d-HMAC — An improved HMAC algorithm

Article · April 2015

CITATIONS
4

READS
2,544

1 author:

Mohannad Najjar
Vtech Systems LTD. Winnipeg- CANADA
11 PUBLICATIONS **49** CITATIONS

SEE PROFILE

d-HMAC — An improved HMAC algorithm

Mohannad Najjar
University of Tabuk
Tabuk, Saudi Arabia
najjar@ut.edu.sa

***Abstract*—The keyed-hash message authentication code (HMAC) algorithm is a security tool primarily used to ensure authentication and data integrity in information systems and computer networks. HMAC is a very simple algorithm, and relies on hash functions that use a secret key. HMAC's cryptographic strength is based on the use of effective cryptographic characteristics such as balancing and the avalanche effect. In this study, we develop a new algorithm, entitled dynamic HMAC (d-HMAC), to improve and enhance the cryptographic characteristics of HMAC. The improved algorithm provides stronger resistance against birthday attacks and brute force attacks. To achieve this objective, HMAC constant values *ipad* and *opad* are dynamically calculated in d-HMAC. Values for *ipad* and *opad* will be obtained from the HMAC input message, the public key of the receiver, and a substitution-box (*S*-box) table with enhanced security characteristics specifically created for this purpose. We demonstrate that the improved d-HMAC algorithm is more resistant to known cryptographic attacks, and prove that it exhibits similar or better cryptographic characteristics than HMAC.**

Keywords-cryptography; data integrity; authentication; MAC; HMAC; hash functions; SHA-256

I. INTRODUCTION

Authentication is one of the primary aspects of security, because it confirms the identity of the source and ensures that data has not been altered. Message authentication codes (MACs) are one of the most important authentication and data integrity tools. In this study, we will improve the functionality of HMAC, which is a type of MAC that uses hash functions.

Compared with other MAC types, HMAC is considered more effective, as described by [12]:

- MAC uses encryption algorithms that are relatively slow.
- Many hardware cryptographic tools are built to manage large volumes of data.
- Many cryptographic algorithms require licenses, whereas HMAC is free of charge.

.

HMAC is a cryptographic tool that ensures authentication and data integrity [2,5]. HMAC is used in data exchanging and warehousing, to ensure the validity of the source. HMAC is a specific type of MAC function that can use any type of hash function that uses a secret key shared between two parties, to process an input message *m*. HMAC is one of the most prominent cryptographic algorithms, and is used to ensure that saved or exchanged data is not changed (intentionally or accidently) without authorization. HMAC uses two values, *ipad* and *opad*, to ensure that keys are pseudorandom. These values are fixed and known, which causes security weaknesses against known cryptographic attacks [13].

In this study, we present an improved d-HMAC algorithm, which uses dynamic *ipad* and *opad* values to provide more robust security than HMAC. Dynamic *ipad* and *opad* values increase the d-HMAC algorithm's resistance to known cryptographic threats such as birthday and brute force attacks. Furthermore, d-HMAC is an effective tool for creating pseudorandom initial values for hash functions.

Cryptographic tests will be conducted against different types of security weaknesses; this will show the effectiveness of d-HMAC compared with HMAC, and demonstrate d-HMAC's useful cryptographic properties, including balance and avalanche effects.

List of important symbols used in this paper (National Institute of Standards and Technology, 2002):

h	Hash function: SHA-256.
b	Number of bits in block h.
IV	Hash function initial value.
m	Input data processed by HMAC.
Yi	Message m ith blocks, $0 \le i \le (l\text{-}1)$.
l	Quantity of blocks in message m after bit padding.
N	Lengths of message digest.
K	Secret key, if K length > b then $K = h(K)$.
$K+$	K padded with zeros.
$ipad$	Inner pad of (36_H) reiterated $b/8$.
$opad$	Outer pad: $(5c_H)$ reiterated $b/8$.
$h(m)$	Message digest for d-HMAC with n bits length.
Y	All possible message digests.
$\oplus$	XOR operation (bitwise exclusive-OR).

This paper is organized as follows: in Section 2, hash functions are introduced; in Section 3, HMAC functions are explained; in Section 4, the improved d-HMAC algorithm is described; in Section 5, test results are presented; in Section 6,

http://sites.google.com/site/ijcsis/
ISSN 1947-5500

d-HMAC's improved resistance to cryptographic attacks is presented; in Section 7, we present our conclusions.

II. HASH FUNCTIONS

A hash function is a cryptographic tool used to ensure the integrity of data by preventing unauthorized or accidental modifications. A hash function may also be called a manipulation detection code (MDC). Moreover, hash functions are also used for other cryptographic applications, such as digital signatures and password storage [8,11].

A hash function is a function $h: M \rightarrow Y$ that must fulfill the following requirements (where $m \in M$, $y \in Y$):

- It compresses message m with a distinct length to message digest $h(m) \in Y$ with a fixed length.
- It can straightforwardly compute message digest $h(m)$ for any message m.
- It is computationally infeasible to compute any $m' \in M$ for most $y \in Y$ where $y = h(m')$ (one-way characteristic).
- It is computationally infeasible to compute another message m' for message m where $h(m) = h(m')$ (pre-image resistance characteristic).
- It is infeasible to compute two different messages m and m' where $h(m) = h(m')$ (Second pre-image resistance characteristic).

The hash function transformation "Fig. 1" for message $m = m_1 \| m_2 \| \ldots \| m_t$ is divided into fixed length blocks m_1, m_2,…, m_t and can be described as follows:

$$H_0 = IV$$

$$H_i = \varphi(m_i, H_{i-1}), \ Where \ i = 1,2,...,t$$

$$h(m) = H_t$$

Where message m and fixed initial value IV are the input of the hash function, H_i is the chain variable calculated by compression function φ and the ith block of m, and the output result is $h(m)$. $h(m)$ has different names in the cryptography literature, including hash result and fingerprint [10,11]. In this paper, it is referred to as the message digest. The hash function structure is depicted in Fig. 1 [8].

HMAC and d-HMAC can use any type of hash function, including MD5, RIPEMD-160, SHA-256, SHA-512, and PETRA. Most hash functions have constant initial values, except the PETRA hash function, which has dynamic initial values [3]. In this paper, we will focus on the SHA-256 hash function; thus, we will conduct all of our d-HMAC tests using SHA-256 [6].

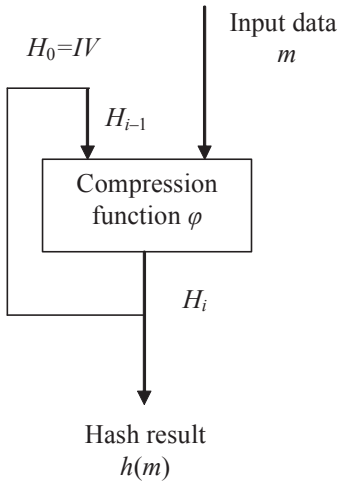

Figure 1. General model of the hash function h

III. KEY-HASHED MESSAGE AUTHENTICATION CODE (HMAC)

Authentication is one of the primary aspects of security [17], and ensures the authenticity of senders, receivers, and data. It confirms that senders and receivers are who they claim to be. MAC is one of the main cryptographic tools used to ensure authentication. MAC can be constructed by using block cipher algorithms, or by using a hash function that employs a secret symmetric key. In this work, we will focus on HMAC, which is a specific type of MAC that uses hash functions and secret keys.

The main design objectives of HMAC functions are as follows [2]:

- HMAC can use any type of hash function as it is. Most hash functions are available free of charge.
- HMAC does not interfere with the hash function, thus its performance is not negatively affected.
- HMAC prefers to use readily available keys in a straightforward manner.
- HMAC's algorithm is simple and easy to modify, allowing the security level or speed of the underlying hash function to be upgraded if necessary.

We assume that HMAC can use any hash function h without modifications and secret key K. Hash function h will execute based on compression function φ for a message m containing l blocks. The length of each block l in bits is denoted by b, which indicates that $l * b$ will equal the length of message m after bit padding. We denote by n the length of the message digest in bits. In our research, we use SHA-256 functions, which results in $n = 256$. The length of shared secret key K can equal b or less; for keys longer than b bits, a hashing computation must be performed using the h function. HMAC's designers recommend using a K that is at least n bits long. Using keys with lengths less than n is discouraged, because it reduces the strength of HMAC's security functions.

http://sites.google.com/site/ijcsis/
ISSN 1947-5500

HMAC's designers recommend the use of highly random keys, and they recommend changing keys as a standard security practice. This minimizes the negative consequences of exposed keys, and mitigates threats that depend on the collection of data calculated by HMAC using the same key, such as offline brute-force attacks.

It is advantageous to improve hash functions to use dynamic initial values *IV* instead of fixed values. Furthermore, HMAC calculates intermediate values of ($K^+ \oplus opad$) and ($K^+ \oplus ipad$) once for the same *k*. These intermediate values will be used many times to authenticate the same key; as a result, these values, including the secret keys, must be protected against any type of disclosure [15].

The main equation of HMAC is defined as follows, Fig. 2:

$$HMAC_K(m) = h\left(K^+ \oplus Opad, h\left(K^+ \oplus Ipad, m\right)\right) \quad (1)$$

HMAC can be calculated using the following steps [2]:

1. K^+ is calculated by padding zeros onto K's left side to increase its length to *b*-bits.
2. Calculate S_i of *b*-bit length by XORing *ipad* with K^+.
3. Concatenate S_i with *m* to be equal to *m'*.
4. Calculate message digest *h*(m') for *m'* by using *h*.
5. Calculate S_0 of *b*-bit length by XORing *opad* with K^+.
6. Concatenate *h*(*m'*) with S_0 to be equal *m"*.
7. Calculate message digest *h*(*m"*) for *m"* by using *h* to get the final result for HMAC.

We can conclude that the main objective of using distinct constant values for *ipad* and *opad* in HMAC while calculating the message digest twice is to avoid cases in which:

$K^+ \oplus ipad$ OR

$K^+ \oplus opad$ will be a string of zero values

HMAC algorithm keys must have cryptographic characteristics that define their length, randomness, and complexity [2]. In terms of HMAC design principles, the bit length of key *K* can be any arbitrary size. The optimal size of *K* is equal to *b* size. For key sizes less than *n* bits are not recommended because it could weaken the cryptographic properties of the HMAC algorithm. Using a key *K* with a size greater than n bits does not add any cryptographic value to the algorithm. In cases in which the key's randomness characteristics must be increased, using a key larger than *n* is considered to be an advantage. (Keys longer than *b* bits are first hashed using *h*).

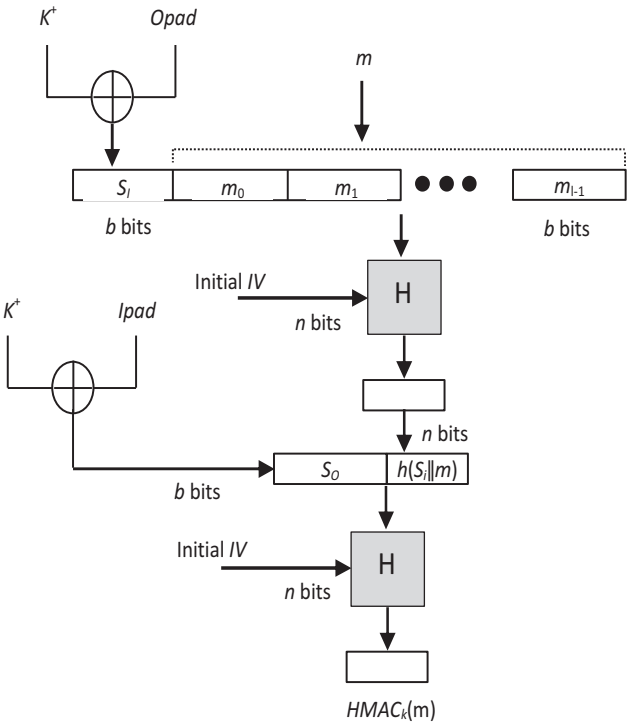

Figure 2. HMAC algorithm calculation

Furthermore, HMAC keys have two additional properties. First, the key must be random and calculated using an effective random value generation tool. Second, keys must be changed periodically, or a one-time key method must be used. There is no established guideline that specifies how many times a key can be repeated when guarding against attacks on HMAC and MAC algorithms. However, key changing is a cryptographic practice that is known to increase the strength of any cryptographic tool; such a practice will limit the exposure of keys [2].

HMAC functions can be compromised by one of the following schemes [9]. First, an attacker can compute hash results using a hash compression function, by using an exhaustive search attack (brute force attack) on the secret key in 2^n trials; second, the attacker can use a birthday attack to find two distinct messages m_1 and m_2 to calculate hash results $h(m_1)$ and $h(m_2)$, where $h(m_1) = h(m_2)$. A birthday attack requires $2^{n/2}$ trials to identify collisions for the selected hash function. In our case, we will require 2^{128} trials to break SHA-256. Although 2^{128} is a large number to calculate in logical time, the growing power of computers increases the threat that offline attacks will be used to find collisions. However, this attack can be effective only by guessing *K*. This can be accomplished by intercepting hash results calculated by the HMAC function, and using the same secret *K* to perform cryptographic attacks on these hash results.

http://sites.google.com/site/ijcsis/
ISSN 1947-5500

IV. DESCRIPTION OF IMPROVED D-HMAC ALGORITHM

Improved d-HMAC [1] operates in a manner similar to HMAC, but uses dynamic values for *ipad* and *opad* instead of fixed values. The calculations for *ipad* and *opad* depend on three parameters: the content of message *m*, the *S*-box table, and the receiver's public key e_R. The main purpose of using these parameters in this work is to calculate different *ipad* and *opad* values for distinct messages *m*, and to calculate different *ipad* and *opad* values for distinct receivers, which are denoted by *R*. Improved d-HMAC is expected to be more secure than HMAC, because it utilizes dynamic *ipad* and *opad* values [12].

In this paper, we developed an algorithm to create and calculate *ipad* and *opad* dynamically with effective cryptographic properties. Several tests have been conducted to prove that the d-HMAC algorithm is superior to the HMAC function.

We will discuss two HMAC cases and explain how our improvements will solve the weaknesses presented below. These weaknesses make HMAC vulnerable to threats such as birthday and exhaustive attacks [13].

Case 1: Data *m* is sent to many recipients using the same content and the same secret key *K*, which results in a similar $HMAC_K(m)$ for all recipients. This enables an attacker to collect message digests for message *m*, to generate offline attacks against HMAC. This weakness can be settled by sending distinct message digests $HMAC_K(m)$ to recipients for the same data and key. Improved d-HMAC uses the public key e_R of the receivers, where every receiver has its own unique public key. This improvement will prevent an attacker from initiating an offline attack.

Case 2: Different messages are sent to the same recipient using the same secret key *K*. In this case, an attacker will collect messages *m* sent to the same recipient, knowing that they were calculated by HMAC with the same *K*, and subsequently attempt an attack. Obtaining some of the messages calculated by the same *K* will not break HMAC; however, it is a step toward increasing the attacker's ability to launch an attack. In d-HMAC, this weakness was solved by calculating the dynamic values of *ipad* and *opad* according to the message *m*; thus, the sender can safely send any number of messages using the same key *K*.

Algorithm 1 was developed to apply the ideas mentioned above, by calculating *ipad* and *opad* dynamically. It is depicted in Fig. 3. Message *m* and Public key e_R and *S*-Box (table T) in Table 1 are the input values for which the dynamic values of *ipad* and *opad* will be calculated.

Algorithm 1 ***(dynamic calculation for* ipad, opad *values)***

Input: message *m*, public key e_R, T.
Method:

1. $w = h(m)$
2. $ip = w \oplus e_R$
3. A="", B="", g=0.
4. for i= 1 to n/4 do
   begin
   $g=g+1$
   x=""; y="";
   for j=0 to 3 do
   begin
   $x = x \,||\, ip[(g*i)+j]$
   end;
   $y = \bar{x}$
   s = T[Decimal(x)];
   z = Binary(s), where length of z equals 4 bits
   $A = A \,||\, z$
   v = T[Decimal(y)];
   w= Binary(v), where length of w equals 4 bits
   $B = B \,||\, w$
   end,
5. $ipad' = A$, $opad' = B$.

Output: *ipad'*, *opad'*.

TABLE I. S-BOX (T)

3	2	1	0	
12	9	3	5	**0**
9	3	10	9	**1**
6	5	12	6	**2**
10	12	6	3	**3**

(a)

Value	
5	**0**
3	**1**
9	**2**
12	**3**
9	**4**
10	**5**
3	**6**
9	**7**
6	**8**
12	**9**
5	**10**
6	**11**
3	**12**
6	**13**
12	**14**
10	**15**

(b)

Note 1. To maintain the balancing property, the 4-bit values used in the *S*-box of in Table 1 all have two "0" bits and two "1" bits.

http://sites.google.com/site/ijcsis/
ISSN 1947-5500

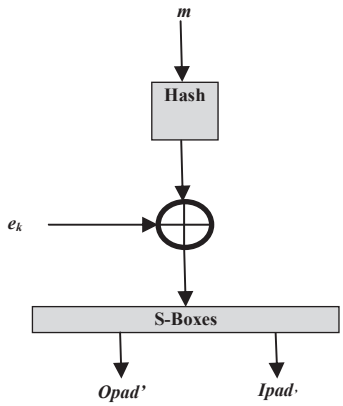

Figure 3. Dynamic *ipad'* and *opad'* generation

Algorithm 1 represents the methodology for calculating the dynamic values of *ipad'* and *opad'*. Such a calculation depends on three parameters in d-HMAC: the message *m*, the *S*-Box table, and the public key with length n from receiver e_R. Improved d-HMAC can use any type of hash function with different message digest lengths. Further, it can use any *S*-Box table with sufficient nonlinear characteristics. In this research, we focus on using the SHA-256 hash function with a 256-bit message digest, and on using table 1.

The following steps explain how *ipad'* and *opad'* are calculated in algorithm 1:

1. Calculate message digest $h(m)$ by using the SHA-256 hash function on *m*.
2. XOR message digest of message $h(m)$ with e_R.
3. Use *S*-Box to calculate *ipad'* and *opad'*.

The *S*-Box [16] was created to maintain balancing properties and nonlinearity. The *S*-Box's input and output both contain four bits. To explain *S*-Box calculations, we will review the following example, in which *S*-Box input = b_1, b_2, b_3, and b_4:

1. Concatenate $x = (b_1||b_2||b_3||\ b_4)$.
2. Calculate *y*, where y is the complement of *x*.
3. Transform the 4-bit *x* into a decimal value from 0 to 15.
4. Select *s* and *v* from T, where $s = T[x]$ and $v = T[y]$.
5. Transform *s* to *z* and *v* to *w*, where *z* and *w* both contain 4 bits.
6. Concatenate *z* to *A* and concatenate *w* to *B*.
7. Repeat steps 1 through 6 to reach *n* bits.
8. Save the final value of *A* to *ipad* and *B* to *opad*.

In algorithm 1, we developed a calculation to preserve the maximum Hamming distance of *ipad* and *opad*.

The Hamming distance of two Boolean functions *f* and *g*, indicated by $d(f, g)$, can be calculated as follows:

$$d(f,g) = \sum_{x\in\{0,1\}^n} f(x) \oplus g(x).$$

Where $f, g: \sum n \rightarrow \sum$ are Boolean functions.

The Hamming distance for the two bit strings *ipad* and *opad* is calculated as following:

$$d(ipad', opad') = \sum_{x\in\{0,1\}^n} ipad' \oplus opad'$$

S-box

S-Box values, which are used to calculate *ipad'* and *opad'*, are introduced in Table 1. The two-dimensional table T (a) consists of four columns and two rows. Each integer in the T table can be transformed into 4-bit binary values. We have selected values 3, 5, 6, 9, 10, and 12, which can be presented in binary as [0011], [0101], [0110], [1001], [1010], and [1100], respectively.

We considered two main cryptographic criteria in the construction of the *S*-Box:

- **Balancing property**: The values above fulfill the balancing cryptographic criteria [18,19]. The balancing property is an important cryptographic criterion, in which the number of ones and zeros in a string *S* with *n* bits will both equal *n*/2; in other words, the string contains an equal number of ones and zeros.

- **Hamming distance**: The distribution of values in table 1 was generated to maximize the Hamming distance of *ipad'* and *opad'*.

The d-HMAC function calculation mentioned in Fig. 4 is as follows:

$$d - HMAC_K(m) = h\left(K^+ \oplus Opad', h\left(K^+ \oplus Ipad', m\right)\right) \qquad (2)$$

http://sites.google.com/site/ijcsis/
ISSN 1947-5500

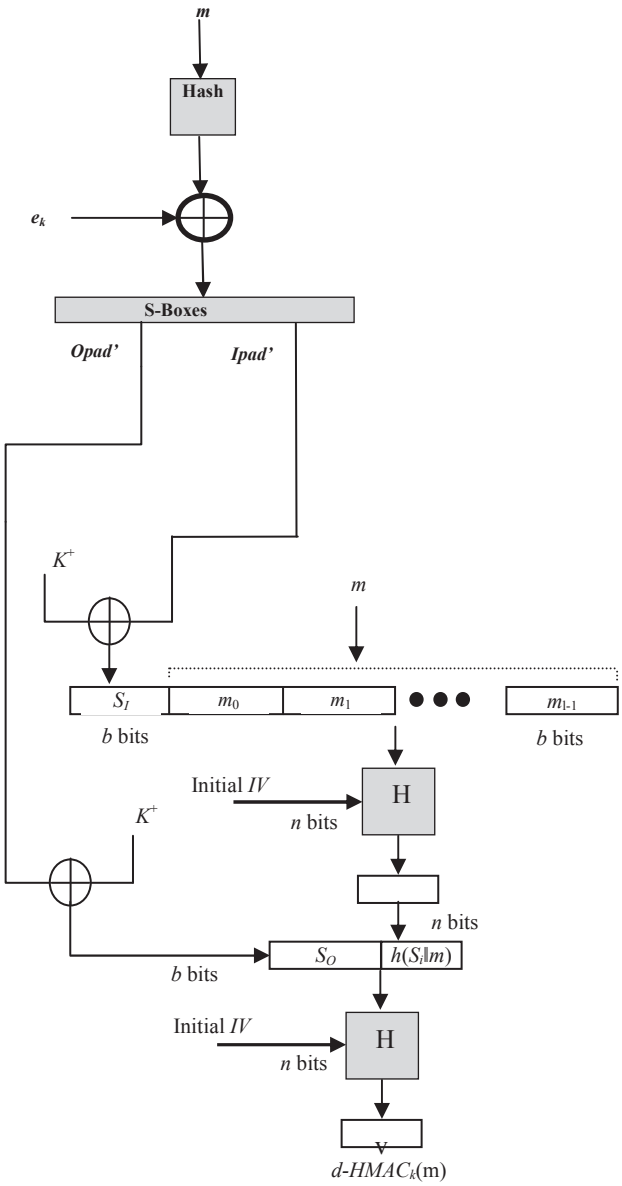

Figure 4. d-HMAC algorithm calculation

V. TEST RESULTS OF IMPROVED D-HMAC AGAINST HMAC

Improved d-HMAC is based on HMAC and it works in a similar manner, as mentioned in Section 3. These improvements are intended to enhance the security properties of HMAC, by increasing its resistance to different types of attacks. One of the advantages of improved d-HMAC is that it does not require any intermediate values to be stored, because these values are dynamically calculated for different recipients and different messages. Therefore, there is no need to implement a technique to protect these values from unauthorized disclosure.

Furthermore, improved d-HMAC does not require key frequent changes, because a random initial value is generated dynamically according to the recipient and the content of the message. The improved d-HMAC tests conducted in this study revealed that the modifications performed on HMAC to create d-HMAC had no negative impact on HMAC's main cryptographic characteristics; in some cases, it improved them.

Three types of tests were conducted. First, we compared the speed of improved d-HMAC against HMAC; in this test, both d-HMAC and HMAC used the SHA-256 algorithm. Second, we compared the algorithms' performance in terms of the avalanche effect. Third, we compared the balancing property [19]. All results are presented in tables and charts. The algorithms in the tests were implemented using C# 2010, and executed on an Intel Core i7 2.10 GHz processor running Windows 7 in 64-bit mode.

The Hamming distance method is used to calculate the avalanche effect and balancing properties by using XOR operations performed on the output bit string. The Hamming distance d for messages m_1 and m_2, in cases in which they are the same size n, can be calculated straightforwardly by counting ones in the set bits of $m_1 \oplus m_2$ and $0 < d < n$. The Hamming distance for m_1 and m_2 equals zero in cases in which $m_1 = m_2$, because there is no difference between m_1 and m_2 at the bit level. The Hamming distance for m_1 and m_2, where m_1 is a complement for m_2, equals n.

A. Speed test

As expected, d-HMAC required approximately twice the processing time as HMAC, because of the dynamic calculations for *ipad* and *opad* using the original message *m* (Roberts, 2008). These tests were conducted using files ranging in size from 1 MB to 100 MB; on our system, file sizes smaller than 1 MB generated processing times of approximately 62 ms and less for HMAC and 107 ms for d-HMAC, including hard drive access time (Table 2).

TABLE II. SPEED TEST FOR D-HMAC AND HMAC

File size (MB)	HMAC (s)	d-HMAC (s)
1	0.062	0.107
5	0.189	0.443
20	0.901	1.886
50	1.844	3.634
100	3.210	6.169

B. Avalanche effect test

The Avalanche effect is one of the primary design objectives for hash functions and any cryptographic tool; if an input message or secret key is changed slightly, approximately half of the output bits will be changed. Approximately 10,000 random sample inputs were tested for HMAC and d-HMAC; each avalanche effect test result listed in Table 3 represents a change of one input bit. In addition, 10,000 random key samples were tested for HMAC and d-HMAC; in these tests, one bit in the key was changed each time. The results are presented in Table 4.

http://sites.google.com/site/ijcsis/
ISSN 1947-5500

TABLE III. AVALANCHE EFFECT RESULTS ACCORDING TO MESSAGE BIT CHANGING

XOR results	HMAC (bits)	d-HMAC (bits)
MIN	97	97
MAX	157	159
Average	128.003	127.956

TABLE IV. AVALANCHE EFFECT RESULTS ACCORDING TO KEY BIT CHANGING

XOR results	HMAC (bits)	d-HMAC (bits)
MIN	103	105
MAX	152	150
Average	127.775	128.295

C. *Balance of ones and zeros test*

Balancing is one of the primary design objectives for hash functions and cryptographic tools, in which the output bits contain approximately equal numbers of 1's and 0's. Ten thousand random input samples were tested for HMAC and d-HMAC; in these tests, one bit in the input string was changed each time (Table 5). Moreover, 10,000 random key samples were tested for HMAC and d-HMAC; in these tests, one bit in the key was changed each time. The test's results do not indicate significant differences between d-HMAC and HMAC; see Table 6.

TABLE V. BALANCE RESULTS ACCORDING TO MESSAGE BIT CHANGING

XOR results	HMAC (bits)	d-HMAC (bits)
MIN	98	98
MAX	162	157
Average	127.962	128.062

TABLE VI. BALANCE RESULTS ACCORDING TO KEY BIT CHANGING

XOR results	HMAC (bits)	d-HMAC (bits)
MIN	106	106
MAX	149	149
Average	127.829	127.899

From the d-HMAC versus HMAC test results, we can conclude that the modifications required to create the improved d-HMAC preserved useful cryptographic properties; in some tests involving balancing and the avalanche effect, improved d-HMAC appears to exhibit better cryptographic characteristics than HMAC. The only difference is speed, where the improved d-HMAC requires approximately twice as much time as HMAC to perform calculations, which can even be a plus in password storage technology.

VI. D-HMAC RESISTANCE AGAINST CRYPTOGRAPHIC ATTACKS

One of the known attacks that can be used against HMAC is to collect many hash results that generate the same secret key K; this allows an attacker to initiate an offline attack by estimating the secret key. In contrast, it is infeasible to collect distinct messages calculated by the improved d-HMAC algorithm using the same secret key, because *ipad'* and *opad'* are always dynamically generated. As a result, d-HMAC is more resistant against cryptographic attacks than HMAC. Moreover, the HMAC function must securely store intermediate values [2], while improved d-HMAC does not require any cryptographic policy or tool to store any intermediate values ($K^+ \oplus$ *ipad'*) and ($K^+ \oplus$ *opad'*), as these values are dynamically calculated.

Programmers using HMAC for authentication must establish a secure method of saving intermediate values, which can be a serious vulnerability. In improved d-HMAC, there is no requirement to store these intermediate values, because they are dynamically calculated.

Finally, improved d-HMAC preserves useful cryptographic characteristics. The tests results show that improved d-HMAC did not compromise the avalanche effect or balancing properties.

VII. CONCLUSIONS

Our tests proved that using dynamically calculated *ipad* and *opad* values increases d-HMAC's resistance to attacks [13], and does not negatively affect the avalanche effect or balancing. The dynamic *ipad* and *opad* values used in d-HMAC enable a sender to send as many messages as required to any number of recipients using the same key, and eliminate the possibility of an attacker collecting the message digests *ipad* and *opad* will be different, depending on the recipient. Furthermore, the sender can also send distinct messages to the same recipient using the same key, because *ipad* and *opad* values are changed with each calculation for these different messages. We also improved the *S*-box tables used in *ipad* and *opad* calculations by improving the cryptographic characteristics. Additionally, the tests showed that d-HMAC and HMAC have similar cryptographic characteristics.

We improved the d-HMAC function to be more resistant to brute-force attacks than HMAC. Additionally, the improved d-HMAC function uses *ipad* and *opad* values that are calculated dynamically, depending on several input parameters mentioned in this paper. This strategy can generate robust random strings that can be used in hash functions as initial values. As a result, SHA-256 and other hash functions having message digest sizes larger than 255 bits are more collision resistant to known attacks.

http://sites.google.com/site/ijcsis/
ISSN 1947-5500

The developments and changes we implemented did not compromise cryptographic criteria such as the avalanche effect and balancing properties; this was proven by our tests described in Section 5.

In future work, we will expand the S-box table and improve its cryptographic characteristics. Furthermore, we will make the S-box dynamic, to more effectively control the algorithm's calculation speed, which can be helpful to use d-HMAC in as a password storage function.Authors and Affiliations

REFERENCES

[1] Najjar, M. and Najjar, F., "d-HMAC Dynamic HMAC Function. Dependability of Computer Systems", DepCos-RELCOMEX '06. International Conference on, 119-126, DOI: 10.1109/DEPCOS-RELCOMEX, 2006.

[2] Krawczyk, H., Bellare, M., Canetti, R., "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, 1997.

[3] Najjar, M., "Petra-r Cryptographic Hash Functions", International Conference On Security and Management SAM '03, Las Vegas, USA, Part I, 2003, 253-259.

[4] National Institute of Standards and Technology FIPS PUB 198, "The Keyed-Hash Message Authentication Code (HMAC)", Federal Information Processing Standards Publication, 2002.

[5] Hansen, T., "US Secure Hash Algorithms (SHA and HMAC-SHA)", RFC 4634, 2006.

[6] Tuner, S., Chen, L., "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, 2011.

[7] Wang, X., Yu, H., Wang, W., Zhang, H., and Zhan, T., "Cryptanalysis of HMAC/NMAC-MD5 and MD5-MAC". LNCS 5479. Advances in Cryptology - EUROCRYPT2009, DOI: 10.1007/978-3-642-01001-9_7, 2009.

[8] Stokłosa J., "Bezpieczeństwo danych w systemach informatycznych (Data Security in Information Systems)". Wydawnictwo Naukowe PWN, 2001.

[9] Stallings, W., "Cryptology and Network Security", Prentice Hall, New Jersey, ISBN 0-13141098-9, 2011, pp: 362-394.

[10] Preneel B., "Cryptographic primitives for information authentication – State of the art. Preneel B., Rijmen V. (eds.)", State of the Art in Applied Cryptography. LNCS 1528, Springer, Berlin, 1998, 49–104.

[11] Menezes A.J., van Oorschot P. C., Vanstone S.A., "Handbook of Applied Cryptography". CRC Press, Boca Raton, FL, 1997.

[12] Speirs, W. R. (II.), "Dynamic Cryptographic Hash Functions", UMI, 2007.

[13] Krawczyk, H., Bellare, M., Canetti, R., "Pseudorandom Functions Revisited: The Cascade Construction and its Concrete Security", Proceedings of the 37th Symposium on Foundations of Computer Science, IEEE, 1996, 514–523, DOI: 10.1109/SFCS.1996.548510.

[14] Tsudik, G., "Message authentication with one-way hash functions, Proceedings", INFOCOM'92, 1992, 22: 29–38, DOI: 10.1145/141809.141812.

[15] Schaad, J., Housley, R., "Wrapping a Hashed Message Authentication Code (HMAC) key with a Triple-Data Encryption Standard (DES) Key or an Advanced Encryption Standard (AES) Key", RFC 3537, 2003.

[16] Hussain, I., Shah, T., Gondal, M. A., Khan, M., and Khan, W. A., "Construction of New S-box using a Linear Fractional Transformation", World Applied Sciences Journal, 2011, 14 (12): 1779-1785, ISSN 1818-4952.

[17] ANSI x9.9, (revised 1986), American National Standard for Financial Institution Message Authentication (Wholesale) American Bankers Association, 1981.

[18] Olejar, D., Stanek, M., "On Cryptographic Properties of Random Boolean Functions", Journal of Universal Computer Science, Springer, 1998, 4: 705–717.

[19] Lloyd, S., "Counting Binary functions with certain cryptographic properties", Journal of Cryptology, 1992, 107–131, DOI: 10.1007/BF00193564.

[20] Roberts, D.,
http://www.powerbasic.com/support/pbforums/showthread.php?t=36862&highlight=d-hmac.

AUTHORS PROFILE

Mohannad Najjar received the B.S. and M.S. degrees in Computer Engineering from Poznan University of Technology, POLAND in 1998. Received Phd. In Telecommunication Engineering – Cryptography in 2002. During 2002-2009, he taught in Applied Science University- Amman, Jordan. Now he is the general manager on Vtech ltd. Company.

http://sites.google.com/site/ijcsis/
ISSN 1947-5500