EXPLOITATION OF HASH FUNCTIONS

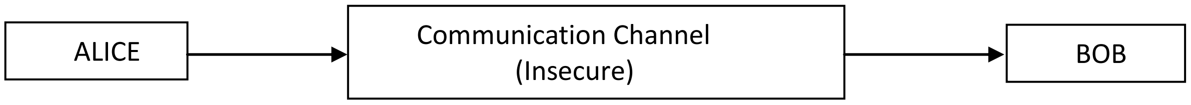
(Playing with hash functions)

Work Hard but with Smartness

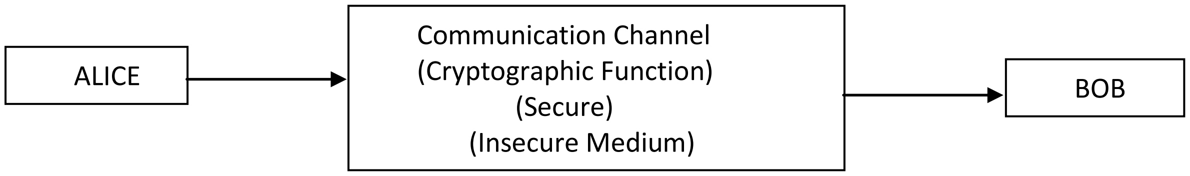
HACKING IS NOT A GAME
To Catch the Thief. THINK Like a Thief.
If you know how to HACK. You must Know how to PROTECT.

By Deepanshu Khanna:
(Linux Security Researcher)

<u>Introduction to Cryptography:</u> The term cryptography is being defined as the method or protocol for developing the security of the information over an insecure channel on which the two parties are being communicated. For an instance let's assume the two parties which are very famous in the field of cryptography that is ALICE and BOB. Let's assume Alice wants to communicate or share some information with Bob. The main problem arises here is how to share the data over such an insecure channel. So, Cryptography comes to the rescue. Hence, cryptography provides us the way to securely (neq 100%) communicates even on the insecure channel.



Fig(i)

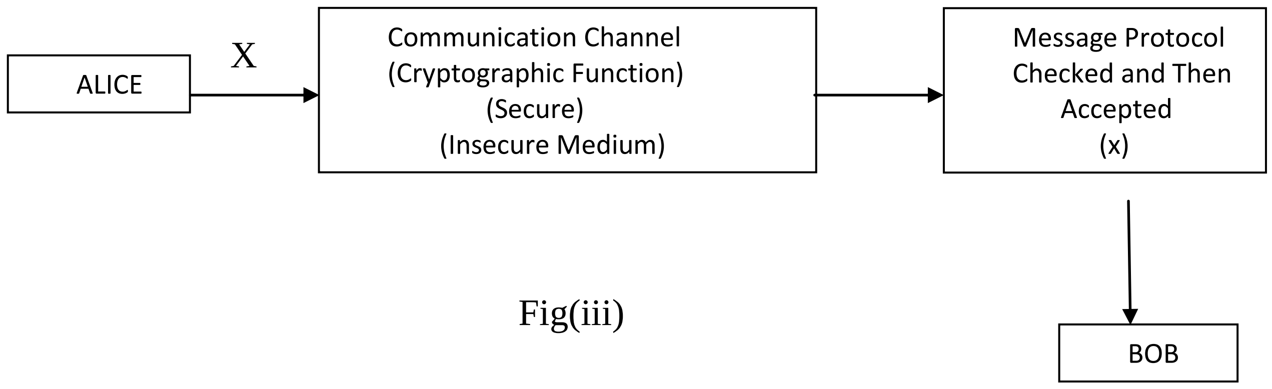


Fig(ii)

So, here is the main question arises is that what exactly this cryptographic magic function includes, which sends the information securely over such an insecure channel. So, let's look inside of this function. But before looking into it, let's try to build some building blocks for the need of these functions.

<u>Building up of the Cryptographic Functions:</u> Before digging much into the depths of cryptography, let's first focus on the building blocks of the cryptographic functions. These cryptographic functions include such properties which will help both the sender and the receiver to share the data on the basis of the trust. So, let's start building the function on the basis of "**Problem of Trust**".

Problem of Trust: Let's assume Alice and Bob wants to communicate over a telephone or fax or by e-mails. But the problem with this communication is that Bob doesn't trust Alice. So, whatever the data that is being sent or whatever the communication is being made by Alice will not be trusted by Bob. So, the solution to this problem of trust is they have to agree on some common "**protocol**". This protocol will be known to both Alice and Bob. Based on this protocol they will share their data over an insecure channel. Now in order to build such a protocol, they require a function. This particular function is known as the magic function which is required to build up that protocol and is also known as the cryptographic function. In whole cryptography we are always trying to build this function. This function is being represented by F(x)., where x is a message.



Fig(iii)

Properties of the magic faction: This magic function F(x) holds some important properties, based on

Only such properties the protocols are build up:

1. Whatever the domain and range chosen up for the function, should be integer values.
2. The function should be one-way function, which means a function F(x) should be easy to compute but hard to invert. Let's take an example, given for some value of x= 2, and a function F(x) is given by,

   F(x)= x pow (2) + x;   [given x=2]
   F(x)= 2 pow (2) + 2;
   F(x)= 4+2;
   F(x)=6

Now as you look in the above problem, that it's easy to compute F(x) from the given value of x, but it's hard to compute x from a given F(x). Say, for a given value of F(x)=6, its hard to find that could be the function. Hence, this property should the magic function holds.

3. Another big property that a magic function must hold is the property of the "**Collision Resistant".** Collision Resistant states that it is very hard or almost impossible to find a pair of distinct values (integers), for which f(x)=f(y). Means, for the given values of x and y say 2 and 3 resp., it is hard to build such two functions f(x) and f(y) such that they both holds the same values and vice-versa. This every function must hold.

Hence, in order to build such **protocols**, these properties hold a very important place. These properties are also known as the building blocks of the functions.

Mathematical formulations of building and computing F(x): Let's try to analyze the mathematical formulations of calculating the function. For that, let's assume that Alice and Bob are communicating over the telephone and deciding to have lunch together. Alice wants to go for Italian food and Bob wants to go for the Chinese. Now the situation is both the parties are sitting very far or apart from each other. And as obvious Bob doesn't trust Alice. So, they decided to build a protocol in which both of them will get satisfied. In order to do that they decided to flip a coin and if "**head**" comes Alice will win, or if "**tail**" Bob will win. Now Alice if flipping the coin, let's build the mathematical formulations for making the function F easy:

1. Assume the function F(x) = the coin Tossed or Flipped.
2. If Head comes, x = even and Alice wins
3. If Tail comes, x= odd and Bob wins.
4. Now Alice will toss the coin and send the value to Bob. This value is the function F(x) value and not the actual x value.
5. Now Bob will receive the information (the calculated F(x) value).
6. Now Bob will calculate his own value F(y), and tell Alice that this value is whether odd or even.
7. Depending upon that Alice will send x to Bob and Bob will check whether the value matches with his value or not by computing F(x).

Security Analysis of the above Statement:

1. **Can Alice cheat?**

For that Alice has to calculate the values which are collision resistant and the also holds the one-wayness property. Means Alice has to chose two randomly large numbers i.e. x and y, such that x is not equal to y and their calculated function F(x) = F(y), which is very hard to develop. So, the probability of Alice will cheat is very much less, almost NIL.

2. **Also, Can Bob guess better than Alice's random guess number??**
   Now Bob listens to F(x), and because of one-wayness property it doesn't reveal anything about x. So, his probability of choosing a random guess = ½.

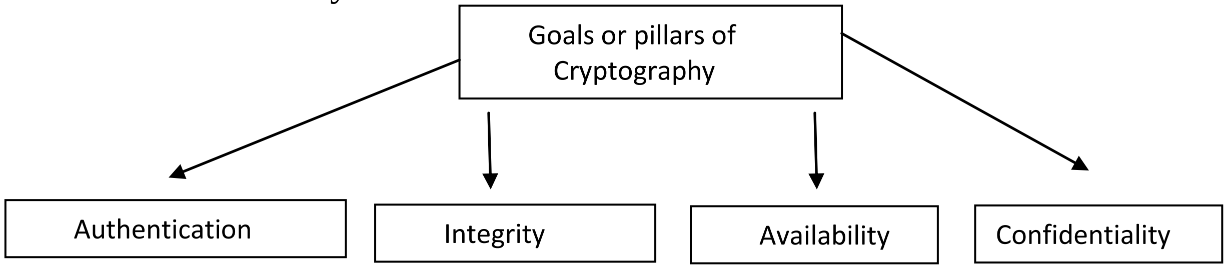
   Hence, Probability[x is even] = Probability [x is odd] = ½.

And therefore, in whole cryptography we actually are trying to focus on creating this function and holding the above three very important properties.

<u>Practical Efficiency:</u> A mathematical problem is practical efficient, if the problem is being solved in the given time and space, which is being measured by the small degree of the polynomial in the size of the problem. The polynomial that actually describes the cost of the resources to the user should be very small.

<u>Goals of Cryptography:</u> There are four main Goals of cryptography or it can also be said that cryptography is actually being balanced on the four pillars that are very necessary in order to build any cryptographic formulations and are being described as:

1. Authentication
2. Integrity
3. Availability
4. Confidentiality



Fig(iv)

1. **Confidentiality:** The term confidentiality can be defined in context to security that whatever the data that is being sent over an insecure channel should be hidden from the unauthorized access. Means the data is not easily visible to the attackers, sitting in between to attack on the data.
2. **Integrity:** The term integrity can be defined as the term that whatever the data that is being shared should not be tempered in between. Means whatever the data that is being send by the sender should receive the same on the other side of the channel to the receiver.
3. **Authentication:** Whatever the data that is being sent over the insecure channel should be sent by an authenticated user. Means the person should not be a malicious person. The authenticity of the person should be 100%, else the data will be discarded by the receiver.
4. **Availability:** the term availability could be defined as the term that whatever the resources are there should be easily available to users. Means it should not be like this that whatever the cryptographic algorithm is there, it should be only meant for the private users and not for the public domain. The algorithm must be in public domain and is 24*7 hours available to users.

Types of Cryptographic Algorithms:

1. **Symmetric key cryptography:** The algorithm which uses the same key for encryption and decryption. For example, Alice wants to send some data to Bob, but she wants data to be secured enough, that it should not get compromised. Now whatever the algorithm is being created is in the public domain, but the main responsibility is of the key (which is kept secret) to make the data secure. In this particular cryptographic algorithm, both the sender (Alice) and receiver (Bob) agreed upon a key, and using that key Alice will first encrypt the data whatever the data she wants to send. After encryption the data is being converted into the cipher text and that particular cipher text is being sent over an insecure channel. After receiving the cipher text by Bob, he will apply the decryption algorithm using the same key and get hold of the plain-text.

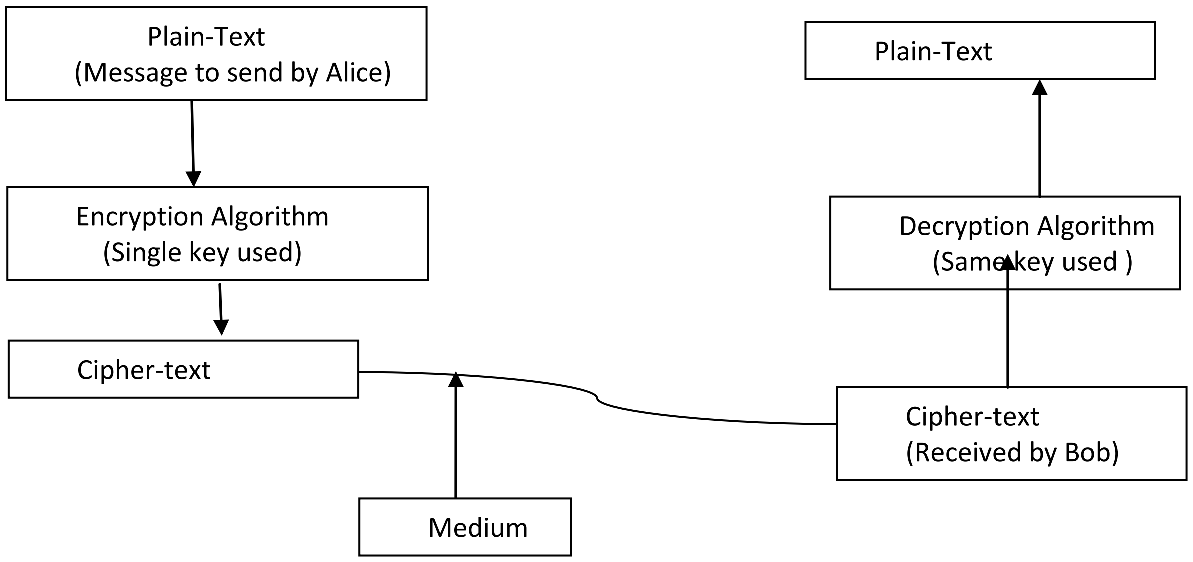
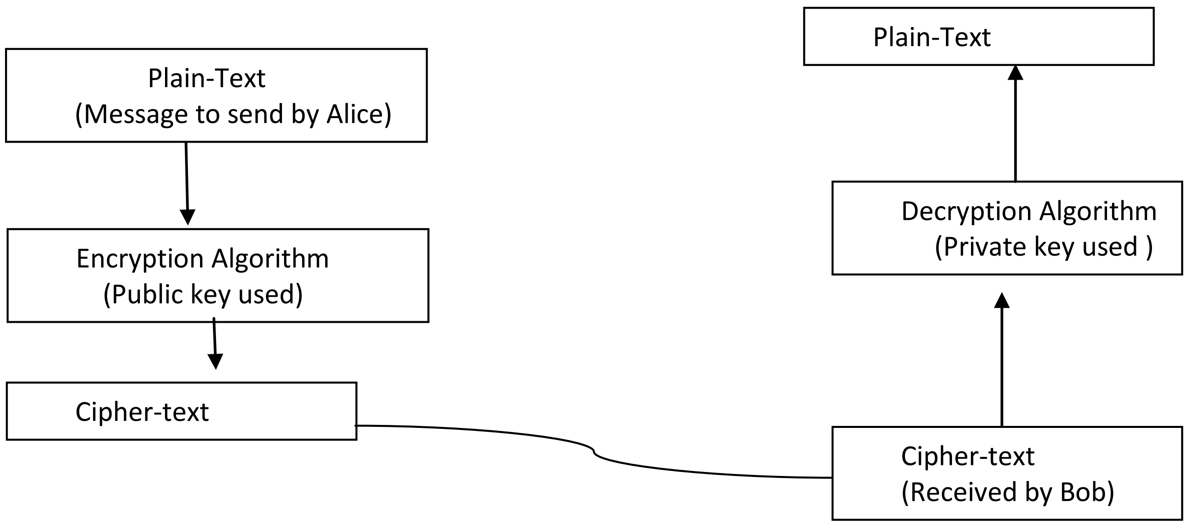


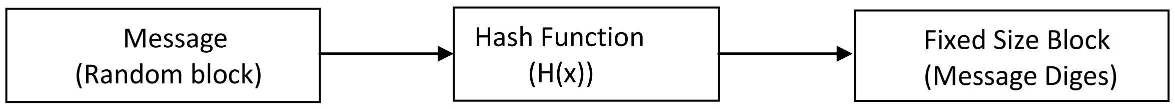
Fig (v)

2. **Public Key Cryptography:** In this particular cryptographic algorithm, except for using one keys, there will two keys that will be used for encryption and decryption. The two keys that will be used are, one is private key which is kept secret to sender or the receiver and other is the public key which is available in the public domain. Now the encryption process will go like this, the sender will take the receiver's public key from the public key directory as it is available to all or open in a public domain and will use that key to encrypt the data and produces the cipher-text. This particular cipher text will be shared on the insecure channel and will be received by the receiver on the other side. The receiver will use his private key which is kept secret to decrypt the cipher-text to obtain back the plain-text. Hence, in this cipher technique, there is a key pair which is used to encrypt the data and decrypt the cipher text. Therefore, these type of cipher techniques are much more useful and hard to break down. But there are still some mechanisms which are available to break down of such ciphers.



Fig(vi)

3. **<u>Hash Functions:</u>** A cryptographic hash function or simply the hash function is a kind of function that actually takes an random block of data, and returns back the fixed size bit string. It is often represented by H(x), where is a random size block message.



Fig(vii)

<u>Note:</u> It is important to note here that any change in x means the data accidently or intentionally will also lead to change the whole hash value with a very high probability.

The data after encoded is called the message and the hash value that is being produced of fixed size is called the message digest or simply the digest.

There are four main properties of the cryptographic hash functions:

1. It is easy to compute a hash value for a given message.

2. It is almost impossible or infeasible to generate a message for a given hash value.
3. It is impossible or infeasible to modify a message without changing the hash value.
4. It is impossible to find the two different messages with same hash value.

But these properties are good for or at one point of understanding because all these properties are being exploited and how to exploit these properties we will look in this paper, which is also the motive of the paper.

WHY **WE NEED A HASH FUNCTION??**
The answer to this question is that, the cryptographic encryptions except for one time pad are being exploited from time to time. Like DES or AES are vulnerable to linear and differential cryptanalysis attacks, which also exposes the vulnerability to the Boomerang attack. So, in order to provide the **data integrity** the cryptographic hash functions are being used. Means whatever the hash which is being produced actually creates a fingerprint on the data, which then is referred to as the message digest.

Applications of the hash functions:
1. Provides the data integrity.
2. Used in digital signatures.
3. Used for MACs (Message Authentication Codes)

1. **Digital Signatures:** These are the mechanisms by which a user is being authenticated. Means whatever the data that is being arrived over the channel is actually coming from the legitimate user or the attacker who was sitting in between and tempered the actual data with his own data. This is actually done by using the hashing.
2. **Message Authentication codes:** MACs or the Message Authentication Codes are the keyed hash functions which are used to verify the integrity of the data and the authenticity of the sender. For example, Alice wants to send some message, but the integrity of the message is very much important. So, she will append the message with the calculated hash value as [M||Hk(M)], where Hk is the hash function using some key value. Now Bob will collect the message pair, and verifies it. If the hash value is same he will accept the pair, if not he will simply discard the pair. So, this will actually prevent the adversary from tempering the message (i.e. integrity) and forcing Bob to believe that it actually came from right source i.e. from Alice (authentication).

Construction of Hash functions: All the hash which is there in the public domain or known to everyone is based upon the iterative hash function which are actually vulnerable to collision resistant attacks. So, this vulnerability is actually removed by Markel Damgard and he named this construction of hash functions as the **MARKEL DAMGARD CONSTRUCTION.** So, in order to discuss the hash functions, it's better to look at their constructions first.

Markel Damgard Construction: Markel Damgard construction is actually based on the iterative hash function methodology just leaving the fact that it uses the compression function which produces a hash which is collision resistant.

Points to Ponder about Markel Damgard Construction:

1. Uses an iterative hash function methodology

2. Compression function used as:
   Compress (C) : {0,1} pow (m+t) → {0,1} pow (m), where compress (C) is a compression function, m is the actual required bits, t is the padded bits, {0,1} are the bit representation of the actual message x.

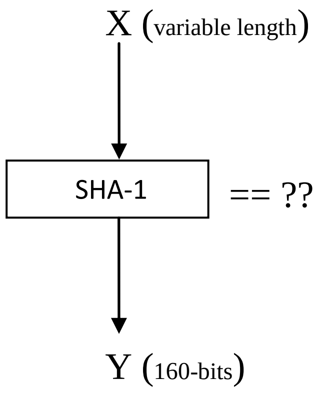
3. This is actually used to construct the hash functions which are also free from the collision resistant attacks.
   H: {0,1} pow (*) --> {0,1} pow (m), * represents the iterations.

This construction yields the proof of the above results because it gives the very wonderful results for whatever the query being asked.

Steps for construction in developing the SHA-1 or any other series of SHA family of hashes using this Markel Damgard Construction:

1. Let's consider the message "x" which is quite large, and using that it's hash value has to be constructed.

<u>Outlook of SHA-1</u>:



Fig(viii)

What's happening is that, a user is imputing the text of the variable length means it could of any length. It's going inside the SHA-1 algorithm some operations happened and it's giving the output of 160-bits as Y.

But how this is happening this? What's inside this? How we are getting this output of very short range? What could be possibly inside this algorithm?
All answers are inside this SHA-1 algorithm. As important, if we carefully look and think that whatever the data is going inside this SHA-1 function is getting compressed, so there will surely a compression function working inside SHA-1.

<u>Inside SHA-1 function Using Marel Damgard Construction:</u> So, this happens like this the user will simply enter the text and the text is simply being divided into the equal blocks of length 512 bits (naming x1, x2 ,x3……..xn), and at the end if block length < 512 bits then it will pad with some bogus bits (padding will be shown later). Then this data will go inside the compression function, and then some compression techniques will go on in between, then it will give the very first hash value for x1with an IV vector, and the hash is represented by z1, then this z1 value go back into the compression function, which becomes the IV vector for x2 and gets concatenated with the next value for x, and produce the next hash value as shown below in the figure. This process will continue for zi values, where i=n, and when whole the data is processed for 'i' iterations, then zi=H(x), where H(x) is the hash of "x". The whole process is shown below.

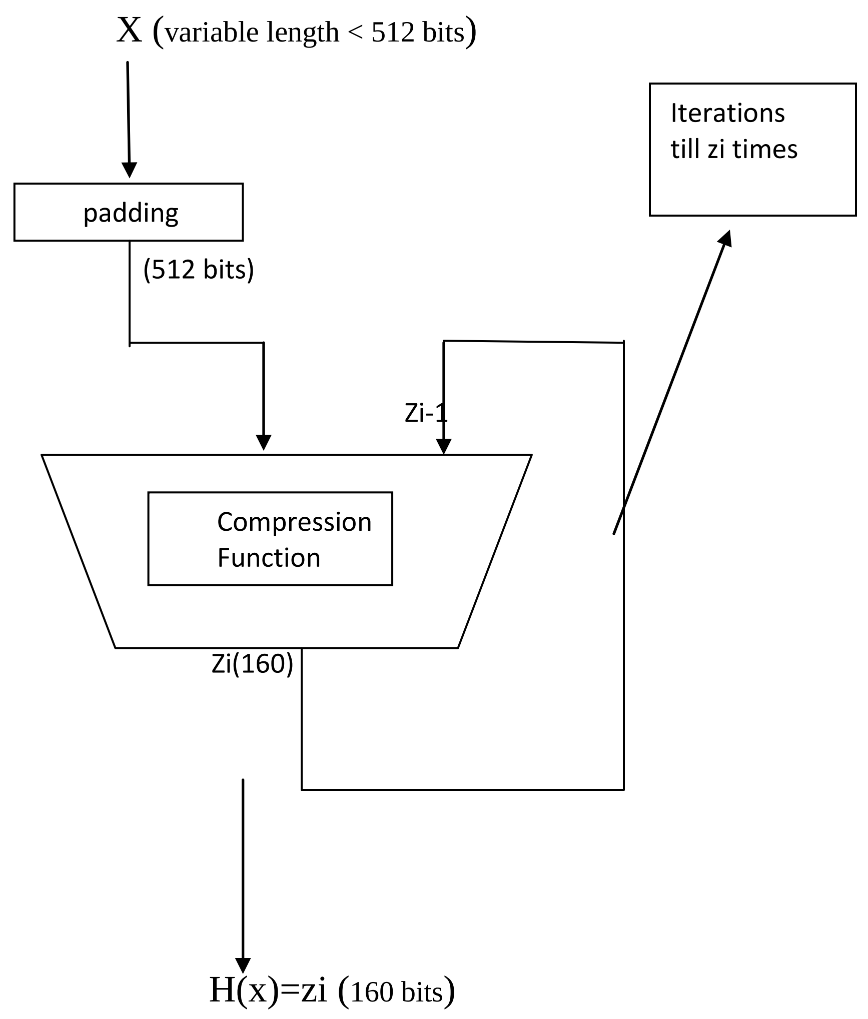
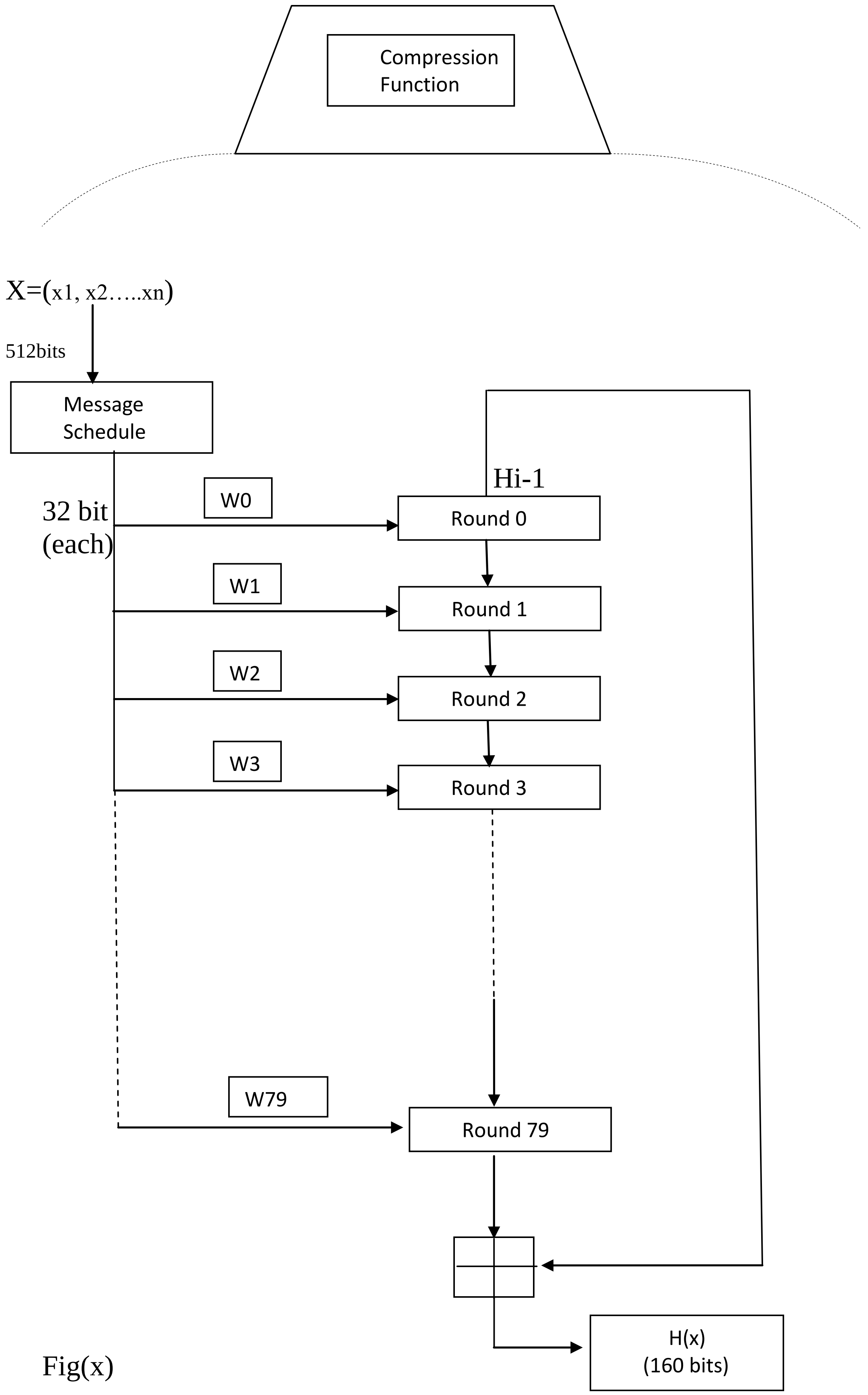


Fig (ix)
(Markel Damgard Construction for SHA-1 function)

So, till now we have seen that the data is being going inside the compression function, and some magic is going on and we got the result. So, what's this magic function?

<u>Magic inside the Compression function:</u> Compression function is a kind of a pool which takes on a variable length data or the fixed length data depending upon the algorithm and performs certain operations and gives back the compressed fixed length of data. This compression function is called the heart of the algorithm because the whole algorithm is based on this particular magic function. So, let's see what the magic that was happening inside this function is.


Compression Function
X=(x1, x2…..xn)
512bits
Message Schedule
32 bit (each)
Hi-1
W0
Round 0
W1
Round 1
W2
Round 2
W3
Round 3
W79
Round 79
H(x) (160 bits)

Fig(x)

So, we are exactly in the middle of the heart of the algorithm. So, let’s try to elaborate it’s working but before the main part, it’s important to discuss certain other factors like padding.

<u>Padding (P):</u> Padding is defined as the term of adding bogus bits to the actual message bits in order to make its length equal to the required length. So, how the process of padding goes on is as shown in the below figure.

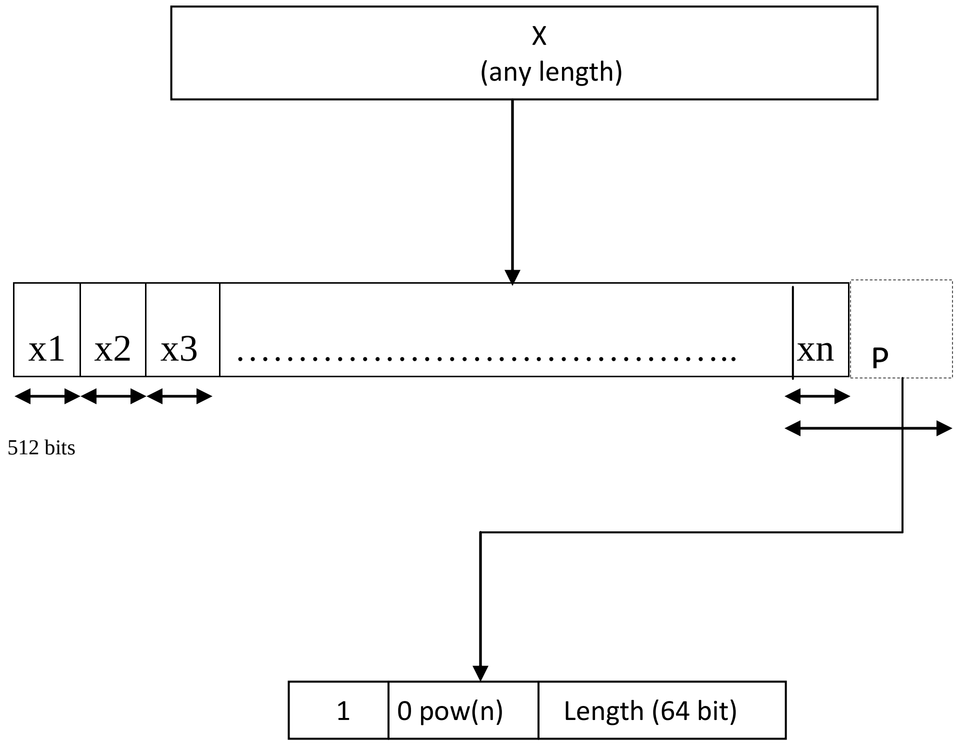


Fig (xi)

The process of padding as shown in figure, is like this we are given with the message x, and this message x is broken down into equal blocks of 512 bits, and for x1, x2,x3……..,xn values. But in the end if xn = 512 bits, then there is no need of padding, else if xn<512 bits, then there is a requirement of padding. Rather than defining with the values, it’s easy to define the values by taking an example.
<u>For example</u>: given, xn=336, xn<512 padding is required.

No. of bogus bits required= 512 – 336 = 176 bits

Now if you will look above the figure, in the end there is written Length (L), which is nothing but the 64 bit representation of the original message.

This means,

64 is fixed and hence,
No. of bogus bits required (in actual) = 176 – 64 = 112 bits.

Hence, the total number of bits to pad required is 112 bits.

In order to do that,
The value “1” is fixed and there will be 0 pow (111) be there. Means there will be 111 0’s in padded bits.

This is how, the padding works on.

Now, as we have seen how the padding process goes let’s get back to our original discussion that what or how this is working. In the compression function there are exactly 80 rounds that are working on from Round 0 to Round 79. These rounds, will take on the previous calculated hash value and gets concatenated with the message schedule, which divides the words into 32 bit value for each round and when the whole iterations gets completed, it will return back to the final calculated hash value which will be equal to 160 bits. And this is represented by H(x).

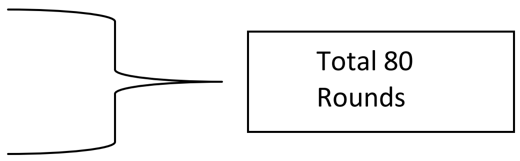
<u>Left off Things:</u> As I said, that we are exactly in the inside of the heart of the algorithm. But I don’t like to finish it here, because there is no meaning of not going into depth. Hackers are not those who work hard, but work hard with smartness requires the in depth knowledge of each and every concept. So, getting ahead of something and let’s look deeper inside it. We are left off with only two things and those are the following listed below:

1. What’s inside the Rounds?
2. What the hell is this Message Schedule?

I) <u>Looking into ROUNDS:</u>

The rounds that are shown above are not that much simpler as they look like. There is much more inside it. So, these rounds are a pack of total 4 stages and each stage contains a total of 20 rounds. And stages are being represented as ti, where i=1 – 4

t1 = Round 0 – Round 19,
t2 = Round 20- Round 39
t3 = Round 40- Round 59
t4 = Round 60 – Round 79



NOTE: I am going to discuss only 1 round here, and rest of the rounds working is same.

Working of Round 0: In Round 0 (considering all rounds), there are 5 (A,B,C,D,E) words of total length 160 bits. Means a single word is of 32 bits (simple mathematics).
5*32 = 160 bits.
And with this there is a word input from above, means on round 0 the word enters will be W0. The whole working of a single round is shown below:

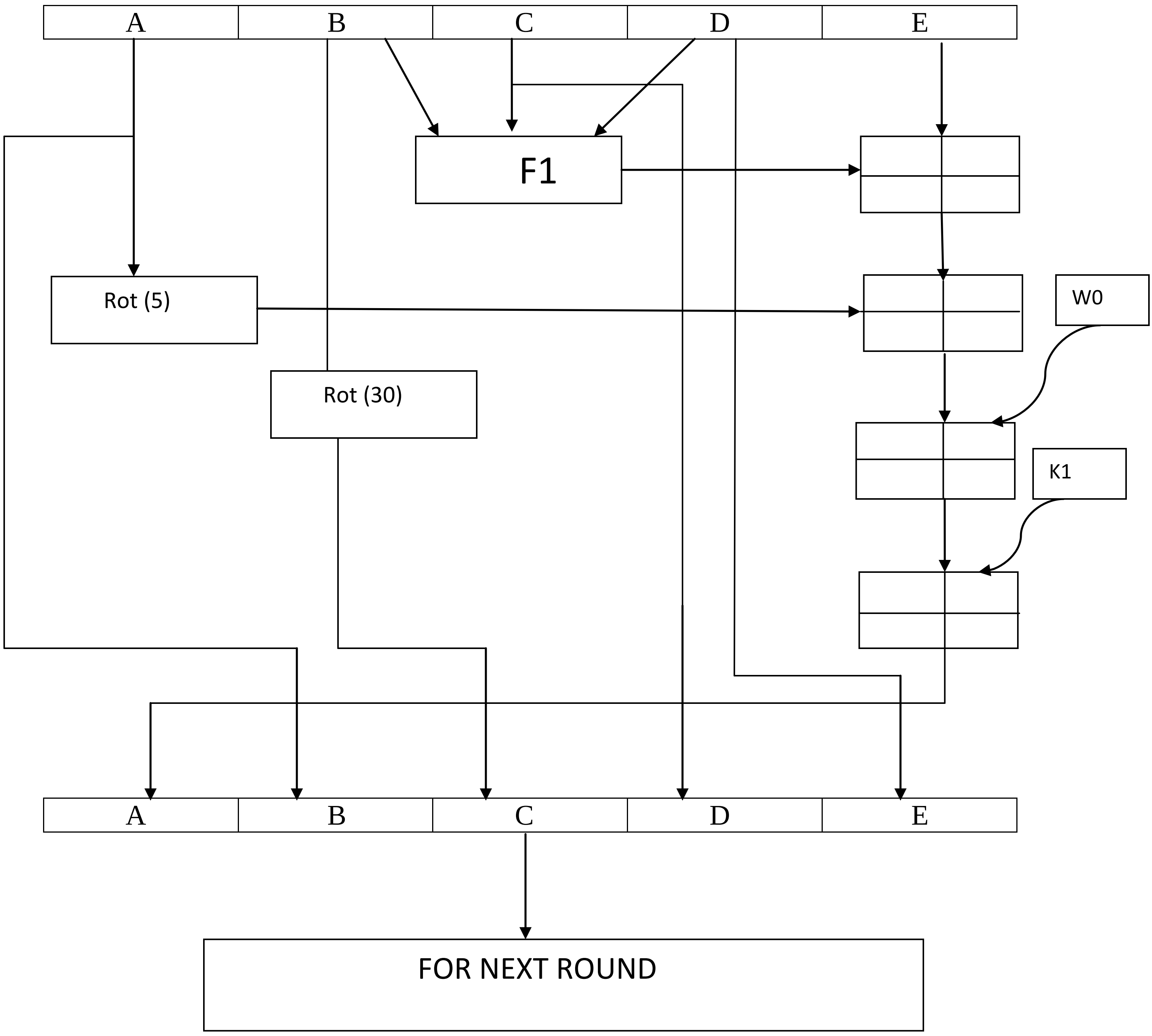


Fig (xii)
Round Working of SHA-1

Here, E is known to be an Encryption word, because whatever the values we are adding onto the addition function are first gets concatenated with the E value.

Some Points to Ponder:

1. There are total of 4 stages t1,t2,t3,t4
2. And with that there will be 4 functions (f1,f2,f3,f4) and each function will be used for its own stage time.
3. With that there will be 4 round constants (k1,k2,k3,k4) and each for each stage.

   What this means is take for **function f:**
   f= function (f1,f2,f3,f4)
   f1= function (f1) = first 20 rounds = first stage i.e. t1
   f2= function (f2) = next 20 rounds = second stage i.e. t2
   f3= function (f3) = next 20 rounds = third stage i.e. t3
   f4= function (f4) = last 20 rounds = last stage i.e. t4

   Similarly for **Round Constant k**:
   k1= Constant (k1) = first 20 rounds = first stage i.e. t1
   k2= Constant (k2) = next 20 rounds = second stage i.e. t2
   k3= Constant (k3) = next 20 rounds = third stage i.e. t3
   k4= Constant (k4) = last 20 rounds = last stage i.e. t4

There are actually some operations that are ongoing on these functions:
<u>Operations performed in functions:</u>
f1= function (f1) = function (B,C,D) = {(B) *AND* (C)} *OR* {(*not* B) *AND* (D)}
f2= function (f2) = function (B,C,D) = (B) *XOR* (C) *XOR* (D)
f3= function (f3) = function (B,C,D) = {(B) *AND* (C)} *OR* {(B) *AND* (D)}*OR* {(C) *AND* (D)}
f4 = function (f4) = function (B,C,D) = (B) *XOR* (C) *XOR* (D)

Here AND, OR nd XOR are the bit-wise operations
<u>Round Constants Value:</u>

$$\left.\begin{array}{l}\text{k1= 5A827999}\\ \text{k2 = 6ED9EBA1}\\ \text{k3 = 8F1BBCDC}\\ \text{k4 = CA62C1D6}\end{array}\right\} \boxed{\text{32-bit hexa-decimal value}}$$

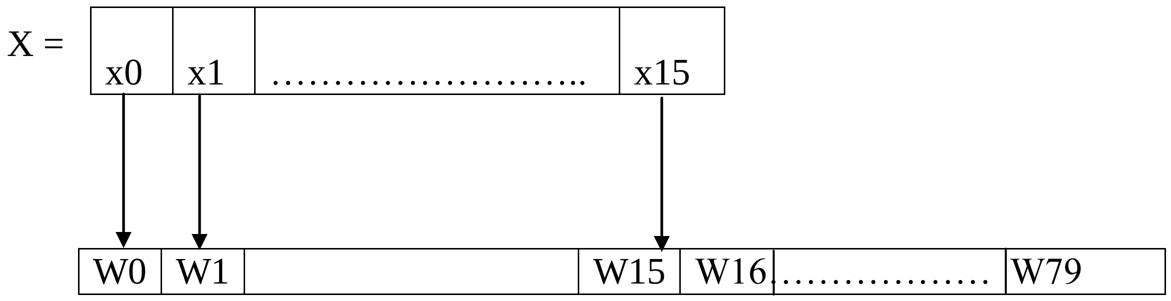
Rotation Functions (ROT()):

1. There are two rotation functions Rot (A) and Rot (B).
2. These are nothing but left-cyclic rotations.
3. A is rotated 5 times and gets concatenated with addition function.
4. B is rotated and becomes the C.

So, we are done with the working with the rounds. I don't know but it's been scary getting deeper and deeper inside someone's heart. But becoming a Hacker is not an easy task. So, we have to look deeper inside. And hence we are only left with the message scheduling algorithm.

Message Scheduling Algorithm:

Given X is of 512 bits each, means there will be equal of 16 blocks and each of size 32 bits.
16*32 = 512 bits



So the process goes like this, whatever the data is given divide that into 32 bits of equal block and the total number of blocks will be created is 16 (16*32=512 bits). Now simply copies the values of x1, x2………..,x15 below to 80 blocks of data and name them as W0, W1, W2…………..,W79. If we look carefully that till W15 all values will be copied as it is. But what about W16, W17………..W79. So, researchers have also found a method of doing that by giving a below written formula:
In general form:
W(j) = (W(j)-16) *XOR* (W(j)-14) *XOR* (W(j)-7) *XOR* (W(j)-2)

So, for example if we have to calculate the value for W(16),

W(16)= W0 *XOR* W2 *XOR* W9 *XOR* W14

So, that’s all we are done getting into the dangerous part of hashing. Means the construction of hash functions. I find it quite difficult while reading so I tried to make the things simpler and simpler so that even a newbie in this field can understand the things better off.

Now the mail interesting part yes means the exploitation of the hash functions. So, let’s get started for what we are trying to understand the things for so long.

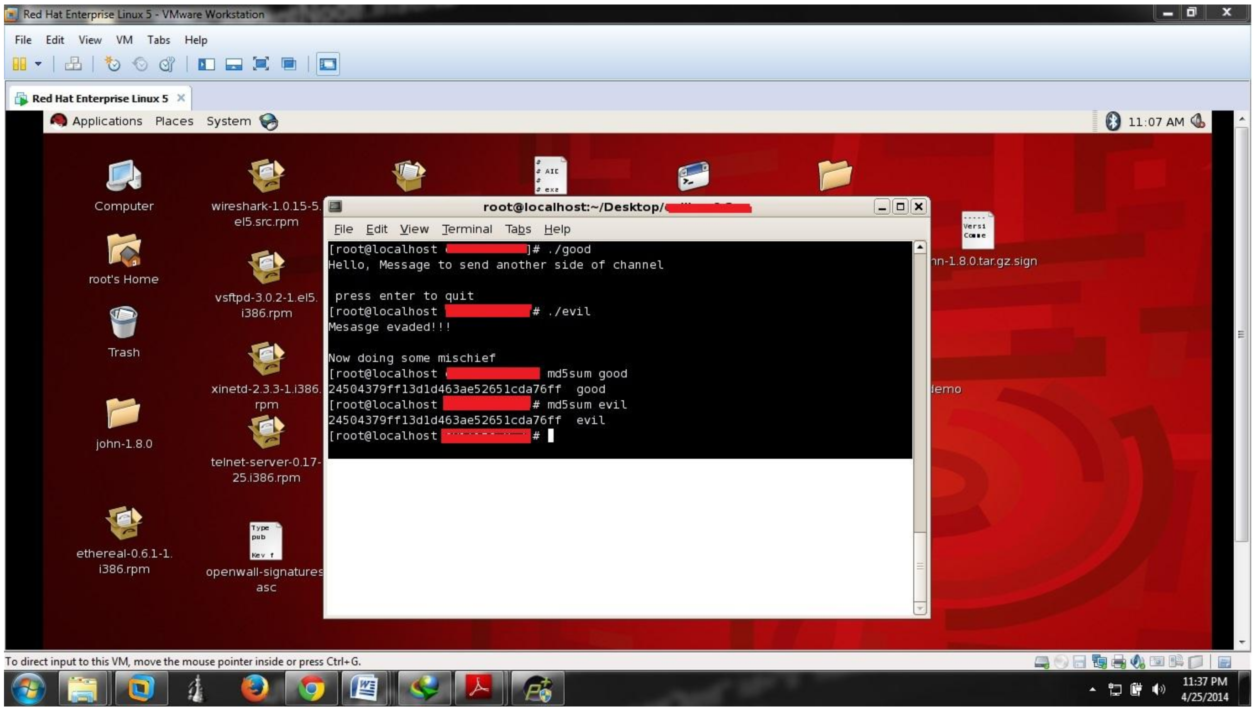
<u>Exploitation of Hash Functions:</u> The term exploitation means to take the unfair advantage of someone for their own benefits. So, here are we also going take the advantage of the vulnerability of above hash functions. There is a full fledge family of hashes like SHA family, MD-0 family (famous one MD5). So let’s try to exploit.

<u>#Exploit 1:</u>
<u>Multi-Collisions Attack :</u> The very first attack that I am going to discuss is the Multi-Collision Attack. This attack works on the methodology that first researchers said after the creation of hash values that this is almost impossible to create such messages which results in the same hash value. But this was actually broken by the Chinese students Wang and Yu who worked so hard to break down the MD5 and proved that this can also work in practice. Later on Peter Selinger was the man who actually showed how the attack works. So, I am continuing here by the whole explanation of the attack that how it works and what was back-ground of this particular attack. And again we get into deep and deep to understand the working of the exploit.

<u>Explanation:</u> Just remember that MD5 taken on an arbitrary length of data and gives back the 128-bit hash value. Rest of the working is same as the SHA-1 as explained above. It is based on the Markel Damgard Construction and also based on the iterative manner to produce hash. For padding it divides the file to 64 bytes of equal length of blocks and it appends the value to 448 modulo 512 and takes on 4 words (A,B,C,D) to process the round functions.

<u>Working of Exploit:</u> The exploits works as follows:
Two files are being generated with different messages but their md5sum is same. This cannot be possible as theoreticians said. But below in the Snapshot you can check POC, that this can also be possible. Now below I am just showing you a very toy example that how these can be exploited but there can much more dangerous things that can happen with large firms like Banks, Security Agencies,

Companies etc who uses MD5 hashes and similar to its level which is shown below:


Red Hat Enterprise Linux 5 - VMware Workstation
File Edit View VM Tabs Help
Red Hat Enterprise Linux 5
Applications Places System
11:07 AM
Computer
root's Home
Trash
john-1.8.0
ethereal-0.6.1-1.i386.rpm
wireshark-1.0.15-5.el5.src.rpm
vsftpd-3.0.2-1.el5.i386.rpm
xinetd-2.3.3-1.i386.rpm
telnet-server-0.17-25.i386.rpm
openwall-signatures.asc
root@localhost:~/Desktop/[redacted]
File Edit View Terminal Tabs Help
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.
11:37 PM
4/25/2014

```
[root@localhost [redacted]]# ./good
Hello, Message to send another side of channel

 press enter to quit
[root@localhost [redacted]# ./evil
Mesasge evaded!!!

Now doing some mischief
[root@localhost [redacted] md5sum good
24504379ff13d1d463ae52651cda76ff  good
[root@localhost [redacted]# md5sum evil
24504379ff13d1d463ae52651cda76ff  evil
[root@localhost [redacted]#
```

Snapshot-1

Scenario:
In the above snapshot there are two files with names good and evil and there is written different-different texts. So, a scenario could be like this a person sitting inside a firm and wants to communicate with another person. Now the message needs to be sent with full integrity, means tempering should not be there. So, what sender does is he simply creates the md5sum of it and sends it to the receiver oven an insecure channel and tells the receiver about the md5sum. Now there was an eve sitting in between. He simply jumps into the network and gets hold of the data. Now as we know that with the hash value, message is also sent. So, eve develops his own message file what he wants in there and produced the same hash value and sent it to the receiver. Receiver will simply open up the file and checks for the md5 value. And it's coming to be same and he simply accepts the message. So, this is the biggest exploitation and the breakdown of the md5sum. But still people are using the md5sum as in their websites to store up the database values.

Working of the Exploit

So, how this magic happened? Ok I will tell you the whole magic. There is no magic in there, only a brain with evil things and hard work is there. So, the whole idea is to grab the files and create the programs of it and change the extensions of it to whatever they are sending. Like I have created here is .txt files from the compiled .c programs. So, this was the whole trick. But still this won't give you what you really expecting is. There has to be a way to do this also.

Grab the text and the hash value from the sender. Create your .c programs with the values you want to display on their screens. So, what you do is,

Program 1: if (data1 == data1) then print (good) else print (evil)
Program 2: if (data1 == data1) then print (good) else print (evil)

So, whatever the conditions that will follow here it is always going to be a true value. Here note that good and evil are the files with the messages. So, how these two programs have to be compiled properly. This theory was actually given by Peter Serlings. The file with the original mesasge is first broken down into equal blocks of length 64 bytes i.e if file with X message is given, and then there will be blocks of say x1, x2, x3 …………,xn is being created. And MD5 is being computed using a sequence of states i.e. z1,z2,z3………..,zn.

According to the rule,

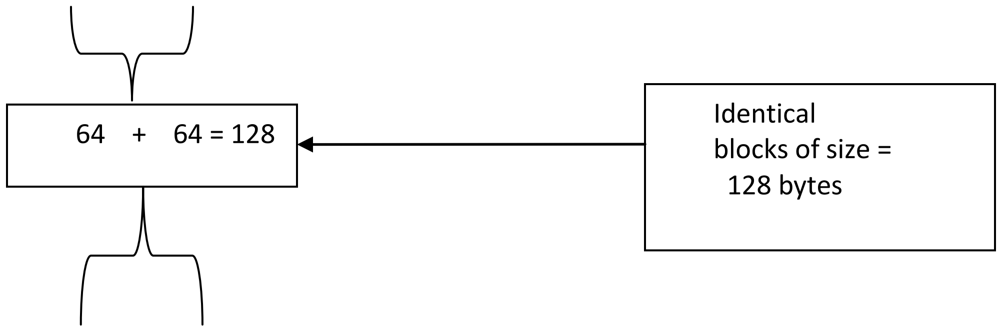
z(i) = f(z(i-1), x(i-1)); where f is a given certain fixed valued function.

z1 = the initialization vector used for very first round step
z(i) = is the final hash output = H(x)

Now let's say that another message that the attacker want to compute is the x' over the same initialization vector z1, then we have to find such two pair of block messages which results in the same value given as:
f(f(z,x(i-1)),x(i)) = f (f(z,x'(i-1)),x'(i));

Elaborating the above value to make it more simple and easy to understand. Diving the message into blocks of 64 bytes assuming the whole message is padded.

X = x1, x2, x3…………***x(i-1), x(i)***……………… xn



X’= x’1,x’2,x’3………***x’(i-1),x’(i)***……………...x’n

So, I am not actually interested in whole above arbitrary values and afterwards arbitrary byte values. What I am interested is in these above two highlighted values. Because these two 128 bytes of identical block will give me the same hash value. So, what I have done is the same thing I found the two identical blocks and apply the hash algorithms only on these two identical 128 bytes of value. Hence therefore, I created two programs as shown above.
So, this was the whole explanation for the exploitation of the MD5 hash values.

#Exploitation 2:
Forging of Hash Based MACs:
Another possible attack on the hash functions is forging. Forging means to copy or produce the identical values for a given message or signature. As discussed above that MACs are the keyed hash functions, it means there is a key that has been inserted in the algorithm along with the hash function to produce a fixed length digest. Let’s explain this particular attack by taking a suitable example:
Given,

Key Length (K) = 80 bits
Message (X) = 256 bits
Hash Algorithm = SHA-1or any other family

Now output that has to be produced is of = 160 bits
So, very first step is to check for padding
Mathematical Formulations:
256 + 80 = 336 bits
Required bits = 512 bits
Remaining bits = 512 – 336 = 176 bits
Original 64 bit length = 176 – 64 = 112 bits
Final Padded bits required is = 112 bits

Add bogus bits = 1 (fixed) 0 pow (111)

Now the original message that will be hashed is of 160 bits and the representation will go like this:

H(x) = SHA1(K||X) = C(K||M||10000…….0|| 101010100000………..1010)

The hash value will be simply gets concatenated with the message and will be sent over the channel. As the assumption has always been made that the medium of sharing is always unsecured channel. So, adversary will simply have a control over the message and the hash value. Now how to forge this particular message??

What adversary has in his hands??
(H(x) || X)

Now let's say adversary has to forge the message with X' value. So, now the work of adversary is that he knows the X value or he will request the MACs from the source, and he will generate another valid pair of (x',y) such that the value of x' should not match with any of the corresponding values of the given x1,x2,x3…….xn values. Then the attacker will be having a valid pair for forgery of MACs.
So, the idea is like this:
Adversary will simply look for the given X values and break down into the following values of x1,x2,x3………..xn

X= x1 || x2 || x3 || ……………….||xn

Now attacker will simply try to create the hash values these broken values of X and start matching its hash values that whether the values match with the user's value or not. And then he will create his own X' value that matches the same number of bits with the given X value. And will break down it in the same way:

X' = x'1 || x'2 || x'3 || …………………………….|| x'n

Now he will start creating the hash values for these particular broken down message values for X'. And start storing it in a buffer. Now looking in both the buffers, if the attacker has successfully calculated the hash value for (X',Y) = ((x'1,Y),(x'2,Y),(x'3,Y)………..,(x'n,Y)) which matches for any of the above calculated hash values for (X,Y) = (x1,Y),(x2,Y)……….(xn,Y), then the attack has

successfully completed and GAME OVER will be there. So, this is how the forgery attack has been donw.
Now this attack comes under the category of the known plain-text attack. And the forging will be possible here, because the attacker is actually gets a hold of particular transfer of data over the channels and can successfully create the valid pair of (x',Y) = (X,Y).

Conclusion: So, these are how the attacks that are being done practically over the hashing algorithms. Till now SHA-2 and SHA-3 are safe but I am trying hard to find a way in order to get these down too. So, in my view use SHA-3 algorithms and also whatever the transactions that is being made should be done with IPSec, having an encryption scheme and the secret keys that are to be used should be a One-Time Pad values. And always digitally sign the messages with a proper calculated hash that has to be sent along with the message. Only then the data will be sent properly and safely over the channels.