Double-Hashing Operation Mode for Encryption

Sultan Almuhammadi, Ahmad Amro
College of Computer Sciences and Engineering
King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia
Emails: (muhamadi, g201404360)@kfupm.edu.sa

***Abstract*—Block ciphers, hash-based encryption, and public-key ciphers are examples of data encryption techniques with different desired features. Strong block ciphers, like AES, must run in some mode of operation to encrypt data larger than the block size. Public-key ciphers, like RSA, are costly and usually used for key-sharing rather than encrypting the data itself. Generic single-hash ciphers are less secure than block ciphers and vulnerable to some cryptanalysis techniques. In this paper, we review some cryptanalysis techniques on AES and some hash-based cryptosystems. We propose a new encryption scheme capable of encrypting big data and supporting both symmetric and asymmetric-key handling. We discuss its security strength against known cryptanalysis attacks. Unlike existing hash-based ciphers, the proposed scheme uses double hashing instead of a single hash function. With double hashing, the proposed scheme totally eliminates the threat of known cryptanalysis attacks and provides a highly secure stream ciphering scheme by its new design.**

Keywords*—Block ciphers, AES, stream ciphers, public-key, hash functions.

I. Introduction

Data encryption provides confidentiality as one of the most important goals of data security. A well-know block cipher adopted by NIST in 2001 is the Advanced Encryption Standard (AES) [1]. Since then, AES has been widely used as standard in all file encryption and network protocols. An earlier Data Encryption Standard (DES) has been used since 1977 which suffers many security issues due to its short key size which could not stand the technology advancement [2]. Other versions of DES, like 2DES and 3DES, have used to overcome some of the security issues without much of success [3]. Since its adoption by NIST, AES has replaced all versions of DES in sensitive applications and proved that it is more secure mainly due to its larger block size and key size for almost two decades. Block ciphers, like AES and DES, typically run in some mode of operation. Secure hash functions are commonly used in these modes of operation to handle data larger than their limited block sizes. Different modes of operations provide different desired security features [4]. Recent research shows that the block size of AES can be targeted by several cryptanalysis techniques [5], [6], [7]. With today's improving technology and availability of high computational power, more sophisticated attacks on AES become feasible. Therefore, a new encryption algorithm is needed to replace AES in the near future.

Secure hashing was used initially to provide modes of operation with some desired features for block ciphers [8]. In some cryptosystems, hash-functions have been used for encrypting the data directly without block ciphers [9], [10]. This makes hash-based encryption a good candidate to replace block ciphers. However, hash-based encryption ciphers should be carefully designed to avoid possible vulnerability to some cryptanalysis techniques.

This paper proposes a new hash-based encryption scheme with a built-in mode of operation using double hashing. We call it DHOME (Double-Hashing Operation Mode for Encryption). The built-in mode of operation in DHOME makes it capable of encrypting big data without block size restrictions. The encryption key can be handled as either symmetric or asymmetric-key without much of changes in the encryption scheme itself or the encrypted data. Moreover, if a symmetric key is used in DHOME to encrypt data, the ciphertext can be later decrypted using an asymmetric decryption key. This can be achieved by just a small modification of the header of the ciphertext without changing the encrypted data itself. The idea of the proposed scheme, DHOME, is to use two hash-functions instead of a single hash function in existing hash-based ciphers. With double hashing, the proposed scheme is more secure against known cryptanalysis attacks than generic single-hash ciphers. Moreover, DHOME is designed to provide a secure stream ciphering scheme.

II. Background

The main difference between block ciphers and stream ciphers is defined by Menezes et al. in Handbook of Applied Cryptography [11]. Stream ciphers encrypt individual characters (usually binary digits) of a plaintext one at a time, using an encryption transformation which varies with time. By contrast, block ciphers tend to simultaneously encrypt groups of characters of a plaintext message using a fixed encryption transformation. Similar comparison was mentioned by R. Rueppel in [12]. Block ciphers operate with a fixed transformation on large blocks of plaintext data. Where stream ciphers operate with a time-varying transformation on individual plaintext digits.

Symmetric-key ciphers, like AES, are known for their performance, but it is difficult to achieve secure key negotiation using only symmetric ciphers in network environments where the keys are usually negotiated over insecure channels. Therefore, an asymmetric-key cipher, like RSA, is typically used to send the shared key that is later used in the symmetric-key cipher. Hence, a combination of both symmetric and asymmetric ciphers becomes the convention approach [13].

In this section, we introduce some of the effective attacks on AES, and discuss the security issues of hash functions.

978-1-5090-4228-9/17/$31.00 ©2017 IEEE

A. AES Attacks

AES is a strong block cipher of size 128 bits, which doubles the size of its predecessor DES. Since its adoption by NIST in 2001, AES has been a target for cryptanalysis for several years. Two cryptanalysis attacks on AES with improving results are the biclique attack and related-key attacks. With biclique attack a key can be recovered faster than brute-force with computational complexity of $2^{126.1}$, $2^{189.7}$ and $2^{254.4}$ for AES-128, AES-192, and AES-256 respectively [14]. While with related-key attacks a key can be recovered with overall time complexity of 2^{131} and memory complexity of 2^{65} [15].

Other attacks are gaining progress in targeting reduced versions of AES, more rounds are being attacked each time. An attack mentioned in [16] targeted the 8-Round AES-192, and AES-256 and gained about 1 million times faster attack than exhaustive search with $1/32000$ reduced data complexity. The best attack on 9-round AES-256 is proposed in [17]. Building on the later attack, the first attack on 10-round AES-256 was proposed in [18]. The authors claimed the attack can be achieved with data complexity of 2^{111} chosen plaintexts, 2^{253} time complexity, and $2^{11.2}$ AES blocks for memory complexity. The previous three attacks suggest a gradual progress in reaching a successful attack on full AES in the coming years.

Recently a new set of flush-and-reload attacks on AES where proposed in [5], [6], and [7]. The flush-and-reload attacks basically takes advantage of the resource sharing feature (memory, CPU, etc.) in virtualization environments to capture information leaks from VMs to the host. All these attacks take advantage of the small block size of AES (128 bits). In [5] three attack scenarios were tested, one of them is claimed to recover the entire AES-128 key in only 15 seconds. In [7] the authors claimed only 6-7 plaintext or ciphertext blocks can lead to the recovery of the entire 128-bit AES key. While in [6], an espionage network is setup to calculate AES keys based on caching information leakage. The authors claimed that their setup can recover the encryption key in less than 30 encryption operations.

A successful power-based side channel attack was conducted in [19]. The authors targeted the software implementation of the ATmega328 microcontrollers. The attack basically takes advantage of power related measurements to leak information about ongoing computations on the chip. The attack time complexity is different from one key to another. For example, one key might require the collection of 600 traces for the encryption for a specific plaintext, while another key might require 300 traces. In general, to collect 100 traces it takes 30 minutes and few more to process.

B. Hash Functions Security

In [20] the authors targeted the SHA-512 hash function, they implemented and improved guess-and-determine approach trying to get differential characteristics and colliding message pairs. The improvements enabled the authors to enhance the semi-free-start collision attack on SHA-512 from 24 steps to 38 steps. Despite this improvement on the attack, it is still not efficient to extend it to fully expose SHA-512, due to the increased word size.

Applying hash function iteration to increase security has been used for password-based encryption rather than data encryption. A password-based encryption method is mentioned in [21], where combinations of passwords and salts are used with some iterative count to repeat the process of key generation in a way that makes exhaustive search infeasible. The difference between data encryption using our proposed method and the method of applying hash function iteration in [21] is that in our proposed method a combination of two different hash functions is used without repetition to defy known-plaintext attacks rather than iterating the same hash function to increase the secrecy of a password.

Hash functions have been used as part of newly suggested encryption algorithms. A new chaotic image encryption algorithm is proposed in [22]. The algorithm implements a hash function (SHA-1) to generate a set of hash keys that are used as initial keys in further operations to provide diffusion and permutation. In [23], the authors implemented a Toeplitz hash value to generate a pseudorandom key stream for transmitting ECG signals securely in medical applications.

III. Analysis of Existing Hash-Based Ciphers

Encryption using hash function has been possible since the starting of secure hashing functions. Many of these approaches use a single hash function to generate pseudorandom bits, which can be used as keys for encryption and decryption [9], [10], [21], [22], [23]. In this section, we explain the general framework of single-hash ciphers, and elaborate on their vulnerability against known cryptanalysis techniques. Some single-hash ciphers are more sophisticated, such as [9] and [10]. We will show possible cryptanalysis attacks on these ciphers as well.

A. Generic Single-Hash Cipher Framework

A hash-based cipher can be built in general with a single hash function. The general framework of a single hash cipher is shown in Figure 1. Basically, the scheme hashes the main key repeatedly to generate a key stream $(k_1, k_2, \ldots k_n)$. The scheme encrypts each segment p_i of the plaintext by XOR-ing it with the key stream segment k_i to generate the corresponding ciphertext segments $c_i = p_i \oplus k_i$, for $i = 1, \ldots, n$.

We claim that any hash-based cipher built with a single hash function according to this general framework is vulnerable to known-plaintext attack. Suppose the adversary knows some segment of the the plaintext, say p_j, then the adversary can compute $k_j = c_j \oplus p_j$ and decipher all the segments after p_j by Algorithm 1. Thus, if the adversary knows the header of the file, p_1, then the entire plaintext will be compromised.

Algorithm 1 Known-Plaintext Attack on Generic Single-Hash Ciphers

Input: known-plaintext segment (p_j, c_j)
Output: decrypted segments $p_{j+1}, p_{j+2}, ..., p_n$
$k_j \leftarrow c_j \oplus p_j$
for $i = j + 1$ to n **do**
  $k_i \leftarrow h(k_{i-1})$
  $p_i \leftarrow c_i \oplus k_i$
end for

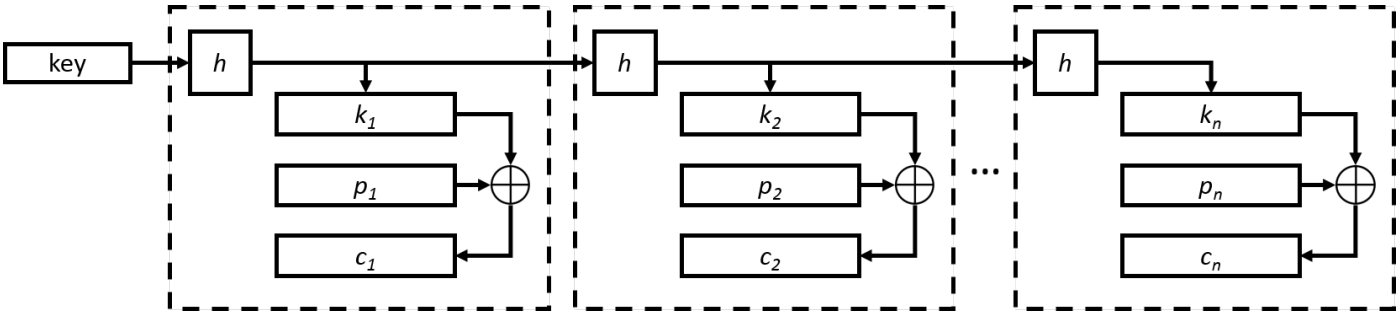

Fig. 1. General Single-Hash Cipher framework

B. Message Encryption Algorithm

Single hash ciphers can be more sophisticated and yet can still be attacked. For example, a patent was issued for a message encryption scheme that applies a single-hash function [9]. This approach has two different methods. In the first method, the first part of a message is being encrypted using a key generated by the hash of the concatenation of a secret key and an initialization vector which is not a secret, and the next key will be the hashing of the concatenation of both the previous key and the secret key, and so on. In the second method, the output of the encryption process will be used to generate the next key segment. We will discuss next some security issues in this approach and possible attacks.

The encryption algorithm requires a secret key K and an initialization vector IV, which is chosen at random at each time. In the first method of encryption, both K and IV are used to generate a series of keys k_i for $i = 1, 2, ..., n$ where n is the number of plaintext blocks. The first key k_1 is calculated using Equation 1.

$$k_1 = hash(K, IV) \tag{1}$$

While the remaining keys, k_i for $i = 2, ..., n$, are calculated using Equation 2.

$$k_i = hash(K, k_{i-1}) \tag{2}$$

Then, each ciphertext block c_i is calculated from a plaintext block p_i using Equation 3, for $i = 1, 2, ..., n$.

$$c_i = k_i \oplus p_i \tag{3}$$

The message encryption algorithm presented here is clearly more secure than the generic single-hash cipher. We cannot apply Algorithm 1 directly on this scheme since the main key K is used at every step of generating the key stream. Moreover, different initialization victors IV's generate different key streams even if the same key K is used. However, there are two possible vulnerabilities on this scheme.

1) Known-Plaintext Attack: It is important to observe that IV is sent in clear before the encryption takes place. So, an adversary can notice when the same IV is repeated. If the adversary has access to a known pair of plaintext/ciphertext, the key streams $(k_1, k_2, \cdots)$ can be calculated using Equation 4.

$$k_i = c_i \oplus p_i \tag{4}$$

Later if the adversary intercepts a ciphertext that is known to be encrypted using the same secret key K and the same IV, the plaintext can be obtained using Equation 5, for $i = 1, 2, ..., n$.

$$p_i = c_i \oplus k_i \tag{5}$$

Notice that the IV is being chosen at random. So, unless the period of the random generator is large enough to make any repetition for both the key and the IV together highly unlikely, the previous attack is feasible. For example, in a high speed network where each message is encrypted alone, if the random generator produces 32-bit IV's, some of them will repeat in less than few seconds according to the birthday problem, which makes the scheme vulnerable to known-plaintext attack.

2) Chosen-Ciphertext Attack: The first key segment is the output of hashing K concatenated with IV. The first block of the plaintext is calculated by XOR-ing the hash output with the first block of the ciphertext. If an attacker has access to the decryption device, he can feed it with a null ciphertext (a binary value with all zeros) and a null IV (empty string). When this null ciphertext is decrypted, the first block of the output would be the hash of the secret key $k_1 = hash(K)$. This information by itself does not mean that the cryptosystem is broken, but it gives the attacker some useful information about the key. Thus, if the hash function is not secure, the attacker can recover the key. The patent for this scheme was issued in 1996 and it recommended the use of the MD4 hash function which is not secure anymore [24], [25].

C. Image Encryption Algorithm

Another example of a relatively new single-hash cipher is the image encryption algorithm presented [10]. In this cipher, a secret key is hashed into a corresponding SHA1 hash value and converted to a binary form. Then the wavelet transform for the image is calculated and converted into binary string too. The key is expanded to fit the size of the image and XOR-ed bit-wise with the image string to create the encrypted image.

There is a weakness in this scheme that makes it vulnerable to known-plaintext attack. The encryption algorithm takes a secret key K and a plain image P as input and generates an encrypted image C as an output. The secret key K will be hashed using SHA1 hash function. The output of the hash will then be expanded to match the size of the plain image. The authors did not specify how the key was expanded. Let us assume that some expansion function F_{exp} is used, and the result of F_{exp} is the encryption key K_{enc} as shown in Equation 6.

$$K_{enc} = F_{exp}(\text{SHA1}(K)) \tag{6}$$

The plain image P will be XOR-ed with the encryption key K_{enc} to produce the encrypted image C using Equation 7.

$$C = P \oplus K_{enc} \tag{7}$$

Using known-plaintext attack, a similar method to Algorithm 1 can be applied to compute the encrypted key K_{enc} and recover the original image. Suppose an adversary has a known pair of the image P_0 and the encrypted image C_0, he can calculate the encryption key K_{enc} using Equation 8, which will always be the same for the same secret key K.

$$K_{enc} = P_0 \oplus C_0 \tag{8}$$

Later if the adversary intercepts an encrypted image C that is known to be encrypted using the same secret key K. The original image can be obtained using Equation 9.

$$P = K_{enc} \oplus C \tag{9}$$

This attack is highly likely to occur since the shared secret key K is not periodically changing, which implies it always generates the same encryption key K_{enc}. Using this attack, the adversary can record all transactions between two communicating entities, and upon acquiring a pair of plain/encrypted images, all other encrypted images can be easily decrypted without even knowing the secret key.

IV. A New Encryption Scheme

Given the potential attacks on AES presented in Section II-A, and the cryptanalysis of single-hash ciphers explained in Section III, there is a rapidly growing need for a more secure encryption algorithm. We propose a new encryption scheme DHOME, which stands for Double-Hashing Operation Mode for Encryption. The proposed scheme is more secure than other hash-based ciphers, and with a built-in mode of operation. Unlike many of the existing hash-based encryption schemes, the proposed algorithm uses two secure hashing functions to generate a pseudorandom sequence of bits that can be used as a key to encrypt the plaintext P in a simple XOR operation. We show that by applying double hashing, we create more confusion and diffusion that ensure the security of the proposed scheme. Moreover, the proposed scheme can be used with symmetric or asymmetric-key.

In our implementation, we assume the plaintext is given in some input file and the ciphertext is generated and saved in an output file. However, the design of DHOME is independent of this assumption. Thus, these input and output files can be replaced by any two parties, like sender and receiver in network communication. In this section, we explain the proposed scheme according to its design.

Figure 2 shows the block diagram of the proposed encryption scheme. It uses two hash functions, h and f. The plaintext is processed in segments, $P = p_1, p_2, \ldots, p_n$, each of length equals to the output length of f. The ciphertext is also computed in segments, $C = c_1, c_2, \ldots, c_n$, where each segment c_i of the ciphertext is corresponding to one segment of the plaintext p_i, plus an additional header of the ciphertext c_0. We recommend the use of two secure hash functions, namely: SHA-384 and SHA-512, for h and f respectively, as explained in Section IV-D. Therefore, each plaintext/ciphertext segment is 512 bits. The last segment p_n can have less than 512 bits without padding and it is XOR-ed with same number of bits of the key segment k_n. The rest of k_n can be truncated.

A. Encryption Algorithm

The encryption algorithm of DHOME runs as follows. First, it generates a sufficiently large random integer $seed$ of at least 512 bits to be used in generating the key stream K. The scheme encrypts the $seed$ and stores the encrypted seed in the header c_0.

Then, the $seed$ is hashed using a secure hash function h and the output of the hash function, $h_1 = h(seed)$, is hashed repeatedly to generate a sequence of n pseudorandom values $H = h_1, h_2, ..., h_n$, where $h_i = h(h_{i-1})$, and n is the number of the plaintext/ciphertext segments. This sequence is later used to generate the key stream $K = k_1, k_2, ..., k_n$.

To resist known-plaintext and chosen-ciphertext attacks, the relation between the key stream K and the pseudorandom values H should be hidden. We achieve this by *double-hashing*. Thus, we apply another secure hash function f on the pseudorandom values H to obtain the key stream, i.e. $k_i = f(h_i)$ for $i = 1, 2, ..., n$. The key stream K is then used to encrypt the plaintext by XOR operation. So, each segment of the resultant ciphertext is computed by $c_i = p_i \oplus k_i$. The steps of the encryption algorithm are outlined in Algorithm 2.

Algorithm 2 Encryption Algorithm

Input: plaintext $P = p_1, p_2, ..., p_n$; key
Output: ciphertext $C = c_1, c_2, ..., c_n$; $c_0 = Seed_{enc}$
$seed \leftarrow Large\_random\_number$
$Seed_{enc} \leftarrow Encrypt(key, seed)$
$h_0 \leftarrow seed$
for $i = 1$ to n **do**
  $h_i \leftarrow h(h_{i-1})$
  $k_i \leftarrow f(h_i)$
  $c_i \leftarrow p_i \oplus k_i$
end for

B. Symmetric and Asymmetric-key Options

There are two key options allowed by the proposed scheme: (a) Symmetric-key Mode, and (b) Asymmetric-key Mode, as shown in Figure 3. In the Symmetric Mode, the scheme can be used as a symmetric-key cipher by applying the shared symmetric secret key with any secure symmetric cipher at both sides (sender and receiver). While in the Asymmetric Mode, the scheme can be used as asymmetric-key cipher by applying two different keys (public and private) on any secure asymmetric-key cipher.

It is important to note that the symmetric and asymmetric ciphers are only used in DHOME to encrypt/decrypt the $seed$, not the data itself. Therefore, the high cost associated with the asymmetric cipher and the overhead associated with the mode of the operation of the symmetric cipher are not effecting the cost and the performance of DHOME.

Moreover, if DHOME is used in Symmetric Mode to encrypt big data, then later the users want to use Asymmetric Mode, the only part needs to be replaced is the header of the

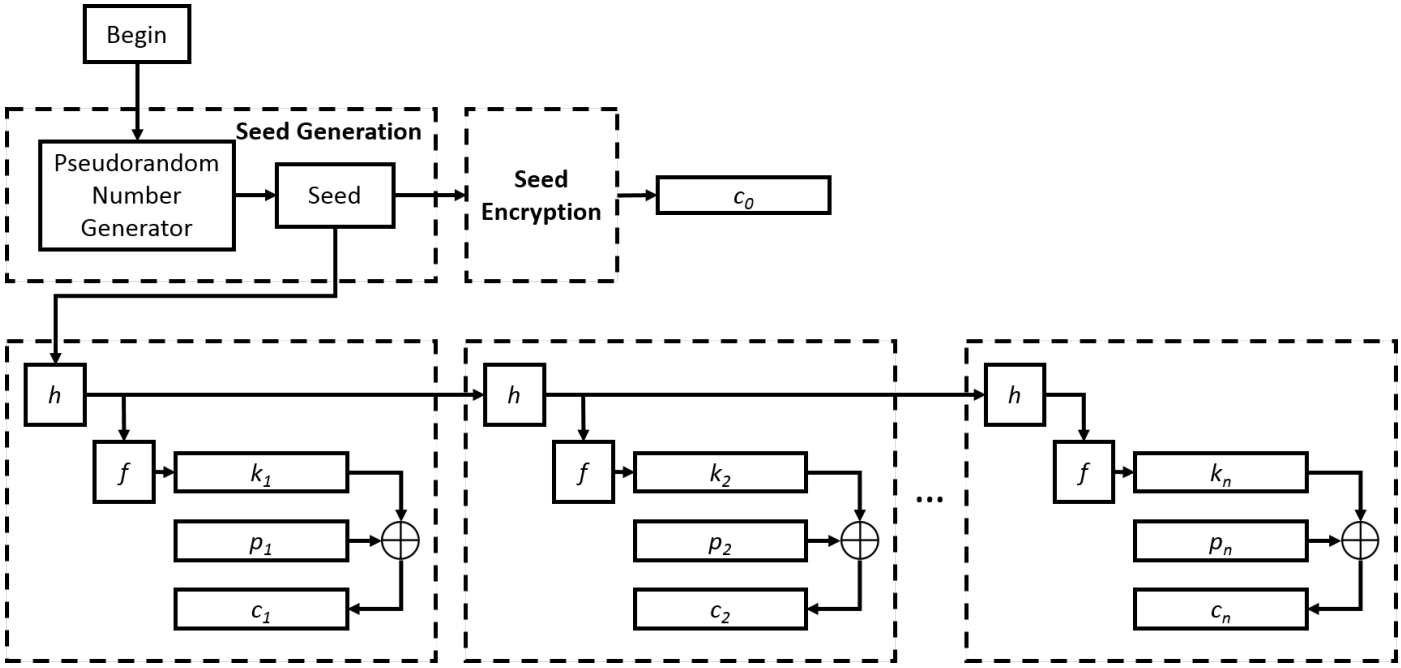

Fig. 2. Block Diagram of DHOME

ciphertext c_0. The sender can encrypt the $seed$ using the public key of the receiver and send the new header c_0 with the same ciphtertext C, without re-enctypting the whole data.

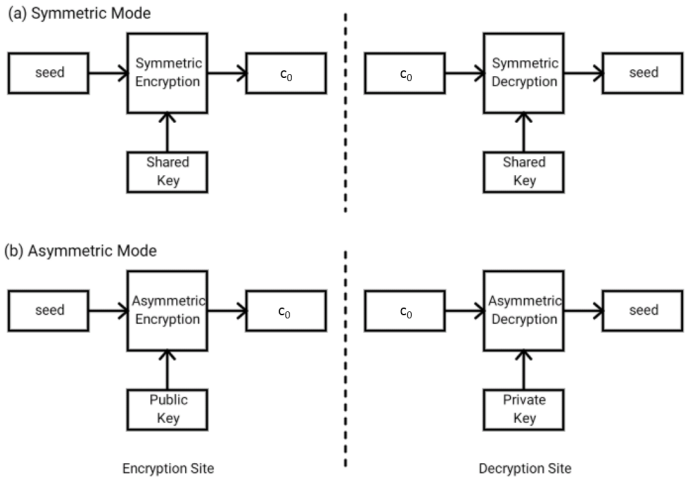

Fig. 3. Seed Encryption Modes

C. Security Analysis

Discussing the security of the proposed scheme can be made through discussing the design issues of DHOME that make the new scheme resist the cryptanalysis attacks on single-hash ciphers, namely: known-plaintext and chosen-ciphertext attacks. We show that DHOME is highly secure against these attacks.

1) Known-Plaintext Attack Proof: In known-plaintext attacks, the adversary has access to a pair of plaintext/ciphertext and wants to either compute the key or decipher another ciphertext. What makes the proposed scheme resilient against such attack is the application of double-hashing. Unlike single-hash ciphers, the proposed scheme uses two hash functions. Therefore, if an adversary has access to a known plaintext segment (p_i, c_i), only the corresponding key segment k_i is compromised by Equation 4. The adversary cannot compute any other key segment k_j for $j \neq i$ since this requires "un-hashing" of f to compute $h_i = f^{-1}(k_i)$, which is equivalent to breaking SHA-512.

Moreover, suppose the adversary has access to a whole plaintext/ciphertext pair (P, C). Even though the whole key stream will be compromised by Equation 4, both the main key and the seed remains secure. The key stream is just a random sequence of bits that is used to compute C from P, or P from C, nothing more. This helps avoiding the security issue that exists in [10]. Even if the main key is not changed, each encryption process has a unique key stream due to the application of a large random seed at the beginning of each encryption process. Therefore, the subsequent hashes initiated with the random seed will produce a different key stream every time. Also, to avoid the possibility of repeating the same key with the same value of the seed, it is a requirement to implement a random generator of output size 512-bit or more. This requirement was not mentioned in [9].

2) Chosen-Ciphertext Attack Proof: In chosen-ciphertext attacks, an adversary has access to the decryption device without knowledge of the embedded key. So the adversary can apply the decryption algorithm on some input ciphertext of his choice and recover the output plaintext in the hope of exposing the key. Unlike the message encryption scheme mentioned in [9], this attack has no effect against this approach since the embedded key will only be used to decrypt the seed. The key has no effect on the output plaintext. Also, the value of the seed cannot be derived from the plaintext due to the application of not only one, but two secure hash functions.

D. Sensitivity Testing

The security aspects of a group of known hash functions were evaluated using sensitivity tests. The targeted hash functions are: SHA-512, SHA-384, SHA-256, MD5 and MD4. The randomness of a hash function output is a key measurement of its security. To acquire accurate measurements of the randomness of each targeted hash function, an intensive sensitivity test was conducted. The idea here is to see how many output bits may change if a single input bit is changed. Each hash function was tested using the sensitivity test shown in Algorithm 3.

To test a hash function h of output length κ bits, the algorithm generates r random strings of length κ. For each random string w, the hash value h_w is computed. Then a single bit in w is flipped, and a new hash value $h_{\hat{w}}$ is computed. These two hash values are compared to each other and the percentage of their *Hamming distance* to their length is recorded in the sensitivity matrix. The location of the flipped bit varies from 1 to κ for each tested hash function. The algorithm computes and returns the sensitivity matrix, where $Sensitivity[i, j]$ indicates the percentage of the change in the output of hashing the j^{th} random string when a single input bit at location i is flipped. The ideal sensitivity score is 50%.

Algorithm 3 Sensitivity Test

Input: hash output length (bits) κ
number of random strings r
Output: sensitivity matrix $Sensitivity[\kappa, r]$

for $i = 1$ to κ **do**
  for $j = 1$ to r **do**
    $w \leftarrow random\_string(\kappa)$
    $h_w \leftarrow h(w)$
    $\hat{w} \leftarrow bit\_flip(w, i)$
    $h_{\hat{w}} \leftarrow h(\hat{w})$
    $\delta \leftarrow Hamming\_distance(h_w, h_{\hat{w}})$
    $Sensitivity[i, j] \leftarrow (\delta/\kappa) \times 100$
  end for
end for

In the sensitivity test, we run the algorithm for $r = 100$ random strings and calculated the minimum, maximum and average sensitivity values for each bit location and for all the five hash functions. The results of all the five tests are shown in Figure 4. Each point (x, y) in Figure 4.a represents the average sensitivity score y of the bit at location x computed by Equation 10. While the points in Figures 4.b and 4.c represent the minimum and the maximum scores computed by Equations 11 and 12 respectively.

$$y = \sum_{j=1}^{r} Sensitivity[x, j]/r \tag{10}$$

$$y = \min_{\forall j}(Sensitivity[x, j]) \tag{11}$$

$$y = \max_{\forall j}(Sensitivity[x, j]) \tag{12}$$

Table I summarizes the sensitivity test results numerically. It shows the output length, the average, the minimum, and the maximum score ranges of all the five hash functions.

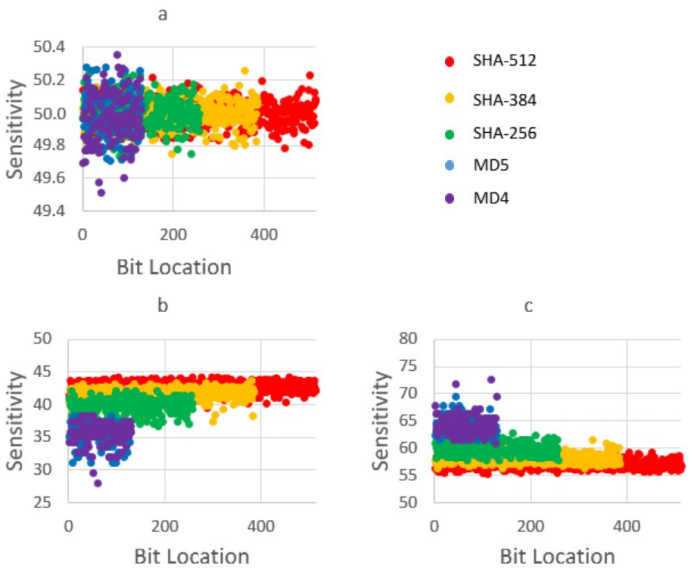

Fig. 4. Results of Hash Functions Sensitivity Scores (a) Average (b) Minimum (c) Maximum

TABLE I. NUMERICAL RESULTS OF THE SENSITIVITY TESTS

Function	κ (bits)	Average (%)	Minimum (%)	Maximum (%)
SHA-512	512	49.79 - 50.23	39.65 - 44.34	55.47 - 60.35
SHA-384	384	49.75 - 50.26	37.50 - 43.75	56.51 - 61.72
SHA-256	256	49.74 - 50.24	36.72 - 42.19	57.81 - 62.89
MD5	128	49.71 - 50.28	31.25 - 38.28	61.72 - 69.53
MD4	128	49.51 - 50.36	28.13 - 38.28	60.94 - 72.66

It can be noticed in Figure 4.a and the average column in Table I that all the hash functions provide good average sensitivity scores since all the average scores are close to 50% in almost all bit locations. For example, the average sensitivity scores of all the bits in SHA-512 are between 49.79% and 50.23% as shown in Table I, which are not far from the average scores of the bits in other hash functions, like MD4 and MD5. Therefore, the average sensitivity score does not reflect which hash function is better than the others.

However, as noticed in Figures 4.b and 4.c and the minimum and maximum columns in Table I, both the minimum and maximum scores clearly reflect which hash function is better than the others. SHA-512 scores are the closest to the ideal sensitivity score. Then SHA-384 comes in the second place and SHA-256 comes next. While both MD4 and MD5 scores are relatively far from the ideal score of 50%. These results suggest favouring the SHA family rather than the MD family as candidates for the implementation.

For DHOME implementation, the same hash function cannot be used in both h and f hashing devices. This will conflict with the double-hashing feature and allow known-plaintext attack. Moreover, we observed that using a second hash function f with a large output length will increase the plaintext segment length, which slightly speeds up the encryption and decryption processes and gives better overall performance. Therefore, we recommend using SHA-512 for f and SHA-384 for h in DHOME implementation.

V. Conclusion

In a quest aiming to provide a secure and efficient solution for maintaining data confidentiality, we proposed a new hash-based encryption scheme (DHOME) that is designed to avoid the security issues existed in single-hash ciphers. The security issues of the proposed scheme are discussed and compared to the security of some single-hash ciphers. The double-hashing design in DHOME makes it more secure than other existing hash-based ciphers. The built-in mode of operation in DHOME makes the encryption of big data a straightforward task without block size restrictions.

DHOME allows two modes of encryption. The encryption key can be handled as either symmetric or asymmetric-key without much of changes in the encryption scheme itself. Moreover, DHOME makes it easier to switch from symmetric to asymmetric-key and vice versa without changing the encrypted data itself. This can be achieved by just a small modification of the header of the ciphertext.

Furthermore, DHOME can be very useful in cloud data sharing with the advantage of the header of the ciphertext. Suppose some big data in the cloud is encrypted using DHOME. The user can share the data later by re-encrypting the seed and sharing the new ciphertext header without the need of encrypting the whole data or changing his secret or private key. Thus, the encrypted data stays as is in the cloud, only the seed is encrypted and shared as needed.

As for future work, DHOME can be utilized in cloud applications very effectively. The header of the ciphertext makes the scheme suitable for encrypting and sharing big data on the cloud with its simple and elegant key handling mechanism.

Acknowledgment

The authors would like to thank King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, for supporting this research. Figures and descriptions in this paper were provided by the authors and are used with permission.

References

[1] J. Daemen and V. Rijmen, *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[2] E. Biham and A. Shamir, *Differential cryptanalysis of the data encryption standard*. Springer Science & Business Media, 2012.

[3] O. P. Verma, R. Agarwal, D. Dafouti, and S. Tyagi, "Peformance analysis of data encryption algorithms," in *Electronics Computer Technology (ICECT), 2011 3rd International Conference on*, vol. 5. IEEE, 2011, pp. 399–403.

[4] M. Dworkin, "Recommendation for block cipher modes of operation. methods and techniques," DTIC Document, Tech. Rep., 2001.

[5] B. Gülmezoğlu, M. S. Inci, G. Irazoqui, T. Eisenbarth, and B. Sunar, "A faster and more realistic flush+ reload attack on aes," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2015, pp. 111–126.

[6] B. Roy, R. P. Giri, C. Ashokkumar, and B. Menezes, "Design and implementation of an espionage network for cache-based side channel attacks on aes," in *Proceedings of the 12th International Conference on Security and Cryptography*, 2015, pp. 441–447.

[7] C. Ashokkumar, M. B. S. Venkatesh, R. P. Giri, and B. Menezes, "Design, implementation and performance analysis of highly efficient algorithms for aes key retrieval in access-driven cache-based side channel attacks," 2016.

[8] J. Black and P. Rogaway, "A block-cipher mode of operation for parallelizable message authentication," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2002, pp. 384–397.

[9] C. W. Kaufman and R. J. Perlman, "Message encryption using a hash function," Jan. 9 1996, uS Patent 5,483,598.

[10] T. Bandyopadhyay, B. Bandyopadhyay, and B. Chatterji, "Secure image encryption through key hashing and wavelet transform techniques," *International Journal of emerging technology and Advanced engineering*, vol. 2, pp. 26–31, 2012.

[11] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.

[12] R. A. Rueppel, "Analysis and design of stream ciphers. communications and control engineering series," *Springer-Verlag Berlin Heidelberg*, 1986.

[13] S. Vandeven, "Ssl/tls: Whats under the hood," *SANS Institute InfoSec Reading Room*, vol. 13, 2013.

[14] A. Bogdanov, D. Khovratovich, and C. Rechberger, "Biclique cryptanalysis of the full aes," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2011, pp. 344–371.

[15] A. Biryukov, D. Khovratovich, and I. Nikolić, "Distinguisher and related-key attack on the full aes-256," in *Advances in Cryptology-CRYPTO 2009*. Springer, 2009, pp. 231–249.

[16] O. Dunkelman, N. Keller, and A. Shamir, "Improved single-key attacks on 8-round aes-192 and aes-256," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2010, pp. 158–176.

[17] L. Li, K. Jia, and X. Wang, "Improved single-key attacks on 9-round aes-192/256," in *International Workshop on Fast Software Encryption*. Springer, 2014, pp. 127–146.

[18] R. Li and C. Jin, "Meet-in-the-middle attacks on 10-round aes-256," *Designs, Codes and Cryptography*, pp. 1–13, 2015.

[19] U. Banerjee, L. Ho, and S. Koppula, "Power-based side-channel attack for aes key extraction on the atmega328 microcontroller," 2015.

[20] M. Eichlseder, F. Mendel, and M. Schläffer, "Branching heuristics in differential collision search with applications to sha-512," in *International Workshop on Fast Software Encryption*. Springer, 2014, pp. 473–488.

[21] B. Kaliski, "Pkcs# 5: Password-based cryptography specification version 2.0," 2000.

[22] T. Gopalakrishnan and S. Ramakrishnan, "Chaotic image encryption with hash keying as key generator," *IETE Journal of Research*, pp. 1–16, 2016.

[23] K. Pandian and K. C. Ray, "Dynamic hash key-based stream cipher for secure transmission of real time ecg signal," *Security and Communication Networks*, vol. 9, no. 17, pp. 4391–4402, 2016.

[24] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu, "Cryptanalysis of the hash functions md4 and ripemd," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2005, pp. 1–18.

[25] H. Dobbertin, "Cryptanalysis of md4," *Journal of Cryptology*, vol. 11, no. 4, pp. 253–271, 1998.