Building Custom Security Library- Theory and Explanations

Adrija Chakraborty
(2020201063)

CS8.401.S22.Principles of Information Security

International Institute of Information and Technology,
Hyderabad

06-03-2022

Contents

1 Abstract **4**

2 Introduction **4**
- 2.1 Perfect Secrecy 4
- 2.2 Relaxation to Perfect Secrecy 4

3 Discrete Logarithm Problem **5**
- 3.1 The multiplicative group modulo p 5
- 3.2 Cyclic groups and generators 5
- 3.3 Discrete Logarithm 5

4 One Way Functions **5**

5 Hardcore Predicates **5**

6 Pseudorandom Generators **6**
- 6.1 What is a Pseudorandom Generator? 6
- 6.2 Designing secure Encryption Scheme using PRG 6
- 6.3 From one-way function to single bit expansion of PRG 6
- 6.4 Single bit expansion to arbitrary expansion PRG 7

7 Pseudorandom Functions **7**
- 7.1 What are psuedorandom functions? 7
- 7.2 Construction of PRF from PRG 7

8 CPA- Security **8**
- 8.1 Chosen Plaintext Attack 8
- 8.2 Moving from Deterministic to Probabilistic Encryption 8
- 8.3 Designing CPA-Secure Encryption scheme 8
- 8.4 Modes of Operations 9
  - 8.4.1 CBC- Cipher Block Chaining 9
  - 8.4.2 Output Feedback Mode 9
  - 8.4.3 Randomized Counter Mode 10

9 CCA-Security **10**
- 9.1 Chosen Cipher-text Attack 10
- 9.2 Does Encryption guarantee Integrity? 10
- 9.3 Message Authentication Code (MAC) 11
  - 9.3.1 Authentication Protocol 11
  - 9.3.2 Construction of MAC using PRF 11
- 9.4 CBC-MAC construction 11
- 9.5 Designing CCA-Security Scheme 12

10 Hash functions **13**
10.1 What is hashing? 13
10.2 Collision resistance 13
10.2.1 Building a Fixed length Collision Resistance Hash Function 13
10.3 Building a Variable Length Collision Resistance hash Function . 14

11 Building MACS using Hashing (H-MAC) **15**

12 Bibliography **16**

1 Abstract

This report documents all the theories, mathematical derivations and proofs used to build our custom security library. Here we make the assumptions that Discrete Logarithm Problem (DLP) is hard, and that one-way functions and pseudorandom generators exist. Then we proceed to create a PRG from one bit expansion to any arbitrary bit expansion using the concept of hardcore bits. We then create a pseudorandom function (PRF) and use it as encryption function for various communication methods like CPA and CCA. We explore MACs to achieve CCA security We also explore hash functions and H-MACS.

2 Introduction

2.1 Perfect Secrecy

The concept of Perfect Secrecy as described by Shannon is that the adversary should not obtain any additional information about the message, due to the cipher text. His knowledge about the message should be same, before and after the adversary inspects the cipher text, assuming that it has unlimited resources to attack the cipher text.

2.2 Relaxation to Perfect Secrecy

Shannon proved that for perfect secrecy, the size and entropy of the key space should be at least as much as the size and entropy of the message space. Achieving such perfect secrecy in today's current scenario is impractical. So we apply the following two relaxations to the notion of perfect secrecy:

- Security is only preserved against efficient adversaries that run in a feasible amount of time.
- Adversary can potentially succeed with some very small probability

3 Discrete Logarithm Problem

3.1 The multiplicative group modulo p

The multiplicative group modulo p is the set of $p-1$ elements $\{1, 2, \ldots, p\text{-}1\}$ under the group operation multiplication modulo p, where p is a prime.

3.2 Cyclic groups and generators

If there exists an element $g \in G$ with order equal to |G| then we say the group is cyclic. We say the element g generates the group and that g is a generator or primitive element of the group.
Interestingly, the group $\mathbb{Z}_p^*$ is always cyclic. This means that for some $g \in \mathbb{Z}_p^*$, we have

$$\{g^1, g^2, \ldots, g^{p\text{-}1}\} = \mathbb{Z}_p^*$$

3.3 Discrete Logarithm

The discrete logarithm problem (DLP) for $\mathbb{Z}_p^*$ is

$$\text{Given } g, b \in \mathbb{Z}_p^* \text{ then find } x \text{ such that } b \equiv g^x (\text{mod} p).$$

4 One Way Functions

A one-way function is a function that is easy to compute on every input, but hard to invert given the image of a random input.

A function $f : \{0,1\}^* \to \{0,1\}^*$ is one-way if the following two conditions hold:

1. **Easy to compute:** There exists a 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 A, there exists a negligible function negl such that

$$\Pr[\text{ Invert }_{A,f}(n) = 1] \leq \text{negl}(n).$$

5 Hardcore Predicates

A hardcore predicate is the hardest bit of information about the input to obtain. A function $\text{hc} : \{0,1\}^* \to \{0,1\}$ is a hard-core predicate of a function f if

1. It can be computed in polynomial time, and

2. for every probabilistic polynomial-time algorithm A, there exists a negligible function negl such that

$$\Pr_{x \leftarrow \{0,1\}^n}[\mathcal{A}(f(x)) = \mathrm{hc}(x)] \leq \frac{1}{2} + \mathrm{negl}(n)$$

6 Pseudorandom Generators

6.1 What is a Pseudorandom Generator?

Pseudo Random Numbers (PRNs) are a type of random number created from a seed value. Pseudo Random Number Generators (PRGs), also known as Deterministic Random Number Generators, generate PRNs.

Let $l(.)$ be a polynomial and let G be a deterministic polynomial-time algorithm such that for any input $s \in \{0,1\}^n$, algorith G outputs a string of length $l(n)$. We say that G is a PRG if following two conditions hold:

1. **Expansion:** For every n it holds that $l(n) > n$.

2. **Pseudorandomness:** For all probabilistic polynomial-time distinguishers D, there exists a negligible function negl such that:

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

where r is chosen uniformly at random from $\{0,1\}^n$ and the seed s is chosen uniformly at random from $\{0,1\}^n$.

6.2 Designing secure Encryption Scheme using PRG

- Gen: on input 1^n, choose $k \leftarrow \{0,1\}^n$ uniformly at random and output it as the key.
- Enc: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^{\ell(n)}$, output the ciphertext

$$c := G(k) \oplus m.$$

- Dec: on input a key $k \in \{0,1\}^n$ and a ciphertext $c \in \{0,1\}^{\ell(n)}$, output the plaintext message

$$m := G(k) \oplus c.$$

6.3 From one-way function to single bit expansion of PRG

Let f be a one-way function and let hc be a hardcore predicate of f. Then

$$G(s) = (f(s), hc(s))$$

makes a PRG with expansion factor $l(n) = n + 1$

6.4 Single bit expansion to arbitrary expansion PRG

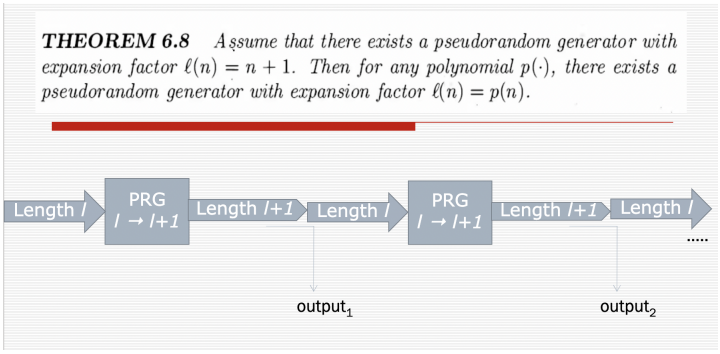

Figure 1: Single bit to Arbitrary length expansion of PRG

let us assume that a PRG with expansion factor $l(n) = n + 1$ exists. Then we do the following to create an $l(n)$ length of pseudorandom output:

- Take last bit from l+1 string for output
- Apply $l^{'}$ times to get output string of length $l^{'}$

7 Pseudorandom Functions

7.1 What are psuedorandom functions?

A psuedorandom function is a function that is indistinguishable from a random function.

Let $F : \{0,1\} \times \{0,1\} \rightarrow \{0,1\}^*$ be an efficient, length-preserving, keyed function. We say that F is a pseudorandom function if for all probabilistic polynomial-time distinguishers D, there exists a negligible function negl such that:

$$\left|\Pr\left[D^{F_k(\cdot)}\left(1^n\right) = 1\right] - \Pr\left|D^{f(\cdot)}\left(1^n\right) = 1\right]\right| \leq n\,\mathrm{negl}(n) :$$

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

7.2 Construction of PRF from PRG

Let G be a pseudorandom generator with expansion factor $\ell(n) = 2n$. Denoted by $G_0(k)$ is the first half of G' 's output, and by $G_1(k)$ the second half of G' s output. For every $k \in \{0,1\}^n$, we define the function $F_k : \{0,1\}^n \rightarrow \{0,1\}^n$ as:

$$F_k\left(x_1 x_2 \cdots x_n\right) = G_{x_n}\left(\cdots\left(G_{x_2}\left(G_{x_1}(k)\right)\right)\cdots\right).$$

8 CPA- Security

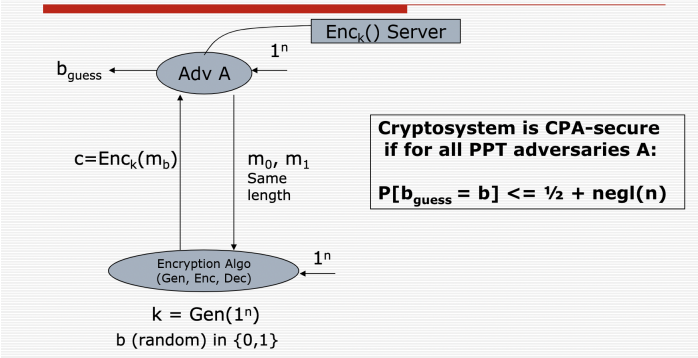

Figure 2: CPA Secure Scheme

8.1 Chosen Plaintext Attack

A chosen-plaintext attack (CPA) is an attack model where the attacker can obtain the ciphertexts for arbitrary plaintexts. The goal of the attack is to gain information that reduces the security of the encryption scheme. Since whatever security scheme we have designed is still for one-time mapping, (i.e if we encrypt same message again and again, same cipher text will be generated), to thwart a CPA attack, we need to move from the deterministic encryption to a probabilistic encryption method.

8.2 Moving from Deterministic to Probabilistic Encryption

If we want that everytime we encrypt a message, the cipher-text generated should be different, a deterministic encryption won't work. But if the cipher text keeps changing when we encrypt the same message, how will the decryption still be deterministic?
The basic idea is to not encrypt the message with probabilistic encryption but use a random noise everytime and encrypt that and xor it with message. Then send both the noise and the xor output i.e the cipher, for decryption. This way, encryption will be probabilistic but decryption will be deterministic.

$$c = (r, F_k(r) \oplus m)$$

8.3 Designing CPA-Secure Encryption scheme

Let F be a pseudorandom function. We define a private-key encryption scheme for messages of length n as follows:
- Gen: on input 1^n, choose $k \leftarrow \{0,1\}^n$ uniformly at random and output it as

the key.
- Enc: on input a key $k \in \{0,1\}^\pi$ and a message $m \in \{0,1\}^n$, choose $r + \{0,1\}^n$ uniformly at random and output the ciphertext

$$c := \langle r_1 F_k(r) \oplus m \rangle .$$

- Dec on input a key $k \in \{0,1\}^n$ and a ciphertext $c = \langle r, s \rangle$, output the plaintext message

$$m := F_k(r) \oplus s.$$

8.4 Modes of Operations

8.4.1 CBC- Cipher Block Chaining

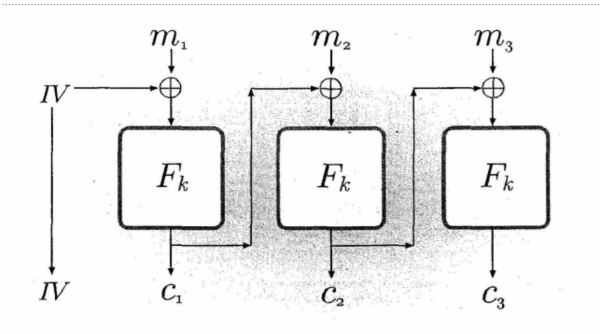

Figure 3: Cipher Block Chaining Mode

8.4.2 Output Feedback Mode

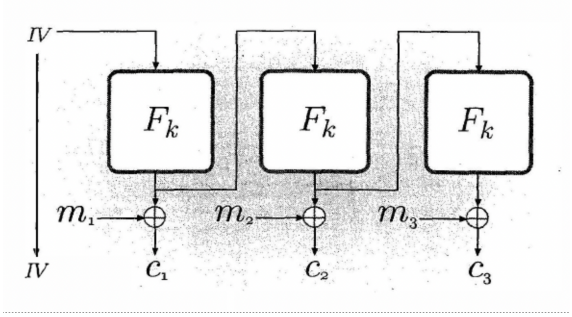

Figure 4: Output Feedback Mode

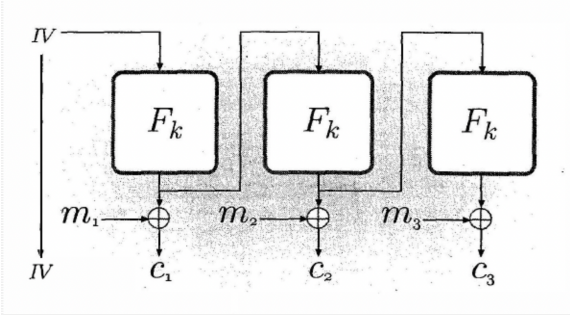

Figure 5: Randomized Counter Mode

8.4.3 Randomized Counter Mode

9 CCA-Security

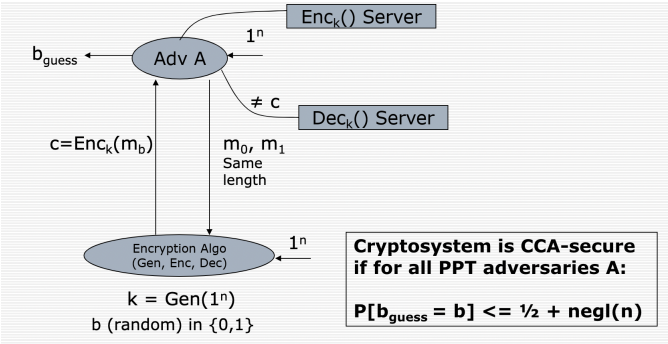

Figure 6: CCA-Security scheme

9.1 Chosen Cipher-text Attack

Here we assume that the adversary has access to both encryption server as well as decryption oracle. A chosen-ciphertext attack (CCA) is an attack model where adversary can gather information by obtaining the decryptions of chosen ciphertexts. From these pieces of information, the adversary can attempt to recover the hidden secret key used for decryption.

9.2 Does Encryption guarantee Integrity?

Encryption does not guarantee integrity. These two are different things. An attacker can modify the cipher in transit and it will hamper the integrity of the message. So we need a way to authenticate messages that are being sent.

9.3 Message Authentication Code (MAC)

9.3.1 Authentication Protocol

Since we saw encryption is not enough for data integrity, we need some authentication protocol. The components of this message authentication protocol is as follows:

- A key generation algorithm that returns a secret key k
- A Mac generating algorithm that returns a tag for a given message m. Tag $t = MAC_k(m)$
- A verification algorithm that returns a bit $b = Verify_k = (m_1, t_1)$ and a tag t_1
- If the message is not modified then with high probability, the value **b** is true, otherwise false.

9.3.2 Construction of MAC using PRF

- Gen (1^n) chooses k to be a random n-bit string
- MAC $k(m) = Fk(m) = t$ (the tag)
- Verify $k(m, t) = Accept$, if and only if $t = Fk(m)$

9.4 CBC-MAC construction

CBC-MAC is fairly similar to the original CBC mode for encryption. The Initialization Vector (IV) is a fixed value, usually zero. CBC-MAC only outputs the cipher-text's final block, which serves as the MAC.

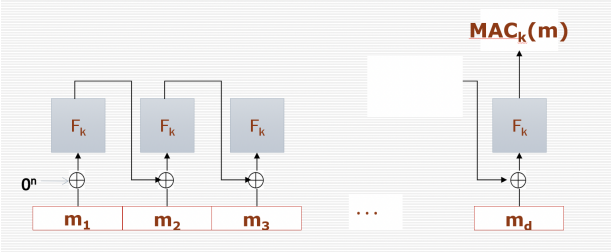

Figure 7: CBC-MAC scheme

There is a simple attack that allows us to forge new messages.

- First we get a MAC t on message m_1
- Now we do XOR the tag t into the first block of some arbitrary second message m_2, and get a MAC on the modified version of m_2.

- The resulting tag $t^{'}$ turns out to be a valid MAC for message $(m1||m2)$

The standard fix to pre-pend the message length to the first block of the message before MAC-ing it, as shown below:

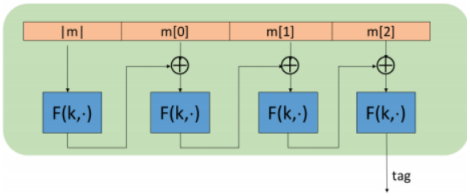

Figure 8: CBC-MAC scheme handling variable length messages

9.5 Designing CCA-Security Scheme

$$c = (r, F_{k1}(r)+m), MAC_{k2}(r, Fk1(r)+m)$$

Figure 9: CCA- encryption cipher scheme

The main crux of this encryption is to first encrypt, then authenticate. While receiving, if the authentication is not passed, we simply do not need to perform redunt decryption of some modified message. If it passes authentication, then we decrypt it.
Mathematically,

Let $\Pi_E = (Gen_E, \text{Enc}, \text{Dec})$ be a private-key encryption scheme and let $\Pi_M = (\text{Gen}_M, \text{Mac}, \text{Vrfy})$ be a message. authentication code. We define an encryption scheme (Gen', Enc', Dec') as follows:

- Gen': on input 1^n, we run $\text{Gen}\, E\,(1^n)$ and $\text{Gen}\, M\,(1^n)$ to obtain keys k_1, k_2, respectively.

- Enc': on input a key (k_1, k_2) and a plaintext message m, we compute $c \leftarrow$ Enc $k1(m)$ and $t \leftarrow \text{Mac}\, k2(c)$ and output the ciphertext $\langle c, t \rangle$

- Dec: on input a key (k_1, k_2) and a ciphertext $\langle c, t \rangle$, first we check whether $V_{rfy} k_2(c, t) \stackrel{?}{=} 1$. If yes, then output $\text{Dec}\, k1(c)$; if no, then output 1 .

10 Hash functions

10.1 What is hashing?

Hashing is essentially the process of running data through a formula that creates a hash as a result. The result is mostly of fixed length, despite the varying size of input data. A hash is designed to act as a one-way function.

10.2 Collision resistance

a hash collision is a random match in hash values that occurs when a hashing algorithm produces the same hash value for two distinct pieces of data. This is a dangerous scenario we would like to avoid, as otherwise the adversary can by-pass the authentication method.
Collisions are unavoidable, as every hash function with more inputs than outputs. Considering a hash function like SHA-256, which generates 256 bits of output from a huge input, the pigeonhole principle ensures that some inputs hash to the same result because it must generate one of 2256 outputs for each member of a much bigger set of inputs. Collision resistance does not suggest that there are no collisions; rather, it implies that they are hard to find.
LEMMA: Fix a positive integer N, and say $q \leq \sqrt{2N}$ elements $y_1, \ldots, y_9$ are chosen uniformly and independently at rondom from a set of size N. Then the probability that there exist distinct i, j with $y_i = y_j$ is at least $\frac{q(q-1)}{4N}$. That is,

$$\mathrm{col}(q, N) \geq \frac{q(q-1)}{4N}$$

PROOF:
$\Pr[\mathrm{NoCol}_q] = \Pr[\mathrm{NoColl}_1] \cdot \Pr[\mathrm{NoColl}_2 \mid \mathrm{NoColl}_1] \cdots \Pr[\mathrm{NoCol}_q \mid \mathrm{NoCol}_{q-1})$

$\Pr[\mathrm{NoColl}_1] = 1$

$\Pr[\mathrm{NoColl}_{i+1} \mid \mathrm{NoColl}_i] = 1 - \frac{i}{N}$

$\Pr[\mathrm{NoColColl}_q] = \prod_{i=1}^{q-1} \left(1 - \frac{i}{N}\right)$

For all x *with* $0 \leq x \leq 1$ *it holds that*

$$e^{-x} \leq 1 - \left(1 - \frac{1}{e}\right) \cdot x \leq 1 - \frac{x}{2}.$$

$\Pr[\mathrm{NoColl}_q] \leq \prod_{i=1}^{q-1} e^{-i/N} = e^{-\sum_{i=1}^{q-1}(i/N)} = e^{-q(q-1)/2N}$

$\Pr[\text{ Coll }] = 1 - \Pr[\mathrm{NoColl}_q] \geq 1 - e^{-q(q-1)/2N} \geq \frac{q(q-1)}{4N}$

10.2.1 Building a Fixed length Collision Resistance Hash Function

Let G be as described in the text. Define a fixed-length hash function (Gen, H) as follows:

- Gen: on input 1^n, run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$ and then select $h \leftarrow G$. Output $s := (\mathbb{G}, q, g, h)$ as the key.

- H: given a key $s = (\mathbb{G}, q, g, h)$ and input $(x_1, x_2) \in \mathbb{Z}q \times \mathbb{Z}q$, output $H^s(x_1, x_2) := g^{x_1} h^{x_2}$.

If we assume that DLP is hard, we can proof that the above method of hash construction is collision resistant, as follows:

$H^s(x_1, x_2) = H^s(x_1', x_2')$

$\Rightarrow g^{x_1} h^{x_2} = g^{x_1'} h^{x_2'}$

$\Rightarrow g^{x_1 - x_1'} = h^{x_2' - x_2}$.

$\Delta \stackrel{\text{def}}{=} x_2' - x_2$

$g^{(x_1 - x_1') \cdot \Delta^{-1}} = \left(h^{x_2' - x_2}\right)^{\left(\Delta^{-1} \bmod q\right]} = h^{\left(\Delta \cdot \Delta^{-1} \bmod q\right)} = h^1 = h$

10.3 Building a Variable Length Collision Resistance hash Function

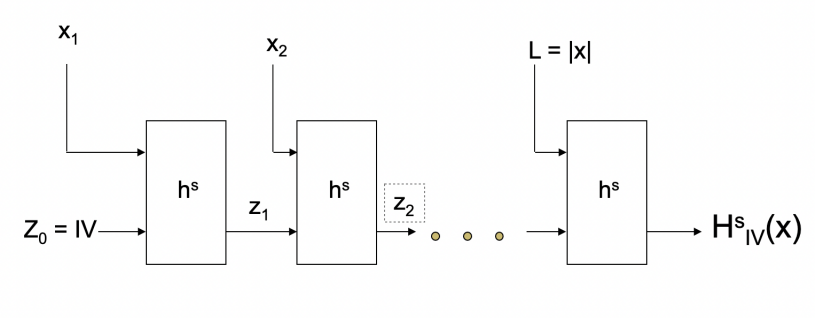

Figure 10: Merkle Damgard Transform

Let (Gen, h) be a fixed-length collision-resistant hash function for inputs of length $2\ell(n)$ and with output length $\ell(n)$. Construct a variable-length hash function (Gen, H) as follows: - Gen: remains unchanged. - H : on input a key s and a string $x \in \{0,1\}^*$ of length $L < 2^{\ell(n)}$, do the following (set $\ell = \ell(n)$ in what follows):

1. Set $B := \lceil \frac{L}{e} \rceil$ (i.e., the number of blocks in x). Pad x with zeroes so its length is a multiple of ℓ. Parse the padded result as the sequence of ℓ-bit blocks $x_1, \ldots, x_B$. Set $x_{B+1} := L$, where L is encoded using exactly ℓ bits.

2. Set $z_0 := 0^R$.

3. For $i = 1, \ldots, B+1$, compute $z_i := h^s(z_{i-1} \| x_i)$.

4. Output z_{B+1}.

This is known as the Merkle-Damgard Transform

11 Building MACS using Hashing (H-MAC)

The working of HMAC starts with taking a message M containing blocks of length b bits. An input signature is padded to the left of the message and the whole is given as input to a hash function which gives us a temporary message-digest MD'. MD' again is appended to an output signature and the whole is applied a hash function again, the result is our final message digest MD. The design is as follows:

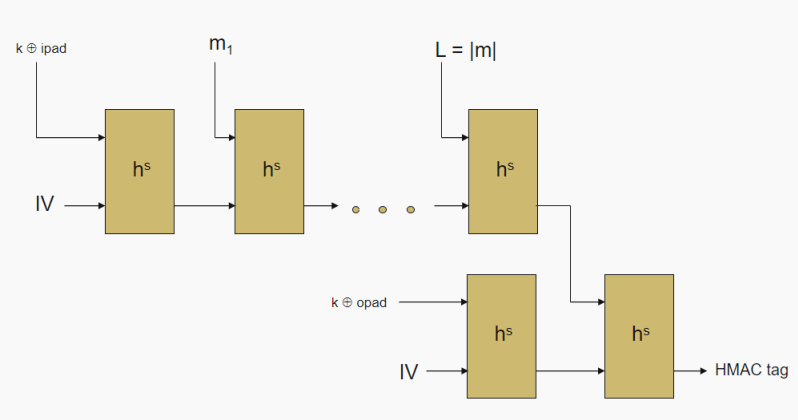

Figure 11: H-MAC scheme

- (Gen,h): A fixed length hash function
- (Gen, H): Hash function after applying MD transform to (Gen,h)
- Fixed constants- IV (initial vector), ipad(input pad) and opad(output pad)
- ipad: 0x5c repeated as many times as needed : 0x36 repeated as many times as needed

HMAC TAG for m =

$$H^s_{IV}((k \oplus opad)||H^s_{IV}(k \oplus ipad)||m)$$

12 Bibliography

References

[1] Jonathan Katz and Yehuda Lindell(2017), *Introduction to Modern Cryptography*
retrieved from: http://staff.ustc.edu.cn/~mfy/moderncrypto/reading%20materials/Introduction_to_Modern_Cryptography.pdf

[2] [MOOC]: Prof. Vinod Vaikuntanathan, Spring 2018 Cryptography Cryptanalysis
retrieved from https://youtu.be/fdr6RKyjhEs

[3] [MOOC]: Prof. Vinod Vaikuntanathan, Spring 2018 Cryptography Cryptanalysis
retrieved from https://www.youtube.com/watch?v=H008GInK0xc

[4] Class Lectures and Slides