr[C = c] > 0$*, we have*

$$Pr[C = c|M = m_0] = Pr[C = c|M = m_1]$$

For the second definition, one must define an experiment $\mathbf{PrivK}^{\mathbf{eav}}_{\mathcal{A},\mathbf{\Pi}}$ as:

1. $m_0, m_1 \leftarrow \mathcal{A} : m_0, m_1 \in \mathcal{M}$
2. $k \leftarrow Gen(), b \leftarrow \{0, 1\}$ and $c \leftarrow Enc_k(m_b)$ is output to $\mathcal{A}$.
3. $\mathcal{A}$ outputs a bit b'
4. The experiment succeeds (1) iff $b = b'$.

DEFINITION (Adversarial Indistinguishability)**.** *An encryption scheme* Π *is adversarially indistinguishible if for any adversary* $\mathcal{A}$*,*

$$Pr[PrivK^{eav}_{\mathcal{A},\Pi} = 1] = 1/2$$

That is to say that no adversary can do better than randomly guessing their output bit.

2.2.1 Restrictions to Perfect Secrecy

A perfectly secure encryption scheme requires two conditions over the standard spaces to be satisfied which makes it impractical. These can be stated as:

THEOREM. *If a scheme* Π *is perfectly secure, then* $|\mathcal{K}| \geq |\mathcal{M}|$*.*

THEOREM. *If a scheme* Π *is perfectly secure, then* $|\mathcal{C}| \geq |\mathcal{M}|$*.*

2.3 Shannon's Theorem

THEOREM (Shannon's Theorem)**.** *An encryption scheme* Π *defined such that* $|\mathcal{K}| = |\mathcal{M}| = |\mathcal{C}|$ *is perfectly secret iff:*

1. *Every* $k \in \mathcal{K}$ *is chosen by* $Gen()$ *uniform randomly.*
2. $\forall m \in \mathcal{M}$ *and* $\forall c \in \mathcal{C}$*, there is a single* $k \in \mathcal{K} : c \leftarrow Enc_k(m)$

An application of Shannon's theorem is Verman's scheme.
Verman's Scheme (One Time Pad) is a perfectly secure.

CONSTRUCTION 1 (One-Time Pad). *Decide an integer (l) and fix* $\mathcal{K} = \mathcal{M} = \mathcal{C} = \{0,1\}^l$.

1. *Gen() : Picks a key k uniform randomly in* $\mathcal{K}$.
2. $Enc_k(m)$ *: Returns* $c \leftarrow k \oplus m$
3. $Dec_k(c)$ *:* $m \leftarrow k \oplus c$

THEOREM. *The One Time Pad is perfectly secure w.r.t the provided definition of security.*

The proof is trivial by shannon's theorem.
It is extremely hard to push perfect secrecy to harder adversarial attacks.

3 Asymptotic Approach

Computationally secure schemes allow 2 relaxations on the definition of perfect secrecy:

1. Security is offered against polynomial time (efficient) adversaries.
2. An adversary can potentially succed with a negligible probability.

Thus, a computationally secure scheme can be broken given enough time and resources.
To view a schme computationally, one can assign a security parameter ($n \in \mathbb{Z}$) to it. The runtime of the parties, the adversary and the probability of break are represented in terms of n. Honest parties run in $p(n)$[1] time and are only concerned with adversaries who also run in $p(n)$ time. The probability of success is ensured to be negligible defined as:

DEFINITION (Negligible Functions). *A function* $f : \mathbb{N} \to \mathbb{R}$ *is negligible if*

$$\forall p(n), \exists n_0 \in \mathbb{N} : f(n) < 1/p(n) \forall n \geqslant n_0$$

Technically, $\mathcal{K}$ cannot be fixed, else we give away the key with $1/|\mathcal{K}|$ probability in constant time. Thus, $\mathcal{K}_n$ grows super-polynomially, so that probability of brute force search is negligible.
Assumptions in computational security are usually of the form:

The problem X cannot be solved by a probabilistic polynomial time algorithm except with negligible probability.

These assumptions are well tested and many are equivalent to the assumption $P \neq NP$.
Thus, a general proof of security for an asymptotically secure scheme Π under the above assumption and definition involves:

1. Fix a PPT adversary $\mathcal{A}$ attacking Π. The probability of success is denoted by $\epsilon(n)$.
2. Construct a PPT algorithm A' which uses $\mathcal{A}$ as a subroutine. This involves :
   - $\mathcal{A}$ only interacts with Π.

[1] $p(n)$ is throughout used to denote any polynomial in n

- A' does not interfere in $\mathcal{A}$'s working.
- If $\mathcal{A}$ breaks the scheme, A' solves X with $p^{-1}(n)$ probability.

3. By the second subpoint above, A' and $\mathcal{A}$ are independent. Thus, A' solves X with probability $\epsilon(n)/p(n)$. Thus, by assumption, $\epsilon(n)$ is negligible.

For computational security, the security parameter needs to be introduced into the syntax as:

DEFINITION (Private Key Encryption Scheme)**.** *A private key encryption scheme* Π *is a tuple* $(Gen(), Enc_k(), Dec_k())$*:*

1. $Gen(1^n)$ *: Outputs a random key* $k : |k| \geqslant n$*.*
2. $Enc_k()$ *: Outputs* $c \leftarrow Enc_k(m)$ *where* $\mathcal{M} = \{0,1\}^*$*.*
3. $Dec_k()$ *: Outputs* $m := Dec_k(c)$

The scheme needs to satisfy that $\forall k \in \mathcal{K}_n \,\&\, m \in \mathcal{M}_n; Dec_k(Enc_k(m)) = m \; \forall n \in \mathbb{N}$

4 Pseudorandom Constructs

Generating random elements from a set is an unsolved problem. The notion of pseudorandomness is thus defined as:

DEFINITION (Pseudorandom Distributions)**.** *A distribution* $\mathcal{D}$ *over a set* $\mathcal{S}_n$ *where* $|\mathcal{S}_n|$ *grows super-polynomially with* n *is pseudorandom if no* $p(n)$ *time algorithm can distinguish between* $\mathcal{D}$ *and the uniform distribution over* $\mathcal{S}_n$*. Usually, the set* $\mathcal{S}_n = \{0,1\}^n$

$\because$ asymptotic approach only deals with polynomial time adversaries, pseudorandom constructs can be safely used in this domain. The definitions of a few useful constructs follow.

4.1 Pseudorandom Generator

DEFINITION. *Let* l *be a polynomial function (called expansion factor)* : $l(n) > n \; \forall n \in \mathbb{N}$ *and* $\mathcal{G}$ *be a deterministic* $p(n)$ *time algorithm* : $\mathcal{G}(s)$ *outputs in* $\{0,1\}^{l(n)}$ *where* $s \in \{0,1\}^n$*.* $\mathcal{G}$ *is a pseudorandom generator if* $\forall$ *PPT distinguishers* D*,* $\exists$ *a negligible function negl such that :*

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

Where s *is chosen uniformly at random in* $\{0,1\}^n$ *and* r *is chosen uniformly at random in* $\{0,1\}^{l(n)}$*, and the probability includes randomness incurred by* $\mathcal{D}$*.*

A pseudorandom generator outputs $y \in \{0,1\}^{l(n)}$ with probability : $\dfrac{|\{s \in \{0,1\}^n : \mathcal{G}(s) = y\}|}{2^n}$.
Which is far from the uniform distribution where the probability is $2^{-l(n)}$.
The definition of a pseudorandom generator ensures that to a polynomial time adversaries, a random and a pseudorandom string are equivalent when s is kept secret.

A pseudorandom generator defined this way can only output strings of length $l(n)$. This is not feesible in application and hence the notion of variable output length pseudorandom generators:

DEFINITION. *A deterministic algorithm* $\mathcal{G}$ *is called a variable output-length pseudorandom generator if for* $s \in \{0,1\}^*$ *and* l *being a polynomial function,*

- $\mathcal{G}_l(s) := \mathcal{G}(s, 1^{l(||s||)})$*is a pseudorandom generator with expansion factor* l*.*
- $\forall s, l, l' : l < l'$*,* $\mathcal{G}(s, 1^l)$ *is a prefix to* $\mathcal{G}(s, 1^{l'})$*.*

4.2 Pseudorandom Functions

DEFINITION. *An efficient, keyed and length preserving function* $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *is called a pseudorandom function if* $\forall$ *PPT distinguishers* $\mathcal{D}$, $\exists$ *a negligible negl :*

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

Where $k \leftarrow \{0,1\}^n$ *and* $f_n \leftarrow Func_n$ *which is the set of all functions* $\{0,1\}^n \to \{0,1\}^n$.

$\mathcal{D}^{O()}$ represents oracle access i.e. $\mathcal{D}$ can ask for $O(x)$ by chosing any x in the right domain. A function is said to be efficient if $\exists$ a $p(n)$ time algorithm to find $F(x)$ provided $x \in D_n$ (domain) and a keyed function $F(k, x)$ is defined as $F_k(x)$ when k is supplied and fixed.

A lot of cryptographic schemes can be constructed using random functions. The definition of pseudorandom functions guarentees that they can be used safely in place of random functions.

4.3 Pseudorandom Permutations

Some applications require functions to be bijections, hence the notion.

DEFINITION. *An efficient, keyed and length conservative function* $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *is a called a pseudorandom permutation if* $\forall$ *PPT distinguishers* $\mathcal{D}$, $\exists$ *a negligible negl :*

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

When $k \leftarrow \{0,1\}^n$ *and* $f_n \leftarrow BijFunc_n$ *which is the set of all bijections* $\{0,1\}^n \to \{0,1\}^n$ *and* $\forall k$, F_k^{-1} *is also* $p(n)$ *computable.*

THEOREM. *A pseudorandom permutation is a pseudorandom function.*

Pseudorandom permutations are also called block ciphers. Practically used block ciphers do not come with a proof of pseudorandomness. These constructs are trusted w/o proofs because they are the building blocks in most schemes, $\therefore$ when found breachable, can be changed easily.

There are two major paradigms for constructing block ciphers :

4.3.1 Substitution-Permutation Networks

Structure : A SPN consists of *S-boxes* (substitutions) which are a set of small permutations and *mixing permutation* (an algorithm for mixing). The input is divided into smaller blocks and random permutations are run on these blocks (introducing *confusion*). Then, the output of the random functions is mixed together (introducing *diffusion*).

Implementing the key : The process counts as a *round* of the network. After doing this for multiple rounds, one expects to achieve pseudorandomness.
The key (k for F_k) can be introduced in two possible ways :

- k can determine which type of S-boxes and mixing permutations should be used.
- k and a set of keys derived from k by a process called *key scheduling* are mixed with the output after each round (the S-boxes and mixing permutations need to be public).

In practice, method 2 is preferred.

Design Practices : The design of S-boxes, mixing permutations and key schedule determines the cipher completely. There are some recommended design practices:

1. Invertible S-boxes :

   S-boxes should be bijections. This ensures that the block-cipher as a whole is *efficiently* invertible.

2. Avalanche Property :

   This property ensures that similar inputs don't produce similar outputs. It involves :

   - Changing 1 bit in input to any S-box changes at least 2 bits in the output.
   - Output bits of any S-box are spread across different S-boxes by mixing permutation.

   Thus, if just a single bit is changed in the input, after the first round, there are atleast 2 blocks with a single change. Thus, in $log_2(N)$ rounds, the avalanche effect is completed, where N is the block size and the output is expected to be pseudorandom.

4.3.2 Fiestel Networks

These use S-boxes and mixing permutations in a different way. In the build, the S-box need not be invertible. A fiestel network can be defined as :

CONSTRUCTION 2 (Fiestel Network)**.** *Consider a family* $f_i : \{0,1\}^{\frac{n}{2}} \to \{0,1\}^{\frac{n}{2}}$ *made using S-boxes and mixing permutations and involving key scheduling. Here* f_i *is the function specification in the* i^{th} *round and is determined by the key schedule. For an input* x *and key* k*:*

1. *Let* x_1, x_2 *be 2 equal parts of* x*.*
2. *Define* $v_1 = x_1$ *and* $v_2 = x_2$ *and decide an* r *(number of rounds).*
3. *for* $i = 1$ *to* r *:*
   - $w_1 = v_2$ *and* $w_2 = v_1 \oplus f_i(v_2)$
   - $v_1 = w_1$ *and* $v_2 = w_2$
4. *The output is* (v_1, v_2)

THEOREM. *A fiestel network is efficienty invertible irrespective of invertibility of* f_i*.*

Removing a restriction such as invertability is expected to increases chaos and thus, randomness.

4.3.3 DES

Data encryption standard (DES) is a 16 round fiestel network with a block size of 64 bits. It uses a master key k of length 56 bits from which a different key of length 48 bits is derived in each round by *key scheduling*. An initial permutation (IP) is applied to the input (64 bit) which is then split into 32 bit pieces to be used in the fiestel network.

Since the key used is 48 bits, the f function expands the 32 bit input into a 48 bit string by duplicating half the bits (known publically). The key and the bitstring are XORed and the result is divided into 8 blocks of size 6 bits. These are then passed through 8 S-boxes $(S_1, S_2, \cdots, S_8)$.

The S-boxes are 4×16 lookup tables and lookup is performed by using the first and last bit for row number and the middle 2-5 for column number. These are constructed such that :

- Each row is a permutation of $0, 1, 2, \cdots 15$.
- Changing a single bit in input changes atleast 2 bits in output.

A mixing permutation then ensures that the changed bits are scattered into different blocks.
The avalanche effect is expected to be completed after $\texttt{log}_2(32) = 5$ rounds.
Each detail about $S_1, S_2, \cdots S_8$, key scheduling, mixing permutations is publically known. Producing the key for a 1 or 2 round variant is trivial because the avalanche effect is incomplete.
An interesting property of DES is complement property, stated as:

THEOREM (Complement Property of DES)**.** *In a DES, if all bits of the key k and input $P = L||R$ are reversed (complemented), the output is complemented.*

Since the keyspace of DES is only of size 2^{56}, it is trivially broken by modern brute force attacks which byy courtsey of the complement property, brute force only uses 2^{55} computations.
There are two possible ways to increase the key size of DES :

- Internal tampering : Changing the internal structure of DES and using a larger key.
- Black box creation : Using the DES cipher as a black box unit multiple times.

Internal tamering is avoided as the current structure is trusted and no change can be justified.

Double DES A double DES $\widetilde{F}$ uses a 112 bit key $k_1||k_2$ as :

$$\widetilde{F}_{k_1||k_2}(x) = F_{k_2}(F_{k_1}(x))$$

There is *meet-in-the middle* attack for this cipher. If the key of F is assumed to be of length n, this attack uses $\mathcal{O}(2^n)$ time and space (time similar to brute force).
Given an input-output pair (x, y), the attack creates 2 lists of pairs (K_1, z_1) and (K_2, z_2) where $z_1 = F_{K_1}(x)$ and $z_2 = F^{-1}_{K_2}(y)$. The matching z in these lists indicate $(K_1, K_2) : K_1||K_2$ can be a valid key. We are expected to get 2^n such pairs (assuming F is random, this is an upper bound) and thus trying these pairs for more input-output pairs will leaves the key.
Double DES still cannot be considered secure as 2^56 time, space is practically achievable.

Triple DES A triple DES $\widehat{F}$ uses a 158 bit key $k_1||k_2||k_3$ as :

$$\widehat{F}_{k_1||k_2||k_3}(x) = F_{k_3}(F^{-1}_{k_2}(F_{k_1}(x)))$$

There has been no attack better than brute-force on 3DES till now.
The F^{-1} in middle is for backward compatibility for sindle DES.
A variant of 3DES exists using a 112 bit key $k_1||k_2$ as :

$$\widehat{F}_{k_1||k_2}(x) = F_{k_1}(F^{-1}_{k_2}(F_{k_1}(x)))$$

This has been proven to be broken in $\mathcal{O}(2^n)$ time and space given 2^n oracle queries can be made which is virtually impossible in real scenarios.

4.3.4 AES

Advanced Encryption Standard (AES) was the standard adopted after DES was known to be practically insecure. The scheme behind AES is *Rijndael*. It has a block length of 128 bits and a key size of 128, 192 or 256 bits.

The data structure used in DES is a 4×4 matrix of bytes (i.e. a vector in $gf(2^8)^{16}$ where gf denotes Galios Field). The operations in a round are :

1. *AddRoundKey* : Key k is derived from the master key by key schedule and represented as a 4×4 byte matrix. It is then added (XORed) with the input matrix.
2. *SubBytes* : A single specified 8 bit S-box is used to determine the new $a'_{i,j} = S(a_{i,j})$.
3. *ShiftRows* : The first row is left untouched. The second row is cylindrically moved left one space. The third row, two spaces and the fourth row, three.
4. *MixColumns* : The columns are mixed by an invertible linear transformation $M \in gf(2^8)^{16}$.

For a 128 bit AES, 10 rounds are carried out. In the last round, *MixColumns* is replaced by an additional *AddRoundKey* step to ensure that the last step is not simply reversible.

The structure of AES is similar to a substitution permutation network, viewing steps 1,2 to induce confusion and 3,4 to induce diffusion.

There exist *side channeled attacks* against AES which assume very high powers for the adversary.

4.4 Hash Functions

DEFINITION. *A hash function H maps a large domain $\mathcal{D}_H$ to a small range $\mathcal{R}_H$.*

Thus, by pigeonhole principle, there exist collisions defined as:

DEFINITION. *A collision is a pair of elements $x, x' \in \mathcal{D}_H$ such that*

$$x \neq x' \,\&\, H(x) = H(x')$$

In this case, it is said that x and x' collide under H.

Cryptographic constructs extensively use hash functions that have a collision resistant property. If a hash function has such a property, no PPT adversary can find a collision except with negligible probability. To formally define this, the notion of family of hashings is introduced:

DEFINITION. *A family of hash functions is a pair of PPT algorithms (Gen, H) :*

1. *$Gen(1^n)$: Outputs a key s : given s, n can be efficiently computed.*
2. *H : For $x \in \{0,1\}^*$, deterministically outputs $H_s(x) \in \{0,1\}^{l(n)}$ in $p(n)$ time.*

As a special case, if $\forall s$ output by Gen, H is defined only over input $x \in \{0,1\}^{\varphi(n)}$: $\varphi > l$, (Gen, H) is said to be a fixed length hash function with length parameter φ.

Surely l is a polynomial. This fact is implicit since H outputs in $p(n)$ time.

To define collision resistance, the experiment $\mathbf{HashColl}_{\mathcal{A},\mathbf{\Pi}}(\mathbf{n})$ is defined as:

1. $s \leftarrow Gen(1^n)$.
2. $(x, x') \leftarrow \mathcal{A}(s) : x, x' \in \{0,1\}^*$.
3. The experiment succeeds (1) iff $x \neq x'$ and $H_s(x) = H_s(x')$.

DEFINITION (Collision Resistance Hashing). *A family of hash functions $\Pi = (Gen, H)$ is collision resistant if $\forall$ PPT adversaries $\mathcal{A}$, $\exists$ a negligible function negl() :*

$$Pr[HashColl_{\mathcal{A},\Pi}(n) = 1] \leq negl(n)$$

One might argue as to why a family is used and not a single function.In that case, a definition of collision resistance would not be so simple. If x, x' is a collision in such a case, it is a collision in every run of the experiment and hence an adversary outputing x, x' always succeeds.

Two propertis weaker than collision resistance can be defined on hash functions as:

DEFINITION (Premiage Resistant Hashing). *A family of hash functions Π is premiage resistant if given a key $s(1^n)$ and a $y \in \{0,1\}^{l(n)}$, no PPT adversary can find an $x \in \{0,1\}^*$ such that $H(x) = y$ except with negligible probability.*

DEFINITION (Second Premiage Resistant Hashing). *A family of hash functions $\Pi = (Gen, H)$ is second premiage resistant it is hard to find a PPT adversary $\mathcal{A}$, such that $\exists$ polynomial p :*

$$Pr[HashColl_{\mathcal{A},\Pi}(n) = 1] \geq \frac{1}{p(n)}$$

The weakness is implied by the fact that:

THEOREM. *Collision Resistance $\Rightarrow$ Second Premiage Resistance $\Rightarrow$ Premiage Resistance.*

For a family of hash functions to be collision resistant, the output length has to honor a certain lower bound. This is due to the proposition of the *birthday problem*. Formally,

THEOREM. *For a family of hash functions outputing in $\{0,1\}^{l(n)}$ to be collision resistant under attack by an adversary which runs in time T, it is neccessary that $l(n) \geq 2 \cdot log(T(n))$.*

Arbitrary length hash functions are not easily constructed. These can be obtained from fixed length hash functions by Merkel Damgård transform as:

CONSTRUCTION 3 (Merkle Damgård Transform). *Given a fixed-length family of hash functions (Gen_h, h) with length parameter $2l$ (output parameter = l), define $\Pi = (Gen, H)$ as:*

1. *$Gen(1^n)$: Outputs $s \leftarrow Gen_h(1^n)$.*
2. *$H_s(x)$: For any $x \in \{0,1\}^*$, define $L := ||x||$*
   Assuming $L \leq [2^{l(n)} - 1]$, pad x with 0s to make it a multiple of l
   Define $B := \lceil \frac{L}{l} \rceil$ and $z_0 = 0^{l(n)}$ and $z_i = h_s(z_{i-1}||x_i) \forall i \in \{1, 2, \cdots B\}$.
   (here x_i is the i^{th} $l(n)$ length prefix of x)
   Output $H_s(x) = h_s(z_B||L)$.

The assumption $L \leq [2^{l(n)} - 1]$ is taken so that finally L can be trivially mapped to a $l(n)$ length bitstring. This is also valid $\because$ collision resistance is provided only for PPT adversaries.

THEOREM. *If the supplied fixed-length family of hash functions is collision resistant, the Merkle Damgård construction provides a collision resistant family of hash functions.*

The z_0 in the construct can be changed to any k to yeild keyed hash functions h^k and H^k

4.5 One-Way Functions

DEFINITION. *A function $f : \{0,1\}^* \to \{0,1\}^*$ is called a one-way function if it is :*

- ***Easy to compute*** *: $\exists$ a PPT algorithm M_f : $M_f(x) = f(x) \forall x \in \{0,1\}^*$*
- ***Hard to invert*** *: $\forall$ PPT inversion algorithms $\mathcal{A}$, $\exists$ a negligible $negl()$: $Pr[\mathcal{A}(f(x)) \in f^{-1}(f(x))^2] \leq negl(n)$ where x is chosen uniform randomly in $\{0,1\}^n$*

On similar lines, one can define one-way permutations (bijections):

DEFINITION. *A one-way function $f : \{0,1\}^* \to \{0,1\}^*$ is a one-way permutation if $\forall n \in \mathbb{N}$, the restriction of f : f_n whose domain is $\{0,1\}^n$ is a bijection on $\{0,1\}^n$.*

A family of functions is defined as:

DEFINITION. *A tuple $\Pi = (Gen, Samp, f)$ of PPT algorithms is called a family of functions:*

1. *$Gen(1^n)$: Outputs I ($|I| \geq n$) which defines two sets $\mathcal{D}_I$ and $\mathcal{R}_I$.*
2. *$Samp(I)$: Outputs a uniformly distributed element of $\mathcal{D}_I$ (except with probability negligible in $|I|$).*
3. *$f_I(x)$: Is a deterministic algorithm that on input $x \in \mathcal{D}_I$ outputs $y \in \mathcal{R}_I$.*

A family of permutations can be defined similarly as:

DEFINITION. *A family of permutations is a family of functions which has the property that $\forall I$ output by Gen, $\mathcal{D}_I = \mathcal{R}_I$ and $f_I : \mathcal{D}_I \to \mathcal{R}_I$ is a bijection.*

THEOREM. *The output of a family of permutations is uniformly distributed.*

The one way nature of these families is defined by the experiment $\mathbf{Invert}_{\mathcal{A},\mathbf{\Pi}}(\mathbf{n})$:

1. $I \leftarrow Gen(1^n)$, $x \leftarrow Samp(I)$ and $y := f_I(x)$.
2. $x' \leftarrow \mathcal{A}(y, I)$
3. The experiment is successful (1) iff $f(x') = y$.

The definition of a family of one-way functions follows:

DEFINITION. *A family of functions Π is called one-way if $\forall$ PPT algorithms $\mathcal{A}$, $\exists$ a negligible function $negl()$:*

$$Pr[Invert_{\mathcal{A},\Pi}(n) = 1] \leq negl(n)$$

One-Way functions are minimal cryptographic primitives. Given one-way functions and permutations, one can produce pseudorandom functions and permutations and vice versa.

Another class of one-way constructions are trapdoor permutations. These are defined as:

DEFINITION (Trapdoor Permutation)**.** *A tuple of PPT algorithms $(Gen, Samp, f)$ is called a trapdoor permutation if the following conditions hold:*

[2]This defines the set of all $x' : f(x') = f(x)$

- $Gen(1^n)$ *outputs* $(I, td) : |I| > n$ *and all such pairs define* $\mathcal{D}_I = \mathcal{D}_{td}$
- $(Gen', Samp, f)$ *forms a one-way permutation where* Gen' *runs* Gen *and outputs only* I
- $\exists$ *a deterministic polynomial time algorithm* $Inv : Inv_{td}(f(x)) = x \ \forall x \in \mathcal{D}_{td}$

A trapdoor permutation is one that has can be efficiently inverted only when one has access to the trapdoor *td*. Some trapdoor permutations have a special function called *hard core predicate*.

DEFINITION. *Let* Π *be a trapdoor permutation. A PPT algorithm hc is called a hard-core predicate for* Π *if* $hc_I(x)$ *for* $x \in \mathcal{D}_I$ *outputs a single bit such that* $\forall$ *PPT adversaries* $\mathcal{A}$ *outputing a single bit,* $\exists$ *a negligible negl :*

$$Pr[\mathcal{A}(I, f_I(x)) = hc(x)] \leq \frac{1}{2} + negl(n)$$

Where the probability is over selection of $I \leftarrow Gen(1^n)$ *and then selection of* $x \leftarrow \mathcal{D}_I$

5 Message Authentication Codes

DEFINITION. *The property that ensures that a message is authentic and unaltered in the course of transmission over a communication channel is called message integrity.*

It is impossible to prevent message tampering over a channel, thus the goal of constructing message authentication codes is to *detect and discard*.

DEFINITION. *A MAC is a tuple of PPT algorithms* $(Gen, Mac, Vrfy)$ *such that :*

1. $Gen(1^n)$ *outputs a key* $k \leftarrow \{0,1\}^n$.
2. $Mac_k(m)$ *For* $m \in \{0,1\}^*$ *outputs a tag* $t \in \{0,1\}^*$ *called MAC-tag.*
3. $Vrfy_k(m, t)$ *outputs a bit* $b \in \{0, 1\}$

The validity of a MAC is implied by the existence of a negligible $negl()$*:*

$$\forall n \in \mathbb{N}, k \in \{0,1\}^n, m \in \{0,1\}^*; Pr[Vrfy_k(m, Mac_k(m)) = 1] > 1 - negl(n)$$

If for a MAC, $\exists l(n)$ *such that* $Mac_k(m)$ *is defined only for* $m \in \{0,1\}^{l(n)}$ *and* $Vrfy_k(m, t)$ *outputs 0* $\forall m \notin \{0,1\}^{l(n)}$*, then it is called a fixed length MAC with a length parameter* l*.*

One needs to define a MAC such that no illegal or forged messages are verified positively.
The experiment $\mathbf{MacForge}_{\mathcal{A},\Pi}(\mathbf{n})$ is defined as:

1. $k \leftarrow Gen(1^n)$
2. $(m, t) \leftarrow \mathcal{A}^{Mac_k()}(1^n)$ and all the queries are stored in the set $\mathcal{Q}$
3. The experiment succeeds (1) iff $m \notin \mathcal{Q}$ and $Vrfy_k(m, t) = 1$

DEFINITION. *A message authentication code* Π *is existentially unforgeable under a chosen message attack if* $\forall$ *PPT adversaries* $\mathcal{A}$*,* $\exists$ *a negligible function* $negl()$ *:*

$$Pr[MacForge_{\mathcal{A},\Pi}(n) = 1] \leq negl(n)$$

Length of the tag has to be large enough to avoid guessing with non-negligible probability.

THEOREM. *The maximum length of a MAC-tag for a message m and parameter n is denoted by $q(n + |m|)$. For a MAC to be secure, $q()$ must grow super logarithmic w.r.t n.*

The definition of MAC-security does not prevent a `replay attack`, i.e. an attack in which a legal message is intercepted and replayed multiple times. The transmitted message is unforged and hence, this attack is not a break by definition of security. Such attacks are to be handled at application level and are usually dealt by adding a transaction number to a message and making sure that only one message of a particular transaction number is marked valid.

A fixed length MAC with $l(n) = n$ can be constructred using a pseudorandom function as:

CONSTRUCTION 4 (Pseudorandom MAC)**.** *For a pseudorandom F, define a MAC Π as:*

1. *$Gen(1^n)$: Outputs $k \leftarrow \{0,1\}^n$*
2. *$Mac_k(m)$: Takes as input, message m and outputs $(m, F_k(m))$. If $|k| \neq |m|$, output $\perp$.*
3. *$Vrfy_k(m,t)$: If $t = \perp$, output 0. Else, output 1 if $F_k(m) = t$.*

THEOREM. *The Fixed-Length MAC Π described in construction 4 is existentially unforgeable.*

A variable-length MAC can be constructed by breaking the message into blocks and applying F_k to individual blocks and using the tuple generated as the tag. However, then it is easy to modify the message by repetition or deletion of blocks. This problem is overcome by encoding extra information into the blocks, namely :

- *Index* : To make sure the blocks cannot be rearranged.
- *Total Number of Blocks* : To ensure that blocks are not dropped from the end.
- *Random Identifier* : This is same for every block but probabilistically ensures that two different messages can't be intermixed. (It is possible to get blocks from two different messages with same index and number of blocks)

CONSTRUCTION 5 (Variable-Length MAC)**.** *For a pseudorandom F, define a MAC Π as :*

1. *$Gen(1^n)$: Outputs $k \leftarrow \{0,1\}^n$*
2. *$Mac_k(m)$: For $m \in \{0,1\}^*$ length of which should be at most $2^{\frac{n}{4}-1}$. Parse m into d blocks $(m_1, m_2, \cdots m_d)$ each of length $\frac{n}{4}$ (Pad the last block with $100\cdots$). Chose a random identifier $r \leftarrow \{0,1\}^{\frac{n}{4}}$. For $i \in \{1, 2, \cdots d\}$, compute $t_i = F_k(r||i||d||m_i)$ where i and d are uniquely encoded into $\frac{n}{4}$ size strings. Finally, output $(r, t_1, t_2, \cdots t_d)$*
3. *$Vrfy_k(m,t)$: Runs $Mac_k(m)$ but uses r supplied in t. If $t = Mac_k(m)$, returns 1.*

THEOREM. *The MAC Π described in construction 5 is existentially unforgeable under cma.*

The construction is inefficient as the tag length is 4 times the input length. To encounter this, CBC-MACs are constructed which are fixed length but the expansion factor can be large.

CONSTRUCTION 6 (Basic CBC-MAC)**.** *For a pseudorandom F, define MAC Π as :*

1. *$Gen(1^n$: Output $k \leftarrow \{0,1\}^n$*

2. $Mac_k(m)$: *For* $m = (m_1, m_2, \cdots m_l)$, *where each* $|m_i| = n$. *Initialize* $t_0 = 0^n$ *and define* $t_i := F_k(m_i \oplus t_{i-1}) \forall i \in \{1, 2, \cdots l\}$. *Output* t_l.

3. $Vrfy_k(m, t)$: *Output* 1 *iff* $t = Mac_k(m)$.

THEOREM. *The CBC-MAC described in the construction 6 is a fixed-length MAC with length parameter* $l \cdot n$ *which is existentially unforgeable under chosen message attacks.*

However, if arbitrary length[3] messages are allowed to be queried, the MAC is no longer secure. An example adversary $\mathcal{A}$ can be constructed as :

1. Chose $m_1 \in \{0,1\}^n$. Get $t_1 := Mac_k(m_1) = F_k(m_1)$.
2. Chose $m_2 \in \{0,1\}^n$. Get $t_2 := Mac_k(m_2 \oplus t_1) = F_k(m_2 \oplus t_1)$ (i.e. query $m_2 \oplus t_1$).
3. Consider message $m_1||m_2$. Surely, $Mac_k(m_1||m_2) = F_k(m_2 \oplus F_k(m_1 \oplus 0^n)) = t_2$. Thus, we have forged $m_1||m_2$ with certainity and $m_1||m_2 \notin \mathcal{Q} = \{m_1, m_2 \oplus t_1\}$.

Variations of CBC-MAC are proven secure. Some of them are :

- The key k and the length l are used to determine another key $k_l = F_k(l)$. Now, this key is used to encode $t_i = F_{k_l}(t_{i-1} \oplus m_i) : t_0 = 0^n$. The tag consists of (l, t_l).
  This method is intuatively secure against the type of attack used above as
  $$Mac_k((2, m_1||m_2)) = F_{k_2}(m_2 \oplus F_{k_2}(m_1)) \neq t_2 = F_{k_1}(m_2 \oplus F_{k_1}(m_1))$$
  As a disadvantage, l must be evaluated before starting bit processing.
- The length l is prefixed to the message m as a n bit string. CBC-MAC is proven to be secure if no 2 messages which are prefixes of one-another are used. Addind l as prefix to m serves as a special case. This also involves knowing l before starting bit processing.
- This approach is called CBC-MAC-ELB$(m, (k_1, k_2))$ or *Encrypt Last Block* method. It involves using a pair of keys, k_1, k_2. First, t is computed as in the Basic CBC-MAC using k_1 and then, $t' := F_{k_2}(t)$ is used as the tag. It is slightly disadvantageous as it uses 2 keys. But, it offers the advantage that l need not be known to compute t, thus, saving processing power.

Another construction of MACs uses collision resistant keyed hash functions as:

CONSTRUCTION 7 (NMAC)**.** *Let* h *be a* $2l \to l$ *compressor and let* H *be a hash function defined by Merkle Damgård transform on* h. *Given* (Gen, H) *and* (Gen_h, h), *define* Π *as:*

1. $\widehat{Gen}(1^n)$: *Outputs* $k = (s, k_1, k_2)$ *where* $s \leftarrow Gen(1^n)$ *and* $k = (k_1, k_2) : k_1, k_2 \leftarrow \{0,1\}^{l(n)}$
2. $Mac_k(m)$: *For* $m \in \{0,1\}^*$, *output* $t = NMAC_s^{k_1,k_2}(m) := h_s^{k_1}(H_s^{k_2}(m))$
3. $Vrfy_k(m, t)$: *Outputs 1 iff* $t = Mac_k(m)$.

[3]By this, it is implied that l is not fixed and is deduced from message length

THEOREM. *Consider the Nested MAC defined in construction 7. If* $(\widetilde{Gen_h}, \widetilde{h})$ *and* $(\widetilde{Gen}, \widetilde{H})$ *are defined as* $\widetilde{Gen}, \widetilde{Gen_h}$ *being same as* Gen, Gen_h *but also including a random selection of a secret key* $k \leftarrow \{0,1\}^{l(n)}$ *and* $\widetilde{H}_{s,k}, \widetilde{h}_{s,k}$ *being* H_s^k *and* h_s^k *respectively. If* $(\widetilde{Gen}, \widetilde{H})$ *is collision resistant and MAC using* $(\widetilde{Gen_h}, \widetilde{h})$ *is fixed-length existentially unforgeable, then, NMAC is a secure arbitrary length message authentication code.*

NMAC is not easy to implement due to multiple keys and keyed hashings used. Thus, HMAC is defined to cope up with the limitations. It uses unkeyed (Gen, H) and h along with 2 arbitrarily chosen fixed random strings of length $l(n)$: `opad` and `ipad`:

CONSTRUCTION 8 (HMAC). *Define* $\Pi = (Gen, Mac_k, Vrfy_k)$ *as:*

1. $Gen(1^n)$ *: Choses* $s \leftarrow Gen_h(1^n)$ *and private key* $k \leftarrow \{0,1\}^{l(n)}$
2. $Mac_k(m)$ *: Receives* (s,k) *and* $m \in \{0,1\}^*$ *and outputs*

$$t = HMAC_s^k(m) := H_s(k \oplus \texttt{opad} || H_s(k \oplus \texttt{ipad} || m))$$

3. $Vrfy_k(m,t)$ *: Outputs 1 iff* $t = Mac_k(m)$

HMAC can be viewed as a special case of NMAC by considering first steps of computation:

- $H_s(k \oplus \texttt{ipad}||m) = H_s^{h_s(z_0||k\oplus\texttt{ipad})}(m)$
- $H_s(k \oplus \texttt{opad}||H_s(k \oplus \texttt{ipad}||m)) = H_s^{h_s(z_0||k\oplus\texttt{opad})}(H_s(k \oplus \texttt{ipad}||m))$
  $= h_s^{h_s(z_0||k\oplus\texttt{opad})}(H_s(k \oplus \texttt{ipad}||m)) = h_s^{h_s(z_0||k\oplus\texttt{opad})}(H_s^{h_s(z_0||k\oplus\texttt{ipad})}(m))$

Since, h_s^k is considered a secure MAC, it is pseudorandom. Thus, HMAC can be viewed as NMAC with $k_1 = h_s(z_0||k \oplus \texttt{opad})$ and $k_2 = h_s(z_0||k \oplus \texttt{ipad})$.Thus, HMAC is also unforgeable.

6 Private Key Encryption Schemes

These are studied under headings referring to powers of an adversary:

6.1 EAV

The adversary here has the minimal power of eavesdropping on a single ciphertext communicated. The experiment $\mathbf{PrivK}^{\mathbf{eav}}_{\mathcal{A},\mathbf{\Pi}}(\mathbf{n})$ is defined as :

1. $m_0, m_1 \leftarrow \mathcal{A}(1^n) : m_0, m_1 \in \mathcal{M}_n$ and $|m_0| = |m_1|$
2. $k \leftarrow Gen(1^n)$, $b \leftarrow \{0,1\}$, $c \leftarrow Enc_k(m_b)$ and $b' \leftarrow \mathcal{A}(c)$
3. The experiment is a success (1) iff $b = b'$.

DEFINITION. *A private key encryption scheme* Π *has indistinguishable encryptions under the presence of an eavesdropper if* $\forall$ *PPT adversaries* $\mathcal{A}$, $\exists$ *a negligible function* $negl(n)$ *:*

$$Pr[PrivK^{eav}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Where the probability includes all the randomness incurred by $\mathcal{A}$ *and the experiment.*

Similar to perfect secrecy, there is an alternate definition to indistinguishability. The definition makes use of $output(PrivK^{eav}_{\mathcal{A},\Pi}(n,b))$ which is the bit output by $\mathcal{A}$ given an encryption of m_b.

DEFINITION. *An encryption scheme* Π *is indistinguishable in presence of an eavesdropping adversary if* $\forall$ *PPT adversaries* $\mathcal{A}$, $\exists$ *a negligible function* $negl(n)$ *:*

$$|Pr[output(PrivK^{eav}_{\mathcal{A},\Pi}(n,0)) = 1] - Pr[output(PrivK^{eav}_{\mathcal{A},\Pi}(n,1)) = 1]| \leq negl(n)$$

THEOREM. *The two definitions of Computational Indistinguishability are equivalent.*

To assess the strength of the definition of security, the following can be observed:

THEOREM. *Let* Π *be an EAV secure encryption scheme. Then,* $\forall$ *PPT adversaries* $\mathcal{A}$ *trying to guess the* i^{th} *bit of* m (m^i) *given* $Enc_k(m)$, $\exists$ *a negligible function* $negl(n)$*:*

$$Pr[\mathcal{A}(1^n, Enc_k(m)) = m^i] \leq \frac{1}{2} + negl(n)$$

Where m *is chosen uniform randomly from* $\mathcal{M}_n$

THEOREM. *Let* Π *be an EAV secure encryption scheme. For any PPT adversary* $\mathcal{A}$, $\exists$ *a PPT algorithm* A' *such that for every efficient function* f, $\exists$ *a negligible function* $negl$ *:*

$$|Pr[\mathcal{A}(1^n, Enc_k(m)) = f(m)] - Pr[A'(1^n) = f(m)]| \leq negl(n)$$

Where m *is chosen uniform randomly from* $\mathcal{M}_n$

The accepted notion of security is semantic security. For defining semantic security, define:

1. $h(m)$ is the external knowledge an adversary might gain about the message m in $p(n)$ time.
2. $X = (X_1, X_2, \cdots)$ is a distribution vector. Each X_i represents a distribution over $\mathcal{M}_i$ which is efficiently samplable and the length of strings in X_i is strictly a function of i (security parameter).

DEFINITION (Semantic Security under Eavesdropping)**.** *A private key encryption scheme* Π *is semantically secure in presence of an eavesdropper if* $\forall$ *PPT algorithms* $\mathcal{A}$, $\exists$ *a PPT algorithm* $\mathcal{A}'$ *such that for any* $p(n)$ *computable functions* f *and* h*, and any efficiently samplable* $X = (X_1, X_2, \cdots)$, $\exists$ *a negligible function* $negl$ *such that*

$$|Pr[\mathcal{A}(1^n, Enc_k(m), h(m)) = f(m)] - Pr[\mathcal{A}'(1^n, h(m)) = f(m)]| \leq negl(n)$$

Where m *is chosen according to* X_n*.*

THEOREM. *The definition of indistinguishable encryptions in presence of eavesdropper and semantic security under eavesdropping are equivalent.*

An EAV secure scheme can be constructed as:

CONSTRUCTION 9 (Fixed-Length Asymptotic One-Time Pad)**.** *Given a pseudorandom generator* $\mathcal{G}$ *with expansion factor* l*, define a fixed length encryption scheme* Π *as:*

1. $Gen(1^n) : k \leftarrow \{0,1\}^n$.
2. $Enc_k(m) : c \leftarrow G(k) \oplus m$
3. $Dec_k(c) : m \leftarrow G(k) \oplus c$

THEOREM. *The fixed length asymptotic one-time pad has indistinguishable encryptions in presence of an eavesdropper.*

To resolve the restriction of fixed length, one can use a variable output-length generator.

6.2 Multiple EAV

The adversary here can eavesdrop on multiple ciphertexts.
The experiment $\mathbf{PrivK}^{\mathbf{multi}}_{\mathcal{A},\mathbf{\Pi}}(\mathbf{n})$ is defined as:

1. $\overrightarrow{M_0}, \overrightarrow{M_1} \leftarrow \mathcal{A}(1^n) : \overrightarrow{M_j} = \langle m_j^1, m_j^2, \dots m_j^l \rangle$ for $j = 0, 1$ and $\forall i \in \{1, 2, \cdots l\}$, $|m_0^i| = |m_1^i|$.
2. $k \leftarrow Gen(1^n)$, $b \leftarrow \{0,1\}$, $c^i \leftarrow Enc_k(m_b^i)\, \forall i \in \{1, 2, \cdots l\}$ and $\overrightarrow{C} := \langle c^1, c^2, \dots c^l \rangle$.
3. $b' \leftarrow \mathcal{A}(\overrightarrow{C})$ and the experiment succeeds (1) iff $b = b'$.

DEFINITION. *A private key encryption scheme* Π *has multiple indistinguishable encryptions in presence of an eavesdropper if* $\forall$ *PPT adversaries* $\mathcal{A}$, $\exists$ *a negligible function* $negl(n)$ *:*

$$Pr[PrivK^{multi}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Where the probability includes all the randomness incurred by $\mathcal{A}$ *and the experiment.*

There is one important requirement for a scheme to have multiple indistinguishable encryptions.

THEOREM. *If scheme* Π *has multiple indistinguishable encryptions in presence of an eavesdropper, then* $Enc_k(m)$ *can not be a deterministic algorithm.*

Multiple encryptions can be managed by using *stream ciphers* (pseudorandom generators) as:

1. Synchronized : The communicating parties use different parts of the same stream while communicating. This method requires maintaining a *state* of the stream, and thus, can only be used for a single session.
2. Unsynchronized : For an unsynchronized mode, no stream maintainance is required but a strengthened pseudorandom generator G is used takes two inputs, a seed s and an initialization vector $\overrightarrow{IV}$ of length n. $G(s, \overrightarrow{IV})$ is to be pseudorandom even when $\overrightarrow{IV}$ is known but s is kept secret i.e. for $\overrightarrow{IV_1}, \overrightarrow{IV_2}$, the streams $G(s, \overrightarrow{IV_i})$ viewed with the respective $\overrightarrow{IV_i}$ are pseudorandom. One can then define Enc as : $Enc_k(m) = \langle \overrightarrow{IV}, G(k, \overrightarrow{IV}) \oplus m \rangle$, where $\overrightarrow{IV}$ is chosen uniformly at random. Decryption involves $m := G(k, \overrightarrow{IV}) \oplus c$.
   This way, Enc is not derterministic. The security of this setup is proven.

6.3 CPA

The adversary has an unrestricted access to the encryption oracle.
The experiment $\mathbf{PrivK}^{\mathbf{cpa}}_{\mathcal{A},\mathbf{\Pi}}(\mathbf{n})$ is defined as:

1. $k \leftarrow Gen(1^n)$
2. $m_0, m_1 \leftarrow \mathcal{A}^{Enc_k()}(1^n) : m_0, m_1 \in \mathcal{M}_n$ and $|m_0| = |m_1|$
3. $b \leftarrow \{0,1\}$ and challenge ciphertext (c) is produced by $c \leftarrow Enc_k(m_b)$
4. $b' \leftarrow \mathcal{A}(c)$ and the experiment is successful (1) iff $b = b'$

DEFINITION. *A private key encryption scheme* Π *has indistinguishable encryptions under a chosen plaintext attack (is CPA-secure) if* $\forall$ *PPT adversaries* $\mathcal{A}$, $\exists$ *a negligible function negl :*

$$Pr[PrivK^{cpa}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Where the probability includes all randomness incurred by $\mathcal{A}$ *and the scheme.*

An important upside of this definition of security is that:

THEOREM. *A private key encryption scheme* Π *is CPA-secure iff it is multi-CPA-secure.*

Thus, extension of such schemes for multiple and arbitrary length messages becomes easier. CPA secure schemes are multi-EAV secure, $\therefore$ these can't have deterministic encryptions too. The construction of such a scheme can be done using pseudorandom functions as:

CONSTRUCTION 10. *Given a pseudorandom function* F*, define the scheme* Π *as :*

1. $Gen(1^n)$ *: Outputs a key* $k \leftarrow \{0,1\}^n$ *uniform randomly.*
2. $Enc_k(m)$ *: Choses* $r \leftarrow \{0,1\}^n$ *uniform randomly and outputs* $c := (r, F_k(r) \oplus m)$.
3. $Dec_k(c)$ *: Receives* $c = (r, s)$. *Outputs* $m := F_k(r) \oplus s$.

THEOREM. *The construct* Π *defined in construction 10 is CPA secure.*

The efficiency of CPA encryption schemes is measured using *ciphertext expansion.*

DEFINITION. *Difference between length of ciphertext and plaintext in an encryption scheme is reffered to as ciphertext expansion. Goal of efficient schemes is to minimize this.*

Such schemes are usually used by fixing a block length $= n$ (security parameter) and padding every message to be of length multiple of n. Then, the l parts $(m_0, m_1, \ldots m_l)$ are transmitted. Typical modes of operation using a block cipher include:

Electronic Code Book (Insecure) The ECB mode functions by computing

$$Enc_k(m) := (F_k(m_1), F_k(m_2), \cdots F_k(m_l))$$

Decryption is just inverse calculation. This mode is deterministic, hence not secure.

Naive Mode This mode creates a ciphertext double the size of plaintext.
$r_i \leftarrow \{0,1\}^n$ for each $i \in \{1, 2, \cdots n\}$ and

$$c \leftarrow Enc_k(m) := ((r_1, F_k(r_1) \oplus m_1), (r_2, F_k(r_2) \oplus m_2), \cdots (r_l, F_k(r_l) \oplus m_l))$$

The mode is CPA-secure but the ciphertext expansion is $l \cdot n$.

Ciphertext Block Chaining (CBC) This mode makes use of an $\overrightarrow{IV} \leftarrow \{0,1\}^n$.

$$c : c_0 = \overrightarrow{IV}; c_i = F_k(c_{i-1} \oplus m_i) \, \forall \, i \in \{1, 2, \cdots n\}$$

Decryption can be carried out in the reverse manner using F_k^{-1}. The mode is CPA-secure with ciphertext expansion n. It is inefficient since it cannot make use of parellel computation.

Output Feedback Mode (OFB) This mode creates a stream from a block cipher as:

$$r : r_0 = \overrightarrow{IV} \leftarrow \{0,1\}^n; r_i = F_k(r_{i-1}) \, \forall \, i \in \{1, 2, \cdots n\}$$

The encryption occurs as :

$$c \leftarrow Enc_k(m) = (\overrightarrow{IV}, r_1 \oplus m_1, r_2 \oplus m_2, \cdots r_l \oplus m_l)$$

The mode is CPA-secure (even when F is a pseudorandom function) with ciphertext extension n. It too, does not use parellel computation but here r can be preprocessed.

Counter Mode (CTR) This mode is similar to OFB. The stream creation follows:

$$ctr = \overrightarrow{IV}; r_i = F_k(ctr + i) \, \forall \, i \in \{1, 2, \cdots n\}$$

The encryption includes :

$$c = Enc_k(m) := (ctr, r_1 \oplus m_1, r_2 \oplus m_2, \cdots r_l \oplus m_l)$$

The mode is also CPA-secure (even when F is a pseudorandom function) with ciphertext extension n. The ciphertext can be generated using parellel computation and random access is allowed (i.e. decryption can be carried out in any order).

6.4 CCA

The adversary has access to both Enc_k and Dec_k oracles restricted to the condition that it cannot ask for decryption of the challenge ciphertext.
The experiment $\mathbf{PrivK^{cca}_{\mathcal{A},\Pi}(n)}$ is defined as:

1. $k \leftarrow Gen(1^n)$
2. $m_0, m_1 \leftarrow \mathcal{A}^{Enc_k(), Dec_k()}(1^n) : \, m_0, m_1 \in \mathcal{M}_n$ and $|m_0| = |m_1|$
3. $b \leftarrow \{0,1\}$and the challenge ciphertext is produced as $c \leftarrow Enc_k(m_b)$ and output to $\mathcal{A}$
4. $\mathcal{A}$ retains adaptive access to $Enc_k()$ and $Dec_k()$ but can-not query $Dec_k(c)$
5. The experiment succeeds (1) iff $b = b'$

DEFINITION (CCA-Secure Scheme)**.** *A scheme* Π *has indistinguishable encryptions under a chosen ciphertext attack (is CCA-secure) if* $\forall$ *PPT adversaries* $\mathcal{A}$*,* $\exists$ *a negligible function negl :*

$$Pr[PrivK^{cca}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Where the probability includes all randomness incurred by $\mathcal{A}$ *and the scheme.*

Any CCA secure shceme is CPA secure. Such schemes can be constructed using MACs as:

CONSTRUCTION 11 (CPA-then-MAC encryptions)**.** *Let* $\Pi_E = (Gen_E, Enc, Dec)$ *be an encryption scheme and* $\Pi_M = (Gen_M, Mac, Vrfy)$ *be a MAC. Define* $\Pi = (Gen', Enc', Dec')$ *as:*

1. *$Gen'(1^n)$: Outputs a pair of keys $k' = (k_1, k_2) : k_1, k_2 \leftarrow \{0,1\}^n$*
2. *$Enc'_k(m)$: Computes $c \leftarrow Enc_{k_1}(m)$ and $t := Mac_{k_2}(c)$ and outputs (c,t)*
3. *$Dec'_k(c,t)$: If $Vrfy_{k_2}(c,t) = 0$, outputs $\perp$ and otherwise outputs $Dec_{k_1}(c)$.*

THEOREM. *Let Π_E be a CPA secure encryption scheme and Π_M be a secure MAC with unique tags. Then, the scheme Π described in construction 11 is CCA secure.*

The scheme produced is also an unforgeable MAC.

7 Number Theoretic Hardness Assumptions

As seen, cryptographic schemes usually rely on tested assumptions. Such assumptions are commonly derived from problems in group theory and number theory. Some are:

7.1 The Factorization Assumption

A `GenModulus` is defined as a PPT algorithm which on input 1^n outputs $(N,p,q) : N = pq$ where p,q are primes : $|p| = |q| = n$ with probability of failure negligible in n.
This is used to define the factoring experiment $\mathbf{Factor}_{\mathcal{A},\texttt{GenModulus}}(\mathbf{n})$:

1. $(N,p,q) \leftarrow \texttt{GenModulus}(\texttt{1}^\texttt{n})$
2. $(p',q') \leftarrow \mathcal{A}(N) : p', q' > 1$
3. The experiment is successful (1) iff $N = p'q'$.

By definition of `GenModulus`, except with negligible probability, $\{p,q\} = \{p',q'\}$.
It is said that factoring is *hard* relative to `GenModulus` if $\forall$ PPT adversaries $\mathcal{A}$, $\exists$ a negligible function $negl()$:

$$Pr[Factor_{\mathcal{A},\texttt{GenModulus}}(n) = 1] \leq negl(n)$$

The factoring assumption states that :

There exists a `GenModulus` *relative to which factoring is hard*

7.2 The RSA Assumption

A `GenRSA` is a PPT algorithm which on input 1^n outputs an integer N (product of 2 n bit primes) and a pair of integers (e,d) where e,d are relatively prime to $\phi(N)$ and $ed \equiv 1 \bmod \phi(N)$.
This is used to define the RSA experiment $\mathbf{RSAinv}_{\mathcal{A},\texttt{GenRSA}}(\mathbf{n})$:

1. $(N,e,d) \leftarrow \texttt{GenRSA}(1^n)$
2. $y \leftarrow \mathbb{Z}_N^*$
3. $\mathcal{A}$ receives N, e, y and outputs $x \in \mathbb{Z}_N^*$
4. The experiment succeeds(1) iff $x^e \equiv y \bmod N$.

The RSA problem is said to be `hard` wrt a `GenRSA` if $\forall$ PPT adversaries $\mathcal{A}$, $\exists$ a negligible function $negl()$:

$$Pr[RSAinv_{\mathcal{A},\texttt{GenRSA}}(n) = 1] \leq negl(n)$$

The factoring assumption states that :

There exists a `GenRSA` *relative to which the RSA problem is hard*

7.3 Diffie-Hellman Assumptions

Let $\mathcal{G}$ be a PPT algorithm which takes as input 1^n and outputs a cyclic group description $\mathbb{G}$ on which the group operation is efficiently computable with order $q : ||q|| = n$ and a generator g.
This is used to define three problems:

7.3.1 The Discrete Logarithm Assumption

the experiment $\mathbf{DLog}_{\mathcal{A},\mathcal{G}}(\mathbf{n})$ is defined as:

1. $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$
2. $h \leftarrow \mathbb{G}$ (efficiently computable because isomorphous with $\mathbb{Z}_q$)
3. $x \leftarrow \mathcal{A}(\mathbb{G}, q, g, h) : x \in \mathbb{Z}_q$
4. The experiment succeeds(1) iff $g^x = h$

The Discrete Logarithm problem is said to be hard wrt a given $\mathcal{G}$ if $\forall$ PPT adversaries $\mathcal{A}$, $\exists$ a negligible function $negl()$:

$$Pr[DLog_{\mathcal{A},\mathcal{G}}(n) = 1] \leq negl(n)$$

The discrete logarithm assumption is that $\exists\mathcal{G}$ for which the discrete logarithm problem is hard.

7.3.2 Computational Diffie-Hellman Problem

For a cyclic group $\mathbb{G}$ with generator g and $h_1, h_2 \in \mathbb{G}$, define $DH_g(h_1, h_2) = g^{log_g(h_1)\cdot log_g(h_2)}$.
The CDH problem is stated as :
Given a cyclic group $\mathbb{G}$, its order q and a generator g such that the group operation is efficiently computable on $\mathbb{G}$ and two random elements $h_1, h_2 \leftarrow \mathbb{G}$, compute $DH_g(h_1, h_2)$.
If logarithms can be computed, computing DH_g is trivial and $\therefore$ CDH is not harder than DLog.

7.3.3 Decisional Diffie-Hellman Problem

The DDH problem is informally stated as :
Given a cyclic group $\mathbb{G}$, its order q and a generator g such that the group operation is efficiently computable on $\mathbb{G}$, two random elements $h_1, h_2 \leftarrow \mathbb{G}$ and a $y \in \mathbb{G}$, decide whether $y = DH_g(h_1, h_2)$ or y is a random element of $\mathbb{G}$.
The DDH problem is said to be hard wrt $\mathcal{G}$ if $\forall$ PPT adversaries $\mathcal{A}$, $\exists$ a negligible $negl()$:

$$|Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] - Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1]| \leq negl(n)$$

Where $(\mathcal{G}, q, g) \leftarrow \mathcal{G}(1^n)$ and $x, y, z \leftarrow \mathbb{Z}_q$.

Usually prime order groups are used in cryptographic application. This is done because :

1. The discrete logartithm problem in a group of order $q = q_1 q_1$ can be reduced to two DLog problems in groups of order q_1 and q_2 by *Pohlig-Hellman Algorithm*. Thus, a composite with small prime factors is unacceptable.

2. Sampling a generator from such a group is trivial and this increases the efficiency of $\mathcal{G}$

3. The DDH is believed to be very hard in such groups because $DH_g(h_1, h_2)$ is distributed close to uniform in such groups.[4]

To illustrate the last point,

THEOREM. *In a group $\mathbb{G}$ of prime order q with generator g, and $h_1, h_2 \leftarrow \mathbb{G}$,*

$$Pr[DH_g(h_1, h_2) = 1] = \frac{2}{q} - \frac{1}{q^2}$$

And for any element $y \in \mathbb{G} : y \neq 1$

$$Pr[DH_g(h_1, h_2) = y] = \frac{1}{q} - \frac{1}{q^2}$$

The quadratic residue modulo p subgroups of $\mathbb{Z}_p^*$ for a *strong* prime p are used extensively in cryptographic application because $\mathbb{Z}_p^*$ does not have a prime order and also DDH is not hard in $\mathbb{Z}_p^*$. This is done to ensure that QM_p is a prime order group and thus DDH is hard in it.
These assumptions have enormous application. For instance, they imply the existence of pseudo-random constructs such as one-way functions and permutations and collision resistant hashings.

8 Public Key Cryptography

8.1 Limitations of Private-Key Cryptography

Private key setting has 3 peculiar limitations which make its use in everyday applications difficult.

1. *Key Setup*
   In a naive private key setting involving multiple users, each pair must have a unique secret key and the keys need to be shared personally and not via the insecure network. This approach is neither feesible nor scalable.

2. *Key Storage and Security*
   In the naive setting for U users, each user has to store $U - 1$ keys. Also, there are U points of access for an attacker to steal keys from[5]. A solution to this is smartcards, all cryptographic computations are handled here. Smartcards are highly protected hardware equipments but have memory and cost limitations thus, non-scalable.

3. *Non-Applicability to Open Systems*
   Open systems (or 'ad-hoc' systems) are ones where communicating parties cannot interact physically for key-exchange (ex. e-mails, e-commerce). In such systems, there is no way of working in a private-key cryptographic setting.

[4] This is neither necessary nor sufficient for DDH to be hard and is merely a heuristic
[5] usually using computer VIRUSes

There are three well known primitives to Public-Key cryptography:

1. Public-key Encryption
2. Digital Signature Schemes
3. Interactive Key Exchange

9 Diffie-Hellman Key Exchange

A key exchange is a protocol Π followed by two parties $\mathcal{A}, \mathcal{B}$ communicating over a network. The protocol is such that the input to $\mathcal{A}$ and $\mathcal{B}$ is the security parameter 1^n and they use random coins $\nabla_{\mathcal{A}}, \nabla_{\mathcal{B}}$ in the interaction. Denote the outputs of $\mathcal{A}, \mathcal{B}$ after following the protocol by $output_{\mathcal{A},\Pi}(1^n, \nabla_{\mathcal{A}}, \nabla_{\mathcal{B}})$ and $output_{\mathcal{B},\Pi}(1^n, \nabla_{\mathcal{A}}, \nabla_{\mathcal{B}})$ and the set of all messages sent by $transcript_{\Pi}(1^n, \nabla_{\mathcal{A}}, \nabla_{\mathcal{B}})$. The output is the secret key k shared by $\mathcal{A}$ and $\mathcal{B} : ||k|| = n$.

DEFINITION. *A key exchange protocol is correct if* $\exists$ *a negligible* $negl()$:

$$Pr[output_{\mathcal{A},\Pi}(1^n, \nabla_{\mathcal{A}}, \nabla_{\mathcal{B}}) \neq output_{\mathcal{B},\Pi}(1^n, \nabla_{\mathcal{A}}, \nabla_{\mathcal{B}})] \leq negl(n)$$

Where the probability is over the randomness in choice of $\nabla_{\mathcal{A}}, \nabla_{\mathcal{B}}$.

Given a correct KE protocol, one can define it's security by the experiment $\mathbf{KE^{eav}_{Eve,\Pi}(n)}$:

1. $\nabla_{\mathcal{A}}, \nabla_{\mathcal{B}}$ of appropriate length are chosen (specified in Π)
2. *Eve* is supplied with $transcript_{\Pi}(1^n, \nabla_{\mathcal{A}}, \nabla_{\mathcal{B}})$
3. $k \leftarrow \{0,1\}^n$ and $b \leftarrow \{0,1\}$
4. If $b = 0$, $output_{\mathcal{A},\Pi}(1^n, \nabla_{\mathcal{A}}, \nabla_{\mathcal{B}})$ is supllied to *Eve*. Else, k is supplied.
5. *Eve* outputs a bit b' and experiment succeeds (1) iff $b = b'$

DEFINITION. *A key exchange protocol* Π *is said to be secure in presence of an eavesdropping adversary if* $\forall$ *PPT adversaries Eve,* $\exists$ *a negligible function* $negl()$:

$$Pr[KE^{eav}_{Eve,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Since, the key is random for an outsider, intuitively, it can be safely used in a private key setting.

An instance of such a protocol is the abstract Diffie-Hellman key exchange protocol.

CONSTRUCTION 12 (Diffie-Hellman KE Protocol)**.** $\mathcal{A}, \mathcal{B}$ *are given input* 1^n.

1. $\mathcal{A}$ *generates* $\langle \mathbb{G}, q, g \rangle \leftarrow \mathcal{G}(1^n)$ *and sends it to* $\mathcal{B}$.
2. $\mathcal{A}$ *choses* $x \leftarrow \mathbb{Z}_q$ *and sends* $h_1 = g^x$ *to* $\mathcal{B}$.
3. $\mathcal{B}$ *choses* $y \leftarrow \mathbb{Z}_q$ *and sends* $h_2 = g^y$ *to* $\mathcal{B}$
4. *output for* $\mathcal{A}$ *is* h_2^x *and for* $\mathcal{B}$ *is* h_1^y.

The correctness of the protocol is trivially verified as $h_1^y = g^x y = g^y x = h_2^x$.
For security, $KE^{eav}_{Eve,\Pi}$ needs to be modified such that $k \leftarrow \mathbb{G}$ instead of $k \leftarrow \{0,1\}^n$. Then,

THEOREM. *The Diffie-Hellman key exchange protocol is secure in the presence of an eavesdropping adversary if the decisional Diffie-Hellman problem is hard relative to $\mathcal{G}$.*

The Diffie-Hellman protocol in this form cannot be used because of two reasons :

1. Usually, keys are required to be pseudorandom in $\{0,1\}^n$. This is done by using `randomness extractors` which convert $g \in \mathbb{G}$ to pseudorandom bitstrings.
2. The protocol is not secure against *man-in-the-middle* type attacks where an attacker can modify the messages sent and add it's own messages.

A simple *man in the middle* type attack can also be used to render the naive version of the protocol incorrect. The adversary intercepts all communication and behaves as 2 different parties $\mathcal{A}', \mathcal{B}'$ for $\mathcal{A}, \mathcal{B}$ respectively such that $\mathcal{A}, \mathcal{B}$ end up with different keys k_1, k_2 both of which are known to the adversary. In this way, neither $\mathcal{A}$, nor $\mathcal{B}$ know that the connection is insecure and the adversary can decipher all the messages.

10 Public Key Encryption

I these schemes, a receiver generates a pair of keys (pk, sk), where pk is called the public key and sk is called the private key. She then makes pk available to anyone who wants to communicate with her. It is assumed that any sender and any adversary can obtain a *legitimate* copy of pk. The security of such schemes depends only on the secrecy of sk.
One instance of such a scheme facilitates communication in only one direction.

DEFINITION. *A tuple of PPT algorithms $\Pi = (Gen, Enc, Dec)$ forms a public key encryption scheme if:*

1. *$Gen(1^n)$: Outputs a pair of keys (pk, sk) : $||pk||, ||sk|| \geq n$. Also, given pk or sk, n is determined efficiently.*
2. *$Enc_{pk}(m)$: Outputs ciphertext c where $m \in \mathcal{M}_{pk}$.*
3. *$Dec_{sk}(c)$: Is a deterministic algorithm and outputs plaintext m or failure $\perp$:*

$$Dec_{sk}(Enc_{pk}(m)) = m \ \ \forall m \in \mathcal{M}_{pk}$$

10.1 EAV and CPA security

The experiment $\mathbf{PubK^{eav}_{\mathcal{A},\Pi}(n)}$ is defined as:

1. $(pk, sk) \leftarrow Gen(1^n)$
2. $(m_0, m_1) \leftarrow \mathcal{A}(pk) : m_0, m_1 \in \mathcal{M}_{pk}$ and $|m_0| = |m_1|$
3. $b \leftarrow \{0,1\}$, $c \leftarrow Enc_{pk}(m_b)$ and $b' \leftarrow \mathcal{A}(c) : b' \in \{0,1\}$ [c is called the challenge ciphertext]
4. The experiment succeeds (1) iff $b = b'$

DEFINITION. *A public key encryption scheme* Π *is secure in presence of an eavesdropper if* $\forall$ *PPT adversaries* $\mathcal{A}$, $\exists$ *a negligible function* $negl()$ *:*

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

$\because \mathcal{A}$ known pk, it automatically has access to $Enc_{pk}()$ oracle $\therefore$ an EAV secure scheme is also CPA secure. Thus, no such scheme can have deterministic encryptions and all such schemes are automatically secure for multiple messages. Formally,

THEOREM. *If a public key encryption scheme has indistinguishable encryptions in presence of an eavesdropper, it has multiple indistinguishable encryptions in presence of an eavesdropper.*

$\therefore$ public-key schemes can be naturally (even though naively) extended to arbitrary length encryptions. Usually public key encryption is 2-3 order of magnitudes slower than private key. To improve efficiency for large messages, hybrid encryptions are used.

10.2 Hybrid Encryption

The naive method of arbitrary length encryptions using public-key encryption scheme will use t instances of the scheme which is computationally extremely heavy. A way around when t is large is to first use the public-key scheme Π to convey a key k secretly to the receiver and then use a private-key scheme Π' with the key k to communicate secretly.

CONSTRUCTION 13 (Hybrid Encryption Scheme)**.** *Given a public-key encryption scheme* Π *and a private-key encryption scheme* Π'*, define a public-key encryption scheme* Π^{hyb} *as:*

1. $Gen^{hyb}(1^n)$ *:* $(pk, sk) \leftarrow Gen(1^n)$
2. $Enc^{hyb}_{pk}(m)$ *:* $k \leftarrow \{0,1\}^n$ *and* $c_1 \leftarrow Enc_{pk}(k), c2 \leftarrow Enc'_k(m)$*. Output* $\langle c_1, c_2 \rangle$
3. $Dec^{hyb}_{sk}(\langle c_1, c_2 \rangle)$ *:* $k := Dec_{sk}(c_1)$*, output* $m := Dec'_k(c_2)$

The scheme Π^{hyb} is asymtotically equal in efficiency to Π' when t is large. For Π^{hyb} to be CPA-secure, Π' only needs to have indistinguishable encryptions in presence of an eavesdropper. This is intuitive because k is chosen randomly everytime a message is sent and is secretly conveyed.

THEOREM. *The public-key encryption scheme* Π^{hyb} *is CPA secure if* Π *is CPA secure and* Π' *has indistinguishable encryptions in presence of an eavesdropper.*

Multiple public-key schemes are used in practice.

10.3 El-Gamal Encryption

Consider an algorithm $\mathcal{G}$ which on input 1^n outputs description of a cyclic group $\mathbb{G}$, its order q and a generator g. The El-Gamal scheme is defined as:

CONSTRUCTION 14 (El-Gamal Encryption Scheme)**.** *Given a* $\mathcal{G}$*, define* Π *as:*

1. $Gen(1^n)$ *:* $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n), x \leftarrow \mathbb{Z}_q$ *and output* $(\langle \mathbb{G}, q, g, g^x \rangle, \langle \mathbb{G}, q, g, x \rangle)$
2. $Enc_{\langle \mathbb{G},q,g,h \rangle}(m)$ *:* $y \leftarrow \mathbb{Z}_q$ *and output* $\langle g^y, h^y \cdot m \rangle$

3. $Dec_{\langle \mathbb{G},q,g,x\rangle}(\langle h_1, h_2\rangle)$: *output* $m := h_1^{-x} \cdot h_2$

The scheme is not instantiable till structure of $\mathbb{G}$ is known. Nevertheless,

THEOREM. *If DDH is hard relative to $\mathcal{G}$, the El-Gamal encryption scheme Π is CPA secure.*

A practical instantiation of El-Gamal encryption uses QM_p (quadratic residue) for a strong prime p. The strings of size $(n-1)$ can be naturally interpreted as elements of $\{1, 2, \ldots (p-1)/2\}$ and mapped $m \to [m^2 \ mod \ p]$. This is an invertible mapping (except with probability $negl(|p|)$).

10.4 RSA Encryption

The first (textbook) RSA was constructed as:

CONSTRUCTION 15 (textbook RSA)**.** *Given a* `GenRSA`*, define Π as:*

1. $Gen(1^n)$: $(N, e, d) \leftarrow$ `GenRSA`(1^n) *and output* $(\langle N, e\rangle, \langle N, d\rangle)$
2. $Enc_{\langle N,e\rangle}(m)$: *For* $m \in \mathbb{Z}_N^*$, *output* $[m^e \ mod \ N]$
3. $Dec_{\langle N,d\rangle}(c)$: *output* $m := [c^d \ mod \ N]$

Even though m cannot be determined efficiently by RSA assumption, the scheme is ensecure $\because$ *Enc* is deterministic. Thus, partial information about m is efficiently accessible.

The scheme serves as a base to build secure schemes. To encode binary strings as elements of $\mathbb{Z}_N^*$, all $|N|-1$ length strings are naturally interpreted[6].

The padded RSA constructed is also based on the RSA assumption and is constructed as:

CONSTRUCTION 16 (Padded RSA)**.** *Given a* `GenRSA` *and a function l : $l(n) \leq 2n-2$ determining the encryptand length , define Π as:*

1. $Gen(1^n)$: $(N, e, d) \leftarrow$ `GenRSA`(1^n) *and output* $(\langle N, e\rangle, \langle N, d\rangle)$
2. $Enc_{\langle N,e\rangle}(m)$: *For* $m \in \{0,1\}^{l(n)}$, $r \leftarrow \{0,1\}^{|N|-l(n)-1}$ *and output* $[(r||n)^e \ mod \ N]$
3. $Dec_{\langle N,d\rangle}(c)$: *output the last $l(n)$ bits of* $[c^d \ mod \ N]$

The security of the scheme depends on l. For $l(n) = 2n - \mathcal{O}(log(n))$, an attacker for $PubK^{eav}$ can use a brute force attack in $2^{\mathcal{O}(log(n))} = p(n)$ time and thus, scheme is insecure.

The security of the scheme is proven when $l(n) = 1$ and it can be extended naively though such a scheme will be very inefficient.

Single bit CPA-secure encryption using trapdoor functions can be constructed as:

CONSTRUCTION 17 (Single-Bit Trapdoor Encryption Scheme)**.** *Given a trapdoor permutataion $\widehat{\Pi} = (\widehat{Gen}, f)$ with an efficient single bit output function hc, define Π as:*

1. $Gen(1^n)$: $(I, td) \leftarrow \widehat{Gen}(1^n)$ *and output* $(I, \langle I, td\rangle)$
2. $Enc_I(m)$: *For* $m \in \{0,1\}$, $x \leftarrow \mathcal{D}_I$ *and output* $\langle f_I(x), hc_I(x) \oplus m\rangle$
3. $Dec_{\langle I,td\rangle}(\langle y, \gamma\rangle)$: $x := Inv_{td}(y)$ *and output* $m := hc_I(x) \oplus \gamma$

THEOREM. *If hc is a hard-core predicate of $\widehat{\Pi}$, the scheme Π is CPA-secure.*

The last bit of RSA family of trapdoor permutations is known to be a hard-core predicate. Thus, it can be cast into an ecryption scheme.

[6]The probability that a randomly chosen $|N|-1$ length string $\notin \mathbb{Z}_N^*$ is negligible in $|N|$. Because if such an m is efficiently determined, $gcd(m, N)$ gives p or q and thus N is factorized.

10.5 CCA security

Chosen ciphertext attacks are more of a threat in the public key setting than they are in the private key domain. They are much more feesible because usually a receiver in this setting is expecting multiple messages. For example, consider a person informing a bank about the loss of his debit card. This is a common message and if an adversary replays this message as his own to a bank, their response might give away the contents.

More sophisticated attacks may exist if a scheme is mallaeble. An can obtain an encryption of a certain $f(m)$ when encryption of m is known. It might be possible to misuse this information.

Thus, as in the public key setting, the experiment $\mathbf{PubK}^{\mathbf{CCA}}_{\mathcal{A},\Pi}(\mathbf{b})$: is defined as:

1. $(pk, sk) \leftarrow Gen(1^n)$
2. $m_0, m_1 \leftarrow \mathcal{A}^{Dec_{sk}(\cdot)}(pk) : m_0, m_1 \in \mathcal{M}_{pk}$ and $|m_0| = |m_1|$
3. $b \leftarrow \{0,1\}, c \leftarrow Enc_{pk}(m_b)$ and $b' \leftarrow \mathcal{A}^{Dec_{sk}(\cdot)}(c)$ where c cannot be queried
4. The experiment succeeds iff $b = b'$

DEFINITION (CCA-secure Encryption Scheme)**.** *An encryption scheme* Π *has indistinguishable encryptions under a chosen ciphertext attack if* $\forall$ *PPT adversaries* $\mathcal{A}$, $\exists$ *a negligible negl* :

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

None of the schemes considered till now are CCA secure. There exist easy methods to completely break them. For example, consider an El-Gamal encryption:

Say A sends the encryption of $m = (\langle c_1, c_2 \rangle)$ to B. An adversary can intercept $\langle c_1, c_2 \rangle$ and ask for a decryption of $\langle c_1, c_2 \cdot M \rangle$ for a chosen M. It is clear that the decryption will be $m \cdot M$ and m can be efficiently computed by operating on M^{-1}. A smart adversary would also chose a random $x \leftarrow \mathbb{G}$ and actually ask for decryption of $\langle xc_1, xc_2 \cdot M \rangle$ to avoid intrusion detection because the probability of chosing the same c_1 for two encryptions is very low. This choice is also an encryption of $m \cdot M$ with the advantage that xc_1 is completely unrelated to c_1.

11 Digital Signatures

These are the public-key equivalents of MAC. They function as follows :

A sender S who needs to send a message m to a large number of parties generates a public key pk and a private key sk and then uses sk to generate a signature for m. If the signature generated is σ, the transferred message is (m, σ). Assuming a receiver has an unaltered copy of pk, he can verify that σ is a valid signature for m. Any adversary who does not know sk, cannot generate a pair (m', σ') which is verified by a holder of pk to be valid except with negligible probability. This verifies that S sent m AND m wasn't modified in transit.

The domain has 3 main properties differentiating it from MACs:

1. *Easy distribution* : Only one pk needs to be known to all the receivers to verify the validity of a signature. Hence, key distribution is much more efficient than MACs.
2. *Public Verifiability* : A digital signature is verifiable by anyone who has a legitimate pk. Thus, all receivers must agree on the verifiability of a message.

3. *Non-Repudiation* : Once a party sends a message, they cannot claim that they did not. Doing so, it can be proved in a court that the sender's claim is bogus $\because$ pk is known.

Since digital signatures are a public key domain, they are much slower than MACs. Thus, whenever these properties aren't required, setting up a MAC is desirable. Formally,

DEFINITION. *A tuple of PPT algorithms* $(Gen, Sign, Vrfy)$ *is called a signature scheme:*

1. $Gen(1^n)$ *: Outputs* $(pk, sk) : ||pk|| = ||sk|| = n$ *and* n *is determined efficiently given* pk*.*
2. $Sign_{sk}(m)$ *: Generates a signature* σ*.*
3. $Vrfy_{pk}(m, \sigma)$ *: Outputs a bit* $b \in \{0, 1\}$ *where 1 represents valid and 0 invalid.*

Also, $\forall n$ *and* $\forall pk, sk$ *output by* $Gen(1^n)$*,*

$$Vrfy_{pk}(m, Sign_{sk}(m)) = 1$$

A singature σ is valid for a message m and a sender with public-key pk if $Vrfy_{pk}(m, \sigma) = 1$
It is assumed that pk is distributed effectively. Distributing a public key is a difficult process and is carried out only once, after which the process becomes efficient.
The experiment $\mathbf{SignForge}^{\mathbf{cma}}_{\mathcal{A},\Pi}(\mathbf{n})$ is defined as:

1. $(pk, sk) \leftarrow Gen(1^n)$
2. $(m, \sigma) \leftarrow \mathcal{A}^{Sign_{sk}(\cdot)}$ and all the oracle queries are stored in $\mathcal{Q}$
3. The experiment succeeds iff $m \notin \mathcal{Q}$ and $Vrfy_{pk}(m, \sigma) = 1$

DEFINITION. *A signature scheme* Π *is existentially unforgeable under an adaptive chosen message attack if* $\forall$ *PPT adversaries* $\mathcal{A}$*,* $\exists$ *a negligible negl :*

$$Pr[SigForge^{cma}_{\mathcal{A},\Pi}(n) = 1] \leq negl(n)$$

Usually, signature schemes are not defined for arbitrary length messages. Although, the message space $\mathcal{M}_{pk}$ has to be superpolynimial in size wrt n to avoid brute-force attacks. One can extend such schemes to arbitrary length schemes by using the *Hash then Sign* paradigm. Formally,

CONSTRUCTION 18 (Hash-then-Sign Signature Scheme)**.** *Given a signature scheme* $\Pi = (Gen, Sign, Vrfy)$ *with a message space* $\mathcal{M}_n$ *and a hash function* $\Pi^h = (Gen^h, H)$ *such that* $H_s : \{0, 1\}^* \rightarrow \mathcal{M}_n$ *for all* s *generated by* $Gen^h(1^n)$*, define* $\Pi' = (Gen', Sign', Vrfy')$ *as:*

1. $Gen'(1^n)$ *:* $(pk, sk) \leftarrow Gen(1^n)$ *and* $s \leftarrow Gen^h(1^n)$*. Output* $(\langle pk, s\rangle, \langle sk, s\rangle)$*.*
2. $Sign'_{\langle sk,s\rangle}(m)$ *: Output* $\sigma \leftarrow Sign_{sk}(H_s(m))$
3. $Vrfy_{\langle pk,s\rangle}(m, \sigma)$ *: Check if* $Vrfy_{pk}(H_s(m), \sigma) \stackrel{?}{=} 1$

THEOREM. *If* Π *is existentially unforgeable under an adaptive chosen message attack and* Π^h *is collision resistant,* Π' *is existentially unforgeable under adaptive chosen message attack.*

11.1 RSA Signatures

11.1.1 Textbook RSA

CONSTRUCTION 19 (textbook RSA Signature Scheme)**.** *Given a* `GenRSA`*, define :*

1. $Gen(1^n)$ *:* $\langle N, e, d\rangle \leftarrow \texttt{GenRSA}(1^n)$ *and output* $(\langle N, e\rangle, \langle N, d\rangle)$
2. $Sign_{\langle N,d\rangle}(m)$ *: For* $m \in \mathbb{Z}_N^*$*, output* $\sigma := [m^d \ mod \ N]$
3. $Vrfy_{\langle N,e\rangle}(m, \sigma)$ *: Checks if* $m \stackrel{?}{=} [\sigma^e \ mod \ N]$

The scheme is easily broken wrt definition provided as one can easily obtain a forgery by chosing $\sigma \in \mathbb{Z}_N^*$ at random and computing $m := [\sigma^e \ mod \ N]$. This method does not give total control over m. Another method can be used to get forgery for a chosen m. Choose $m_1 \leftarrow \mathbb{Z}_N^*$ and compute $m_2 = m/m_1$. Then, m_1, m, m_2 are pairwise distinct except with negligible probability and thus, given signatures σ_1, σ_2 for m_1, m_2, one can compute $\sigma = [\sigma_1 \cdot \sigma_2 \ mod \ N]$.

11.1.2 Hashed RSA

One can construct a hashed RSA by using the hash then sign paradigm. Even though the textbook RSA is not secure, it cannot be said that Hashed RSA will not be secure. A minimum requirement for hashed RSA to be secure (or any hash-then-sign signature scheme) is that the hashing used should be collision resistant. Intuitively, since collision resistance implies non-invertability of a hashing, the attacks for textbook RSA are not possible here in the naive form.

11.2 One-Time Signature Scheme

The one-time signature sheme is a weaker definition of security of a signature scheme.
The experiment $\mathbf{SignForge}^{\mathbf{1-time}}_{\mathcal{A},\mathbf{\Pi}}(\mathbf{n})$ is defined as:

1. $(pk, sk) \leftarrow Gen(1^n)$
2. $(m, \sigma) \leftarrow \mathcal{A}^{Sign_{sk}(\cdot)}$ where a single query m' is answered.
3. The experiment succeeds iff $m \neq m'$ and $Vrfy_{pk}(m, \sigma) = 1$

DEFINITION. *A signature scheme* Π *is existentially unforgeable under a single message attack if* $\forall$ *PPT adversaries* $\mathcal{A}$*,* $\exists$ *a negligible negl :*

$$Pr[SigForge^{1-time}_{\mathcal{A},\Pi}(n) = 1] \leq negl(n)$$

Lamport's one-time signature scheme uses a one-way function as:

CONSTRUCTION 20 (Lamport's Scheme)**.** *Given a one-way function* f *and a polynomial length parameter* $l(n)$ *define* Π *as:*

1. $Gen(1^n)$ *: for* $i \in \{1, 2, \ldots l\}$*,*
   - $x_{i,0}, x_{i,1} \leftarrow \{0,1\}^n$
   - $y_{i,j} := f(x_{i,j})$ *for* $j = 0, 1$

Fix $pk = [y_{i,j}]$ *and* $sk = [x_{i,j}]$ *for* $i = 1, 2, \ldots l$ *and* $j = 0, 1$

2. $Sign_{sk}(m)$ *: For* $m \in \{0,1\}^l$*, output* $\sigma = (x_{1,m_1}, x_{2,m_2}, \ldots x_{l,m_l})$

3. $Vrfy_{pk}(m, \sigma)$ *: Output 1 iff* $\forall i \in \{1, 2, \ldots l\}$*,* $f(x_{i,m_i}) = y_{i,m_i}$

THEOREM. *If* f *is a one-way function, Lamport's Scheme is existentially unforgeable under a single message attack.*

11.3 Stateful Signature Schemes

Construction of secure signature schemes is hard and so is expanding one-time schemes to them. To accomplish a construction, state of the signer is included in a scheme as:

DEFINITION. *A tuple of PPT algorithms* $(Gen, Sign, Vrfy)$ *is called a stateful signature scheme when:*

1. $Gen(1^n)$ *: Outputs* $(pk, sk, s_0) : ||pk|| \geq n$ *where* pk, sk, s_0 *represent public-key, private-key and initial state respectively and* n *is deftermined efficiently given* pk*.*

2. $Sign_{sk,s_{i-1}}(m)$ *: Generates a signature and a state* (σ, s_i)*.*

3. $Vrfy_{pk}(m, \sigma)$ *: Outputs a bit* $b \in \{0, 1\}$ *where 1 represents valid and 0 invalid.*

Also, $\forall n$ *and* $\forall pk, sk, s_0$ *output by* $Gen(1^n)$ *and* $\forall$ *sequence of messages* $\langle m_1, m_2, \ldots m_l \rangle$ *when* $(\sigma_i, s_i) \leftarrow Sign_{sk,s_{i-1}}(m_i)$ *is computed recursively for* $i \in \{1, 2, \ldots l\}$*,*

$$Vrfy_{pk}(m_l, \sigma_l) = 1$$

Such schemes are relatively easier to construct given a one-time signature shceme:

11.3.1 Chain-Based Signature

Parts of pk and sk are used to encrypt new messages using a one-time scheme. If each part is independent of the others, intuitionally, the signature scheme is unforgeable.

There are two approaches to this :

1. *l-Time Signature Schemes* : These schemes can be used to transmit maximum $l(n)$ messages. An initial $pk = \langle pk_1, pk_2, \ldots pk_l \rangle$ and $sk = \langle sk_1, sk_2, \ldots sk_l \rangle$ is created by running $Gen(1^n)$ l times and using pk_i, sk_i to transmit message i. $Vrfy$ works by checking if any pk_j gives a valid signature to wrt m. An obvious choice of state is i.

   Other than the fact that the number of messages one can send is limited, the generation and verification are inefficient because the key lengths are linear in l.

2. *On-the-fly Signature Scheme* : The public key is pk_1 and the private key is sk_1. When a message m_i needs to be signed, $(pk_{i+1}, sk_{i+1}) \leftarrow Gen(1^n)$ and $\sigma_i \leftarrow Sign_{sk_i}(m_i||pk_{i+1})$ along with $pk_i + 1$ and $\{m_j, pk_{j+1}, \sigma_j\}_{j=1}^{i-1}$ are sent as the signature. The receiver can verify that each σ_j generated is correct and thus verification takes time linear in i. The state consists of $\{m_j, pk_{j+1}, sk_{j+1}, \sigma_j\}_{j=1}^{i}$ after i messages have been sent.

   The construction is not very efficient. Also, a message m_i reveals all the messages m_j for $j \in \{1, 2, \ldots i-1\}$ which may be undesirable. A technical requirement for this scheme is that the signed message needs to be larger than the pk. This is not the case in lamport's scheme but the hash and sign paradigm can be effectively used here.

11.3.2 Tree-Based Signature

Such signature schemes are used to sign messages that are n bits long. The set of all such messages can be represented by a binary tree. Each node of this binary tree contains a $\langle pk, sk \rangle$ pair and the head is called ε, the empty string. The public and private key are $pk_\varepsilon, sk_\varepsilon$.

Constructing the whole tree is infeasible, $\therefore$ nodes are added on-the-fly. The signature consists of the whole path in the tree needed to verify the final message.

To define the scheme, the notation $m|_i$ is used representing the i bit prefix of m.

CONSTRUCTION 21 (Tree-Based Signatue Scheme)**.** *Given a one-time signature* Π*, define* $\Pi' = (Gen', Sign', Vrfy')$ *as:*

1. $Gen'(1^n)$ *: Output* $(pk_\varepsilon, sk_\varepsilon) \leftarrow Gen(1^n)$
2. $Sign'_{sk_\varepsilon}(m)$ *: For* $m \in \{0,1\}^n$ *:*
   - *for* $i = 0$ *to* $n-1$ *:*
     If $pk_{m|_i 0}, pk_{m|_i 1}, \sigma_{m|_i}$ *are not included in the state,* $(pk_{m|_i 0}, sk_{m|_i 0}) \leftarrow Gen(1^n)$ *and* $(pk_{m|_i 1}, sk_{m|_i 1}) \leftarrow Gen(1^n)$ *and* $\sigma_{m|_i} \leftarrow Sign_{sk_{m|_i}}(pk_{m|_i 0} || pk_{m|_i 1})$ *are added to state.*
   - *if* σ_m *is not in state,* $\sigma_m \leftarrow Sign_{sk_m}(m)$
   - *Output signature as* $(\sigma_m, \{\sigma_{m|_i}, pk_{m|_i 0}, pk_{m|_i 1}\}_{i=0}^{n-1})$
3. $Vrfy'_{pk_\varepsilon}(m, \sigma)$ *: Checks if*
   - $Vrfy_{pk_{m|_i}}(pk_{m|_i 0} || pk_{m|_i 1}, \sigma_{m|_i}) \stackrel{?}{=} 1 \ \forall i \in \{0, 1, \ldots n-1\}$
   - $Vrfy_{pk_m}(m, \sigma) \stackrel{?}{=} 1$

THEOREM. *The tree-based signature scheme is existentially unforgeable under adaptive cma.*

An approach to removing the state would be to store random strings to define pk_w, sk_w and σ_w. Say the whole string is r_w. One can then store a key k to a pseudorandom function F with output length $||r_w||$ and input w. Then one can generate a polynomial number of r_w on-the-fly.

The tree based approach is advantageous because the signature length and thus the verification time are linear in n. Also, pk is not updated and can be kept fixed for any number of messages sent. The signature process has surely become lengthier but the bargain pays off.

11.4 Public Key Infrastructure

A PKI deals with the secure distribution of keys. It is assumed that $\exists$ at-least a single honest party $\mathcal{A}$ whose key $pk_\mathcal{A}$ is distributed. If $pk_\mathcal{B}$ is the public key of another party $\mathcal{B}$ and $\mathcal{A}$ gets has this legitimate key, it can issue a certificate $C_{\mathcal{A}\to\mathcal{B}} \leftarrow Sign_{sk_\mathcal{A}}(\mathcal{B}\text{'s key is } pk_\mathcal{B})$.

Since $\mathcal{A}$ is trusted and its key known, $\mathcal{B}$ can use the certificate to distribute its public key by sending the message $(pk_\mathcal{B}, C_{\mathcal{A}\to\mathcal{B}})$ to any party. Thus, effectively sharing their key.

The vague points here are the existence of an honest $\mathcal{A}$, the distribution of $\mathcal{A}$'s key and the way $\mathcal{A}$ determines the legitimacy of $\mathcal{B}$'s key. These define the description of a PKI.
There are various models of PKI that handle these points differently:

1. *Single Certification Authority* : In this model, a single authority CA is fixed and assigned the job of certification. Usually pk_{CA} is distributed one-time by secure means. For example in an enterprise, every new employee on their first day may be contacted by the CA to receive the key and to share their key $pk_{\mathcal{B}}$.

2. *Multiple Certification Authorities* : The process is similar to single CA system but there are multiple CAs. Usually the distribution is done via software embedding i.e. pk_{CA} is hard-cored in the software distribution used to send the messages. Also, any user has the authority to set their priorities as to which authority they trust more.

3. *Delegation* : This is an advancement over multi CA system. Here, a CA issues two types of certificates. The first type is the same as in multi CA. The second type is a delegation certificate which entrusts the right to certify to a party. Thus, if CA delegates $\mathcal{B}$ with a key $pk_{\mathcal{B}}$, it can send its key along with the certificate to authenticate other parties.

4. *Web of Trust* : This model decentralizes the delegation model completely. Each party has the right to certify another party and a receiver determines the legitimacy of a received certificate just by its level of trust in the issuer.

Appropriate models are used according to the requirements of a network.

Usually these certificates are also stored by the certification authority along with their date of expiry and serial number. This is used when the certificate needs to be invalidated.