International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
*Volume **– No.*, ____ 2012 – www.ijais.org*

Hybrid HMAC using Improved SHA-512

Kamal Shah, Ph.D
Professor, IT Department
Thakur College of Engineering and Technology
Mumbai-400101, India.

Vikas Kaul
Assi.Professor, IT Department
Thakur College of Engineering and Technology
Mumbai-400101, India.

Pratik Kanani
M.E.I.T (pursuing)
Thakur College of Engineering and Technology
Mumbai-400101, India

ABSTRACT
With the fast progression of digital data exchange in electronic way, information security is becoming more important in data storage and transmission. Cryptography has come up as a solution which plays a vital role in information security system against malicious attacks. The confidentiality, Integrity and Availability are the three main goals of Information Security. To protect Confidentiality and Integrity of the message, key mechanisms and Hash functions are used. This paper proposes a novel scheme, which computes Hybrid HMAC based on improved SHA-512 algorithm. The proposed system gives a constant output of 512 bits for any input length of data and generates HMAC using multiple available keys. Paper also provides a method for reducing the time taken by traditional HMAC functions and increases complexity of SHA-512.

General Terms
Information Security, Data Integrity.

Keywords
HMAC, SHA-512, Modified SHA-512, Avalanche effect, Buffers.

1. INTRODUCTION
Robust and fast security functionality is basic tenant for secured computer transactions. A **hash function** H accepts a variable-length block of data as input and produces a fixed-size hash value h=H(M) with the objective principle of data integrity. Hash functions needed for security applications are stated as **Cryptographic Hash Function**.

National Institute of Standards and Technology (NIST) as a U.S. Federal Information Processing Standard (FIPS) endures a category of cryptographic hash function known as **Secure Hash Algorithm** (SHA)[1].

Hashing algorithms has certain hitches within its community, with their security receiving inattention than standard encryption algorithms and excluding speed parameter. Hence many standards and products have started preferring larger hash sizes. This comes with a cost of SHA-256 which is about 2.2 times slower than SHA-1.

SHA-512[2] is faster than SHA-256 on 64-bit machines because it has 37.5% less rounds per byte. The adoption across the extensiveness of the range of 64 bit ALU's make it probable to achieve enhanced security using SHA-512 in less time than it takes to compute a SHA-256 hash[3].

Generally for message authentication, **Message Authentication Code (MAC)** is used between two parties that share a secret key to authenticate information exchanged between those parties. A MAC function takes as input, a secret key and a data block and produces a hash value, referred as MAC. The combination of hashing and encryption results in an overall function i.e. E(K, H(M)) called as **Hash Based Message Authentication Code (HMAC)**. It is a function of a variable-length message and a secret key that produces a fixed-size output which is secured against an opponent who does not know the secret key.[1,2,3]

2. SHA – 512

2.1 Previous SHA Versions
Multiple versions of SHA exists, such as SHA-0(160 bit), SHA-1(A 160-bit hash function resembling the earlier MD5 algorithm), SHA-2(224-bit, 256-bit, 384-bit, 512-bit), SHA-3(the same hash lengths as SHA-2, and its internal structure differs significantly from the rest of the SHA family)[1].

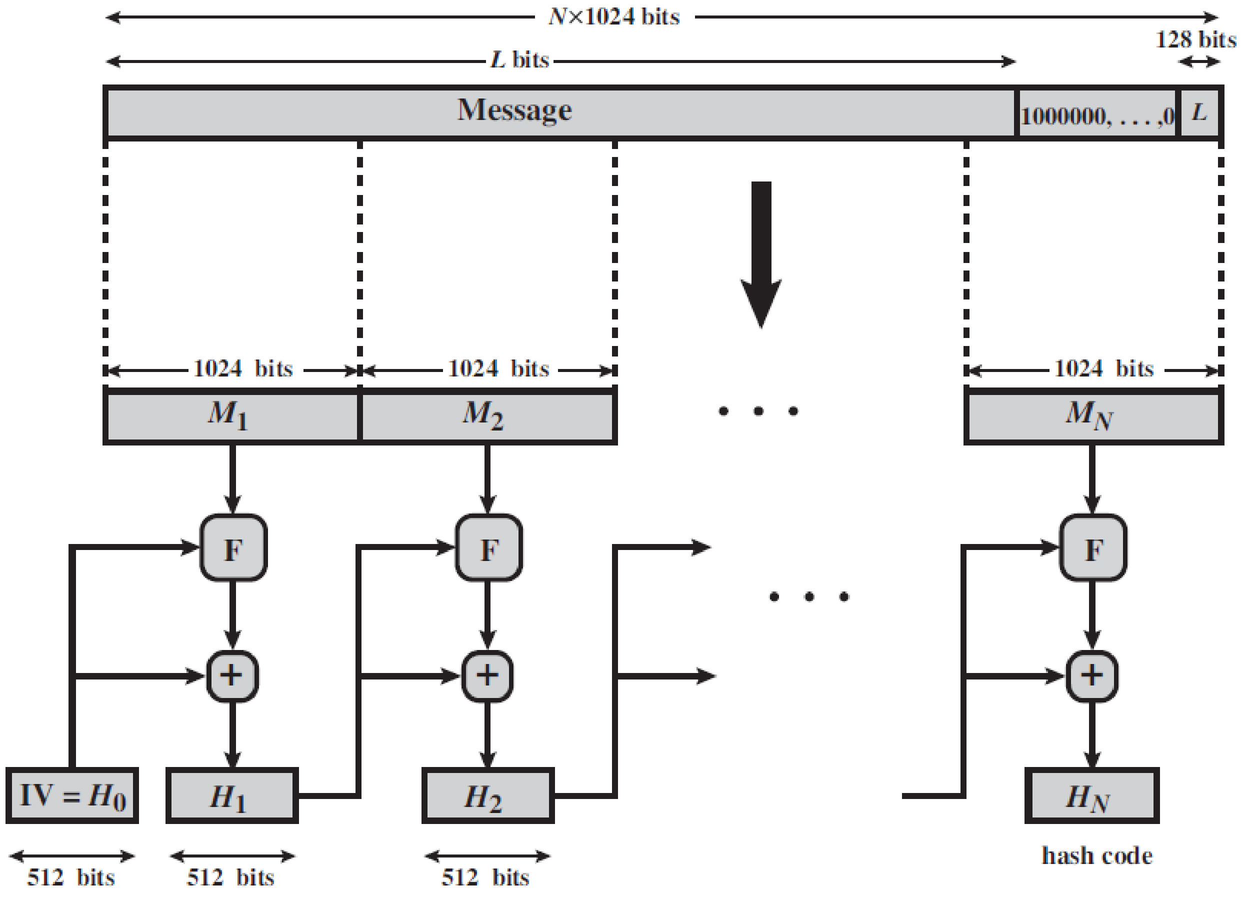
Table 1. Comparison of SHA parameters [2]

Versions And parameters	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Message Digest size	160	224	256	384	512
Message size	$<2^{64}$	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
Block size	512	512	512	1024	1024
Word size	32	32	32	64	64
Number of Steps	80	64	64	80	80

The algorithm takes as input a message with a maximum length of less than 2^{128} bits and produces as output a 512-bit message digest. The SHA-512 compression function operates on a 1024-bit message block and a 512-bit intermediate hash value. The basic eight buffers used in SHA-512 are as follows.

a = 6A09E667F3BCC908	e = 510E527FADE682D1
b = BB67AE8584CAA73B	f = 9B05688C2B3E6C1F
c = 3C6EF372FE94F82B	g = 1F83D9ABFB41BD6B
d = A54FF53A5F1D36F1	h = 5BE0CD19137E2179

These buffers are obtained by taking the fractional parts of the square roots of the first eight primes.
The input is divided into to 1024 bits block and then processed. If the input is less than the 1024 bits then the extra bits are appended to make 1024 bits.
The message digest generation is shown in Fig.1.



+ = word-by-word addition modulo 2^{64}

Fig 1: Message Digest Generation Using SHA-512 [2].

2.2 SHA – 512 Compression Function

The SHA -512 compression functions can be given as follows. Where 80 iterations are done .

```
For i=0 to 79
{
    $T_1 \leftarrow h + \Sigma_1(e) + Ch(e,f,g) + K_i + W_i$
    $T_2 \leftarrow \Sigma_0(a) + Maj(a,b,c)$
    $h \leftarrow g$
    $g \leftarrow f$
    $f \leftarrow e$
    $e \leftarrow d + T_1$
    $d \leftarrow c$
    $c \leftarrow b$
    $b \leftarrow a$
    $a \leftarrow T_1 + T_2$
}
```

Where , $Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge y)$

$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$
$\Sigma_0(x) = S^{28}(x) \oplus S^{34}(x) \oplus S^{39}(x)$
$\Sigma_1(x) = S^{14}(x) \oplus S^{8}(x) \oplus S^{41}(x)$
$\sigma 0(x) = S^{1}(x) \oplus S^{18}(x) \oplus R^{7}(x)$
$\sigma 1(x) = S^{19}(x) \oplus S^{61}(x) \oplus R^{6}(x)$

R→ left shift of the 64-bit argument x by n bits with padding by zeros on the right
S→ circular right shift (rotation) of the 64-bit argument x by n bits
⊕→ bitwise XOR
∧→ bitwise AND
∨→ bitwise OR
¬→ bitwise complement
+→ mod 2^{64} addition

$W_i = M_i$ for i=0,1,2…15 and

```
For i=16 to 79
{
        $W_i \leftarrow \sigma 1(W_{i-2}) + W_{i-7} + \sigma 0(W_{i-7}) + W_{i-16}$
}
```

The entire message of 1024 bits is divided into the chunks of 64 bits, where each 64 bit chunk is represented as W. The W starts from 0 and goes till 79. The definition of W is given above.

After 79th iteration the new buffers are added to the old buffers and by appending all buffers respectively it forms the output message.

$A_1 = a + new(a)$
$B_1 = b + new(b)$
$C_1 = c + new(c)$
$D_1 = d + new(d)$
$E_1 = e + new(e)$
$F_1 = f + new(f)$
$G_1 = g + new(g)$
$H_1 = h + new(h)$

The final message digest = A1 B1 C1 D1 E1 F1 G1 H1

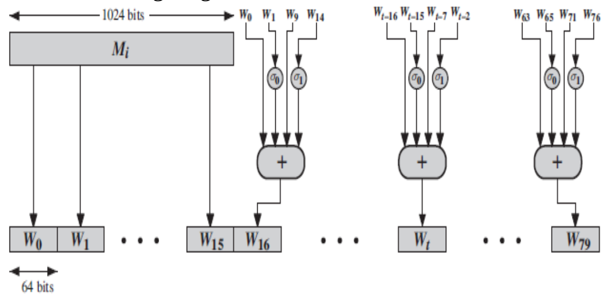

1024 bits
M_i
W_0 W_1 W_9 W_{14}
W_{t-16} W_{t-15} W_{t-7} W_{t-2}
W_{63} W_{65} W_{71} W_{76}
σ_0 σ_1
+
W_0 W_1 W_{15} W_{16} W_t W_{79}
64 bits

Fig.2 Creation of 80-word Input Sequence for SHA-512 processing of Single Block [2]

To compute HMAC function by using exiting SHA-512, first we need to find the message digest for the given input length and by encrypting the message digest value using some key.

$$HMAC(M)=E(K,H(M)).$$

3. Improved HMAC Using Modified SHA

3.1 New Buffers

The eight new buffers used in improved SHA are as follows

a = 6A67E685F3CAC93B	e = 5105528CAD3E821F
b = BB09AE6T84BCA708	f = 9B0E687F2BE66CD1
c = 3C4FF33AFE1DF8F1	g = 1FE0D919FB7EBD79
d = A56EF5725F94362B	h = 5B83CDAB1341216B

These buffers are obtained by taking the permutations and combinations of the available buffers to increase the avalanche effect and the complexity of the message digest.
The message digest function is same as the Fig.1.

3.2 Modified SHA – 512 Compression Functions

The modified SHA-512 is same as traditional SHA-512 with the only change in some of the compression functions that are changed and the new buffers are used. After taking the required text to find its Hash function append the string of o's and one's to it. E.g. 0101010101…

The modified SHA -512 compression functions are as follows. Where 80 iterations are done .

```
For i=0 to 79
{
    $T_1 \leftarrow h + \Sigma_1(e) + Ch(e,f,g) + \mathbf{K_i} + W_i$
    $T_2 \leftarrow \Sigma_0(a) + Maj(a,b,c)$
    $h \leftarrow g \mathbf{+ Ki}$
    $g \leftarrow f$
    $f \leftarrow e \mathbf{+ c}$
    $e \leftarrow d + T_1$
    $d \leftarrow c$
    $c \leftarrow b$
    $b \leftarrow a$
    $a \leftarrow T_1 + T_2$
}
```

Where , $Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge y)$
$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$
$\Sigma_0(x) = S^{18}(x) \oplus S^{44}(x) \oplus S^{29}(x)$
$\Sigma_1(x) = S^{44}(x) \oplus S^{8}(x) \oplus S^{21}(x)$
$\sigma 0(x) = S^{7}(x) \oplus S^{38}(x) \oplus R^{34}(x)$
$\sigma 1(x) = S^{15}(x) \oplus S^{61}(x) \oplus R^{26}(x)$
K_i = Key used for encryption for HMAC
$R \rightarrow$ left shift of the 64-bit argument x by n bits with padding by zeros on the right
$S \rightarrow$ circular right shift (rotation) of the 64-bit argument x by n bits
$W_i = M_i$ for i=0,1,2…15 and

```
For i=16 to 79
{
    $W_i \leftarrow \sigma 1(W_{i-2}) + W_{i-7} + \sigma 0(W_{i-7}) + W_{i-16}$
}
```

This approach differs from traditional SHA-512 algorithm by an additive constant k. However in this proposed idea the additive constant k is replaced by the Key. And furthermore the key at iteration is used if

$K_i\%i == 0$ for i<10
$K_i\%i == 0$ for i>10, i=i/10.

It means that multiple keys are available to calculate HMAC function and based on the above conditions each key will be used at particular number of iteration and with this while finding a SHA – 512, an encryption is also done so no extra efforts are required for the same.

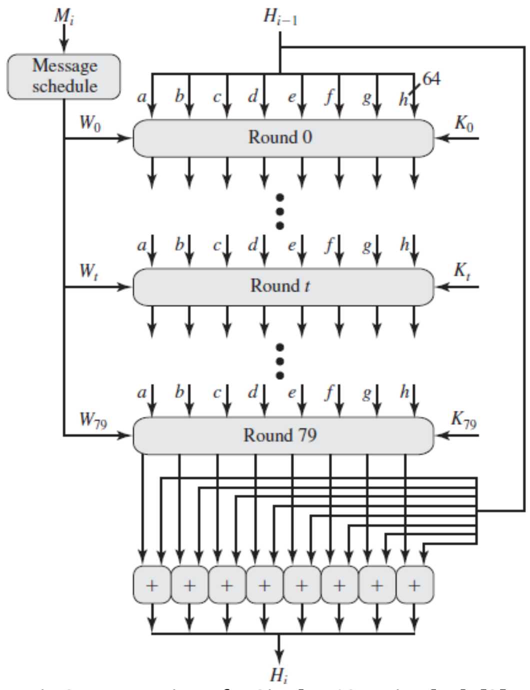


Fig.3 Processing of a Single 1024-Bit Block [2]

In the above figure shown, at each iteration the respective key K will be introduced to find SHA – 512 and HMAC.

4. PERFORMANCE IPMPROVEMENTS

Performance improvements for any system can be accomplished in two ways. 1) Majorly, by modifying the architecture design of the system and 2) Slightly, by applying some software and hardware techniques that is implementation idea plays a role. In this system we have used some of the compact coding techniques. The above system is implemented in java programming language and the performance is measured.

The traditional SHA-512 was implemented using recursion function as (for W_i)

```
String getw(String Binaryinput,int k)
   {
         String w=Binaryinput.substring((64*k),(64*(k+1)));
         return (w);
   }

String getW(String Binaryinput,int k)
   {
      if(k<16)
      {
         String W=getw(Binaryinput,k);
         return (W);
      }
      else
      {
         ………..Find Corresponding W by
            W_i ← σ 1(W_i-2) + W_i-7 + σ 0(W_i-7) + W_i-16
      }
```

In above piece of code whenever the W is requested the function checks for w existence and if not it goes for W. But for higher passes the function even calculates every w and W recursively which consumes lots of system memory and time.

So an amendment to this can be

```
String WBuffer[80];

String getw(String Binaryinput,int k)
   {
         String w=Binaryinput.substring((64*k),(64*(k+1)));
         WBuffer[k]=w;
         return (w);
   }

String getW(String Binaryinput,int k)
   {
      if(!WBuffer[k]=="")
       return(WBuffer[k]);

      if(k<16)
      {
         String W=getw(Binaryinput,k);
         return (W);
      }
      else
      {
         ………..Find Corresponding W by
         W_i ← σ 1(W_i-2) + W_i-7 + σ 0(W_i-7) + W_i-16
        WBuffer[k]=W;
        return(WBuffer[k]);
       }
    }
```

The new logic initially works same as old logic and takes almost same time and space as initial logic but as loop goes to higher passes it will be much efficient than the previous passes, since at each and every iteration it will check for the existing value of W in an array of WBuffer[]. If the value exist, than it returns the same, and if not than it calculates the new value of W and stores in the array. It means, each W is calculated only once in a second piece of code.
Furthermore new available data types in java such as BigInteger and BigDecimal can be used which provides in built functions for logical operations such as AND, OR, NOT, MOD, Ex-OR, which gives better and fast results rather than working every time with String data types where each operation has to be implemented manually.[4,5]

5. SIMULATION

The simulation program is developed using java. This model asks user to input the message of any length and gives output of fixed length 512bits. The SHA – 512 and improved SHA - 512 and HMAC are generated and different parameters are noted down as follows.

Table 2. Specification of the test-bed hardware and software.

CPU	Intel core 2 duo 1.60FHz
RAM	1GB
Platform	Windows XP 32-bit

5.1 Time Requirements

The times taken by the functions are uncovered using System.currentTimeMillis(). Only first 35 passes are executed and the corresponding results are shown.

Table 3. Time taken in ms

Iteration	SHA-512	Modified SHA-512	HMAC
10	172	187	201
15	453	475	490
20	610	625	653
25	**969**	**975**	**986**
30	**1922**	**1715**	**1732**
35	**5163**	**4632**	**4666**

As the iteration goes higher the time taken by program reduces.

5.2 Space Requirements

The space requirements by the system are discovered by using java Runtime class and its methods. Space requirements of first 35 passes are shown in table 4.

From the given table it can be concluded that after the modification in recursion function the space required by modified SHA-512 and HMAC is less than the traditional SHA for higher iterations.

Table 4. Space requirements in bytes

Iteration	SHA-512	Modified SHA-512	HMAC
10	460296	470200	470542
15	1073224	1092334	1094579
20	1142552	1142599	1143652
25	**863744**	**853252**	**853895**
30	**1525376**	**1437440**	**1438104**
35	**2662680**	**2542440**	**254304**

5.3 Evaluation parameter

Each of the encryption techniques has its own strength and weaknesses. In order to analyze the encryption schemes the critical analysis of such techniques are necessary. A desirable property of such algorithms are a small change in the input must cause the bigger change in the output that is in cipher text. So that a small change also makes the output more complex in order to analyze it.

$$\textbf{Avalanche Effect} = \frac{\textbf{Number of flipped bits}}{\textbf{Number of bits}}$$

When plain text and cipher text is calculated to find the Avalanche effect[6] the result is as follows.

Table 5. Avalanche Effect in %

Iteration	SHA-512	Modified SHA-512	HMAC
10	42.24	42.58	42.68
20	48.34	48.66	48.96
30	**51.96**	**52.21**	**52.47**
35	**52.266**	**52.30**	**53.01**

The avalanche effect does not vary much between modified SHA-512 and HMAC but it matters in HMAC function since the encryption key is involved. The other existing tools can be used to find the similarity between plain test and the cipher text.[7]

6. CONCLUSION

In this paper a modification in SHA-512 algorithm is done to in order to improve its efficiency and to find HMAC out of it. This proposed algorithm suggests new buffers for modified SHA-512 algorithm and some of the compression functions are added. It is then simulated in java environment and its corresponding time, space and avalanche effect has been measured. The results obtained shows that the improvement to existing SHA-512 is done.

In Future, this function can be more advanced with more complex buffers and functions. Also it can be integrated with other security parameters to make a system which will alone ensures the CIA of Information Security.

7. REFERENCES

[1] Secure Hash Algorithm, Wikipedia (access on 10 January 2014) http://en.wikipedia.org/wiki/Secure_Hash_Algorithm

[2] William Stallings, Cryptography and Network Security : Principles and Practice , fifth Edition, Prentice hall.

[3] S.Gueron, S.Johnson and J.Walker, "SHA-512/256", *IEEE Conference on IT:new Generations*, pp.354-358, 2011.

[4] BigDecimal and BigInteger, Stackoverflow (access on 20 January 2014) http://stackoverflow.com/search?q=bigdecimal+and+biginteger

[5] Java Optimization Rules, Appperfect (access on 22 January 2014) http://www.appperfect.com/support/java-coding-rules/optimization.html

[6] A.Mandal and A.Tiwari, "Analysis of Avalanche Effect in Plaintext of DES using Binary codes", *International Journal of Emerging Trnds and Technology in Computer Science,* vol. 1 , pp. 166-171, 2012.

[7] String Similarity test, Tools4noobs (access on 2 February 2014) http://www.tools4noobs.com/online_tools/String-similarity