A-RSA: Augmented RSA

Abdallah Karakra and Ahmad Alsadeh
Faculty of Engineering and Technology
Birzeit University
P.O.Box 14-Birzeit, Palestine
Email:{akarakra,asadeh}@birzeit.edu

***Abstract*—Today, RSA algorithm is the most widely used public-key cryptosystem around the world. It is used for security in everything from online shopping to cell phones. However, the basic RSA is not semantically secure, i.e., encrypting the same message more than once always gives the same ciphertext. For this reason, the basic RSA is vulnerable to set of indirect attacks, such as *known plaintext*, *chosen plaintext*, *timing*, *common modulus*, and *frequency of blocks* (FOB) attacks. Moreover, RSA is known to be much slower than the standards symmetric key encryption and it does not used for encrypting large data. In this paper, we design and implement a swift and secure variant of RSA based on Rabin and Huffman coding called Augmented RSA (A-RSA) to solve aforementioned limitations of the basic RSA. A new additional randomization component r is added in A-RSA. This component is encrypted by *Rabin* algorithm to improve the security level of RSA against the *indirect attacks* and make RSA semantically secure. Moreover, A-RSA makes the factorization problem harder, since the attackers need to break the factorization of large numbers for both RSA and Rabin. Besides, employing Huffman Coding compression in A-RSA prevents *FOB* attack and speeds up the execution time for the A-RSA. Our testing results over set of file sizes of 1MB, 2MB, 3MB, to 10 MB show that *A-RSA*'s average execution time is equal to 0.55 of the average execution time of the basic RSA in encryption process and 0.01 in decryption process. Also, we found that RSA system increases the size of ciphertext by 1% compared to the original file size, while the average size of *A-RSA* files is equal 0.46 of its original sizes.**

***Keywords*—Public-key Cryptosystems; RSA; Rabin; Huffman Coding; Semantically Secure; RSA Attacks.**

I. Introduction

RSA was developed by **R**ivest, **S**hamir and **A**dleman in 1978 [1]. RSA uses two keys, private and public, where the private-key is kept secret and the public-key is published publicly. One key can be used for encryption and the other is used for decryption. RSA security is based on the hardness of factoring large prime integers. To date, it has been considered that factoring large prime integers is a hard problem, thus RSA is secure. Currently, RSA modulus should be 1024 bits long or greater to be considered secure.

RSA is the most widely used cryptosystem around the world [2], [3]. It is used in digital signatures, keys exchange and encrypting data. Many security protocols depends on RSA, such as Transport Layer Security (TLS) protocol which is used in Internet-based e-commerce [4]. In addition, it is used for protecting emails and traffic of the web, and it is used for securing some of the wireless devices and network resources [5]. Therefore, any leak in the RSA security will lead to security breaches in the Internet applications that rely on the RSA and make them vulnerable to the attacks.

In fact, RSA has some limitations and downsides. The use of RSA for encrypting large-scale data is a challenging issue, referable to the computational intensive characteristics of the RSA arithmetic operations. Diffie states that "the restriction of public-key cryptography to key management and signature applications is almost universally accepted" [6]. Since RSA is not practical and takes long time for encrypting and decrypting large data [7], the RSA usage is limited for authentication capabilities, or exchange keys between two parties, and then use another symmetric key for encrypting the entire data. In addition, basic RSA not semantically secure [8], which make the algorithm vulnerable to some types of indirect attacks, such as *known-plaintext-attack* (NPA), *chosen-plaintext attack*(CPA), *timing attack* (TA), *frequency of blocks* (FOB) and, *common modulus attack* (CMA). Moreover, encrypting data using RSA generates ciphertext greater than the original plaintext.

Different implementation and optimization methods have been suggested to enhance the execution time of RSA algorithm [5], [9], [10]. Some other approaches have been proposed to make RSA semantically secure, such as blinding the message during the encryption process [11].

In this paper, we suggest an enhanced variant of RSA called Augmented RSA (A-RSA). A-RSA is designed and implemented in away to boost three sensitive factors of RSA; encryption and decryption execution time, space, and security. We summarize our main contributions as follows:

1) Augment RSA cryptosystem to make it more secure against CMA, NPA, CPA, and TA; by adding a randomized component r to the basic RSA and encrypting this component by another public-key cryptosytem called *Rabin*. This makes RSA semantically secure against these attacks by generating different ciphertexts for the same message. Also, this makes RSA stronger against *brute force attack*, since the attackers need to break the factorization of large numbers for both RSA and Rabin. Consequently, the attackers will require longer time than before.
2) Thwart the FOB attack by using Huffman coding. Huffman coding compresses data in away to reduce the redundancy in the message, which helps to prevent this attack.
3) Enhance the execution time comparing with the basic RSA by using *Huffman coding*. Also, by encrypting part of the message, while blinding the other part of the same message using *XOR*-operator ($\oplus$), since $\oplus$ operator is always faster than multiplication, division and addition. Our experimental results show that *A-RSA* is faster than basic RSA by about 45% in

encryption process and around 99% in decryption process.

4) Reduce the sizes of large data by using Huffman coding make it feasible to use *A-RSA* for encrypting large files. *A-RSA* reduces the size of encrypted file by 54% from the original sizes. This reduction depends on the number of occurrences of the symbols inside the file. While RSA system increases the size of ciphertext by approximately 1% compared to the original file size.

This paper is organized as follows: Section II presents the important definitions and concepts of encryption methods that are related to the remaining sections in this paper. Also, we discuss the most important public-key cryptosystems. Section III discusses some of indirect attacks against RSA cryptosystem and their countermeasure. Section IV presents our approach to enhance RSA called *A-RSA* cryptosystem. Section V presents a security analysis of *A-RSA*. Section VI compares the performance of A-RSA with basic RSA. Finally, the conclusion is given in section VII.

II. Background

A powerful technique for protecting data is the use of cryptography. Cryptography is the science or the art for protecting data [12]. It enables us to store or to transmit sensitive data over insecure channels. The data is scrambled (encrypted) to be unreadable to anyone other than intended persons. Converting the encrypted data into its original is called decryption.

The cryptographic techniques can be classified into two groups based on the number of used keys in encryption and decryption processes; Symmetric (is known also as shared, conventional, secret-key, or single-key) encryption and asymmetric (public-key) encryption. Each of these techniques has its own characteristics and is used in a variant way based on the need.

A. Symmetric Key Encryption

Symmetric key encryption in which the sender and receiver share the same secret key (K) for encryption and decryption process. For example, when Alice wants to send Bob a message M, she encrypts M using K to produce the ciphertext such that $C = E(M, K)$, where E is the encryption function, and send C to Bob. He in role decrypts the ciphertext such as $M = D(C, K)$, where D is the decryption function, by using the same K to recover the original message M.

The security of symmetric encryption process depends on several factors. First, building strong encryption algorithm known to everyone, where no one can break it to figure out the secret key or to decipher the ciphertext to find the plaintext [13]. Second, the key must be long enough to avoid the possibility of finding the key through the *brute force* search. If the key is known to the adversary in anyway, then entire encryption process will be broken. Therefore, the key must be kept secret between the two authorized communicated parties (sender and receiver). However, sharing or exchanging the same key between the two communicated parties is not a trivial task.

The communicated parties may exchange the key physically. That is, Alice chooses the key and physically delivered to Bob. However, if there is a fare distance between them, exchanging keys physically is not a good solution, since how many minutes, hours or days one needs for exchanging the key physically. If the two parties want to change the key more than once, one of them should travel to other to deliver the new key. If there are more than two parties like three, four, or more, where each one of them in different countries, it is more difficult to do that. Thus they may look for another approach to exchange the key. As, at the first time they exchange the key physically, and then they use the recent key for encrypting the newly one. But if the attacker in somehow succeeds in gaining access to one key, then all of the subsequent keys are compromised, and the ciphertext will be disclosed.

B. Public-Key Encryption

Public-key encryption was introduced in 1976 by Diffie and Hellman to solve the key distribution problem [14]. Public-key encryption, also called asymmetric encryption, where one party has a secret key called *private-key* and the other party has a *public-key*. The private-key must be kept secret, while the public is published, so no need for the sender and receiver to share secret information, all communications involve only public-keys, and no private-keys are ever transmitted or shared. When Alice wants to send Bob a message M, she uses the Bob's public-key KU_B for encrypting M such that $C = E(M, KU_B)$, while Bob uses his private-key KR_B for decrypting the ciphertext C such that $M = D(C, KR_B)$ to recover the original message M. Therefore, there is no secret key exchanged between Alice and Bob.

Currently, there are many public-key algorithms, such as Diffie-Hellman [15], RSA [1], Rabin [16], ElGamal [17], Elliptic Curve Cryptography (ECC) [18], [19], and others.

Diffie-Hellman algorithm is used for exchanging a secret key between two parties securely, then they use the exchanged key to encrypt the subsequent messages using symmetric key encryption algorithm [20]. The security of Diffie-Hellman depends on the difficulty of computing discrete logarithms [14]. However, the algorithm itself suffers from the *man-in-the-middle attack*, since it does not authenticate the communicating parties [21]. ElGamal public-key system also depends on discrete logarithms of a large prime modulus [17]. It is closely related to the Diffie-Hellman technique. ElGamal is not deterministic algorithm [22], encrypting the same plaintext gives a different ciphertext, but unfortunately the algorithm has a disadvantage related to the message size, such as the size of the ciphertext is twice the size of the original message [17].

RSA is most widely accepted as trusted public-key cryptosystem, its security depends on the idea of the hardness of factoring large integers [23], [24]. However, the main disadvantage of current RSA encryption schemes is the computational overhead. Rabin encryption algorithm is a public-key algorithm, whose security similar to the RSA, based on the hardness of integer factorization. In [16], Michael Rabin proved that Rabin is more secure than RSA, since Rabin is hard as hard of integer factorization, which is not true for the RSA.

ECC is a public-key that provides equal protection to the RSA by using smaller key size [25]. Since the algorithm is new, the confidence level of using it is not yet as high as that

in RSA [13]. The critical disadvantages of ECC is significant increase in the size of the encrypted message comparing to the RSA encryption [26].

In this paper, we focus on RSA and Rabin algorithms since they are similar in some process like encryption. Also, the two algorithms depends on the integer factorization. But this is a suggested approach, Rabin can be replaced by another public-key cryptosystem like ECC.

For using RSA cryptosystem, the sender and the receiver need to have their own key pairs. Bob generates the RSA key pair (KU_B, KR_B). He publishes his public-key while keeping the private-key secret. If Alice wishes to use RSA for encrypting a message M and send it to Bob, she uses the Bob's public-key (KU_B) for encrypting M. Once Bob receives the encrypted message C, he uses his private-key for decrypting C and retrieving M. Algorithm 1 summarizes the RSA cryptosystem steps.

Algorithm 1 : Basic RSA Cryptosystem

Step 1: RSA Generating public/private key pair at the receiver side

1: Choose two distinct large prime integers p and q.
2: Compute $N = pq$.
3: Compute Euler's totient function $\phi(N) = (p-1)(q-1)$.
4: Choose an integer e (public exponent), such that $1<e<\phi(N)$ and gcd(e,$\phi(N)$)=1.
5: Find an integer d (private exponent), such as $ed = 1 \bmod \phi(N)$.
6: Publish the public-key (N,e) and keep d, p, q, and $\phi(N)$ secret.

Step 2: RSA encryption process at sender side

1: Encrypt M by computing $C = M^e \bmod N$, where $0<M<N-1$.

Step 3: RSA description process receiver side

1: Decrypt C to get the original message $M = C^d \bmod N$.

Rabin algorithm has been developed by Michael Rabin in 1979. The security of a Rabin algorithm is based on the difficulty of factoring large integers. Rabin algorithm is depicted in Algorithm 2. As Srivastava and Mathur state in [27], the main disadvantage of Rabin algorithm is the extra complexity required in decryption process for identifying the corresponding plaintext M from the four possible roots. That is, there are four possible output roots M_i where $i = 1, 2, 3, 4$ generated from the decryption process. Thus, we need extra time to know which M_i represents the original message M.To make it is easier to distinguish the correct message from the four roots, we pad the message before encryption. After decryption, only one from the four possible messages will contain that special padding characters.

After introducing RSA and Rabin cryptosystems, in the following section we present the indrect attacks against the basic RSA.

III. RSA Indirect Attacks

RSA encryption algorithm is secure as no one gets other than the public-key, otherwise the algorithm is not secure. For example, $(d, p, q, \phi(N))$ are the important four RSA parameters that form the RSA trap-door. If any one of these parameters are known, then the RSA will be compromised completely [11].

Algorithm 2 : Rabin Cryptosystem

Step 1: Generating public/private key pair at the receiver side

1: Choose two distinct large prime integers p and q in the form $4k + 3$.
2: Compute $n = pq$.
3: Publish the public-key (n) and Keep the private-key (p, q) secret.

Step 2: Rabin encryption process at sender side

1: Encrypt M by computing $C = M^2 \bmod n$, where $0<M<n-1$.

Step 3: Rabin description process receiver side

1: Decrypt C by using the private-key (p, q) as follows:
- Compute $R = C^{(p+1)/4} \bmod p$ and $S = C^{(q+1)/4} \bmod q$
- Find a, b such that $ap + bq = 1$
- Find $M_1 = (apS + bqR) \bmod n$, $M_2 = (n - M_1)$, $M_3 = (apS - bqR) \bmod n$ and $M_4 = (n - M_3)$
- Choose which M_i is the correct root (plaintext M), where $i = 1, 2, 3, 4$.

There are different possible approaches to attacking RSA cryptosystem. We can put them into two main categories; *direct* and *indirect* attacks. Direct attack includes *mathematical* and *brute-force* attacks. The mathematical attacks represented by the effort to factorize the product of two large prime integers, which is widely believed to be a hard problem. In the *brute-force* attack the attacker tries all possible combinations to find to the private-key. The defense against the brute-force attack is to use a large key space. In *indirect* attacks the attackers try to exploit the weaknesses of the algorithm implementation or depending on other information available to them to generate their attacks.

The basic RSA is a deterministic algorithm, which means the message has always the same encryption for same key. This property enable attackers to successfully launch many kinds of *indirect* attacks, such as *common modulus attack*, *known-plaintext attack*, *chosen-plaintext attack* and timing attack. This paper focus on the following *indirect* attacks, which are important for the remaining of this paper. The paper does not cover the *chosen-ciphertext attacks* (CCA) and the *Hardware fault-based attack* [28].

A. Common Modulus Attack (CMA)

Suppose a message M is encrypted twice by using RSA cryptosystem using the same modulus $N = pq$ with different public-keys (e_1, N) and (e_2, N) such that $gcd(e_1, e_2) = 1$. If the attacker know $C_1 = M^{e_1} \bmod N$ and $C_2 = M^{e_2} \bmod N$, then she can recover the original message M [11], [29]. For knowing e_1 and e_2, find two integers a and b such that $a \times e_1 + b \times e_2 = 1$ using the Extended Euclidean Algorithm. Then compute ${C_1}^a {C_2}^b \equiv M^{a \times e_1} M^{b \times e_2} \equiv M^{a \times e_1 + b \times e_2} \equiv M$. This implies that any party can obtain the public-keys and the corresponding ciphertexts could be capable to intercept all the

messages which would be encrypted twice to different users. One of the suggestions to protect RSA from *common modules attack* is to never use common modules in RSA [11].

B. Known-Plaintext Attack (KPA)

In this type of attack, the adversary may be able to capture a set of plaintexts with its corresponding ciphertexts, to build the set $S = \{(P_1, C_1), (P_2, C_2), ..., (P_i, C_i)\}$, where $P_i \in$ plaintext and $C_i \in$ ciphertext [13], [30]. The attacker can used any later captured messages encrypted with the same key to find the plaintext P_i if the corresponding C_i is in set S. Because the basic RSA is a deterministic algorithm, encrypting the same message more than once with the same key gives the same ciphertext. Based on the pre-built set S, the adversary can use any later captured data to find the plaintext P_{i+1} if the corresponding C_{i+1} is in the set S. Accordingly, the adversary who obtain partial of plaintext, can guess the other parts.

C. Chosen-Plaintext Attack (CPA)

This attack is similar to *known-plaintext attack* but the adversary has more power, The adversary chooses an arbitrary message M and in somehow she is able to insert the chooses M into the system to get the corresponding ciphertext [13]. Thus, the attacker can build a set of plaintexts-ciphertexts, such as $S = \{(P_1, C_1), (P_2, C_2), ..., (P_i, C_i)\}$, where $P_i \in$ plaintext and $C_i \in$ ciphertext.

D. Timing Attack (TA)

In this type of attack, the attacker obtains the information based on the implementation of the algorithm itself, without exploiting any weakness in the mathematical approach that the algorithm applied [31]. The attacker exploits the variance of the time in cryptographic operations. That is, the computations performed by a cryptographic algorithm takes a different amount of time based on the input and the value of the secret parameter in addition to the performance of the system that involved in this computation. If RSA private-key operations can be timed accurately, statistical analysis can be used to obtain the secret key involved in the computations. Kocher demonstrates this kind of attack in [32] by computing how long a computer takes to decrypt a message. Boneh [8] show that the attacker can recover the private-key d one bit at a time until the secret exponent d is known. One suggestion to prevent *timing attack* is to use random component as shown in Algorithm 3 [11].

Algorithm 3 : Anti-Timing attack

1: Generate a secret random number $r \in Z_N^*$.
2: Compute $C' = C \times r^e \bmod N$.
3: Compute $M' = C'^d \bmod N$.
4: Compute $M = M' \times r^{-1} \bmod N$.

E. Frequency of Blocks Attack (FOB)

RSA is a kind of block cipher cryptosystem even it is not intended to be used as a block cipher. RSA is typically used for encrypting small pieces of data, such as symmetric key that is then used for encrypting the entire message. Nevertheless, RSA cryptosystem works on block cipher manner where the message is divided into a number of blocks based on the size of the block. The block size can be chosen between 1 to $N-1$ for some N.

The basic RSA suffers from *Frequency of Blocks* (FOB) attack. When one of the blocks repeated within the same message, then the block has the ciphertext similar to that in the first block. The main cause of this problem return to the fact that basic RSA is deterministic algorithm. The following example shows how basic RSA is not secure against this attack. Suppose the block size is 127 bytes and the message size is 635 bytes, so the number of blocks are 5. If two blocks are repeated within the same message, each of them will have the same ciphertext. If the attacker in somehow know that the message contains repeated block, may exploit this weakness to find the plaintext. To the best of our knowledge, no one points to this type of attack in RSA. The reason may refer to the fact that RSA is usually used for key exchange and digital signature, instead of encrypting the majority of data.[1].

IV. Augmented RSA

A-RSA cryptosystem has two new additions besides the basic RSA and Rabin algorithms; Huffman coding and Random component r. The basic RSA and Rabin are explained in Section II. Here, we discuss the other two modifications in more details.

A. Huffman Coding in A-RSA Cryptosystem

According to the Simmons [34], cryptographers consider data compression algorithms as a ciphering scheme. Also, Shannon [35] suggested that reducing the redundancy in data before encryption, protect it against statistical analysis. Therefore, we use Huffman coding to achieve Simmons and Shannon suggestions.

Huffman coding is a common method for data compression [36]. It is an algorithm that uses for lossless data compression, where the original data can be recovered exactly from the compressed data. The algorithm is used to compress data (symbols or alphabet) to generate variable-length codes instead of fixed-length codes for each symbol. Given a set of symbols in a file, the algorithm performs some statistical analysis to construct a table that contains the frequencies of occurrence for each symbol. The algorithm uses the constructed frequency table to build Huffman tree, which is used to assign each symbol with its appropriate code length based on the symbol occurrence. The result of applying Huffman coding on the data file is two files; *binary file* (B) and *header file* (H). The binary file depends on the header file for retrieving the original data. Thus, if the header file is lost, then the true data cannot be retrieved. The header file contains all symbols of the original data file or its corresponding ASCII codes. The header file contains unique symbols assigned with its occurrence, where no symbol is repeated twice. The binary file contains the code for each symbol.

[1]FOB attack exists in symmetric key encryption, and many techniques are used to solve it, such as using cipher block chaining (CBC) mode, cipher feedback (CFB) mode and counter mode (CTR) [33].

A-RSA system depends on Huffman coding for enhancing security and speeds up the encryption and decryption processes. The Huffman coding output characters have the same level of distribution and this in turn solve the *FOB* attack. To enhance the execution time, *A-RSA* encrypts the header file and blinds the binary file instead of encrypting the entire message. Blinding the binary file by r parameter makes the encrypted message semantically secure.

Consider the following example to explain the usage of Huffman coding in *A-RSA*. Let the message before compression is (`Hello, this message is created by Alice.`). Before compression, the number of bits in this message $= 41 \times 8 = 328$ bits, since each symbol represents in 8 bits (fixed length). Deploying Huffman coding produces the corresponding header and binary files for the above message.

Header File:

```
sp101e110s000c01011000i1110t0110d00111A00
110a11110o00100m10010H01000,01110r00101y01
001À01111g111110b100111.111111h100110
```

Binary File:

```
01000110100010000010001110101011010011011
00001011001011000000011110111110110101111 0
00010101010010111011110011011000111101100 1
11010011010011010001110010111011111101111 0
```

The number of bits for the compressed message in the binary file is equal to 167 bits. Each symbol represents in the binary file is assigned to different length of binary code based on its frequency of occurrence. For example, we represent the characters e by 110 and a with 11110. The character e has shorter binary code comparing with a, since e is repeated in the message 6 times, while a is repeated 2 times.

For un-compressing the message, first read the header file and use its information to build the Huffman tree. Second, read the binary file bit by bit, then begin from the root of the tree. When finding the 0 bit, move to the left on the tree; when finding the 1 bit, move to the right on the tree until finding a leaf node (we have found the symbol). Then repeat the process for all remaining bits until all message characters are retrieved.

B. Random Component (r) in A-RSA

We use a randomized component r in A-RSA cryptosystem to make the encryption process semantically secure. Each time the message is encrypted, different ciphertext is obtained. Therefore, it is hard for an attacker to learn from the ciphertext about the original message, because ciphertexts for the same message look different. We use the letter r to denote the randomized component, where r is a random number generated by using a cryptographically secure pseudo-random number generator and used once for each message (nonce). We use r for blinding the ciphertext of the header file and blinding the binary file. In the case of r less than the ciphertext, we repeat r many times to be as the length of the ciphertext and if the r greater than the ciphertext, we remove the number of bits from r to be as the same length of the ciphertext. And applying the same idea when blinding the binary file.

Algorithm 4 summarizes the *A-RSA* cryptosystem encryption and decryption processes. Huffman code is used for compressing the message and produces the header and binary files. The header file is encrypted using RSA $C = H^e \bmod N$. To make encryption process semantically secure, we choose a random component r and calculate a new ciphertext $C' = C \oplus r$. We called the C' *Mixture*. To make the binary file B semantically secure, B is blinded with r, such as $B' = B \oplus r$. The random component r should be protected, so we use Rabin encryption algorithm for encrypting r.

Algorithm 4 : A-RSA Cryptosystem

Step 1: Keys generation at receiver side

1: Compute RSA public/private keys as explained in Algorithm 1
2: Compute Rabin public/private keys as explained in Algorithm 2

Step 2: Encryption preparation at sender side

1: Generate a random component r for each message
2: Compress message using Huffman code. The outputs from Huffman code are:
- Binary file (B)
- Header file (H)

Step 3: Encryption at sender side

1: Encrypt H by RSA and the result is $C = H^e \bmod N$, where $0<H<N-1$
2: Blind C by r to generate the mixture $C' = C \oplus r$
3: Blind binary file B by r to generate $B' = B \oplus r$
4: Encrypt r by Rabin and the $r' = r^2 \bmod N$

Step 3: Description at receiver side

1: Decrypt r using Rabin cryptosystem as explained in Algorithm 2
2: Compute C from the mixture $C = C' \oplus r$
3: Compute $B = B' \oplus r$
4: Decrypt C using RSA decryption to generate $H = C^d \bmod N$

Step 4: Uncompressed the message at receiver side

1: Pass the H and B to Huffman code to reconstruct the message

V. A-RSA SECURITY ANALYSIS

A-RSA cryptosystem is semantically secure and mitigates the indirect attacks. Choosing a message M and two different random components r_1 and r_2, leads to different ciphers for the same message as shown in Algorithm 4. The attacker cannot know if these ciphertext related to one message or to the different messages. While it was not the case for the basic RSA where encrypting the message with same key always gives the same ciphertext. The semantically secure property of *A-RSA* solves the attacks presented in section III.

1) *A-RSA* cryptosystem is secure against the *common modulus* attack. This attack is used to recover the message that was encrypted using two RSA keys by using the same modules N with different public exponents. In the case of *A-RSA* cryptosystem, the attacker cannot find the message since $C'_1 = C_1 \oplus r$ and $C'_2 = C_2 \oplus r$. Thus, applying the formula $C_1'^{a} C_2'^{b} \equiv M'$ where $M' \neq M$.

2) *A-RSA* cryptosystem is protected from *known-plaintext* attack. *A-RSA* cryptosystem is not deterministic, since it depends on random component that makes the ciphertext always different, even if the same message is encrypted more than one time with the same key. The attacker cannot guess the right plaintext-ciphertext pair. Given $\{(P_1, C'_1), (P_2, C'_2), ..., (P_i, C'_i)\}$. Where $C'_i = C_i \oplus r_i$ for $r_1 \neq r_2 \neq r_3, ..., \neq r_i$. Since each ciphertext is blinded with different r, the attacker cannot guess further new plaintext based on a pre-built set of plaintext-ciphertext.
3) *A-RSA* cryptosystem is protected from *chosen-plaintext* attack because it depends on r, where each message has different r. That is encrypting the same message more than once produces different ciphers, so no one can build a unique set like S that contains plaintext with corresponding ciphertext.
4) *A-RSA* cryptosystem is secure against *timing* attack. In *A-RSA* cryptosystem the attacker still needs the random component r to decrypt the ciphertext, finding d does not allow the attacker to decrypt the message without r since r is *XOR*ed with the ciphertext. Thus, the decryption time depends on the random component and this add some confusion to the attacker.
5) *A-RSA* cryptosystem is protected from the *frequency of blocks* (FOB) attack, since it compresses the message before the encryption. Consequently, the characters in the compressed message has a uniform distribution and thus no repeated blocks are produced for the same message after applying *A-RSA* cryptosystem.
6) The *brute-force* and the *factorization* attacks against *A-RSA* are much harder than in basic RSA, since *A-RSA* relies on the basic RSA and Rabin. Therefore, the brute-force attack have to be carried out against two keys and the factorization have to be done against two public cryptosystems.

VI. A-RSA PERFORMANCE ANALYSIS

For testing the performance of *A-RSA* cryptosystem comparing with the basic RSA, we carried out a set of experiments on different PCs. PC1 is Dell machine runs Windows 7 Professional SP1 with 32-bit on Intel(R) Core(TM), i7-2640M, CPU 2.80 GHz, and 8.00 GB RAM. PC2 is Lenovo machine runs Windows 7 Ultimate SP1 with 32-bit on Intel(R) Core(TM), i5-2520M, CPU 2.50 GHz, and 4.00 GB RAM. PC3 is Dell machine runs Windows 7 Ultimate SP1 with 64-bit on Intel(R) Core(TM), i7-2630QM, CPU 2.00 GHz, and 8.00 GB RAM. We select $N = 1024$ bits and a standard public exponent $e = 65537$ for RSA. We use ten files with different sizes from 1MB to 10MB. Each file is encrypted three times with both RSA and A-RSA cryptosystems and we compute the average of these reading times.

Figure 1 shows the behavior of RSA and *A-RSA* in encryption process for different file sizes. Actually, the figure represents the average execution time. Each point on the chart represents the average of the execution time for three PC's for each cryptosystem. For instance, the point (1, 2.88) represents

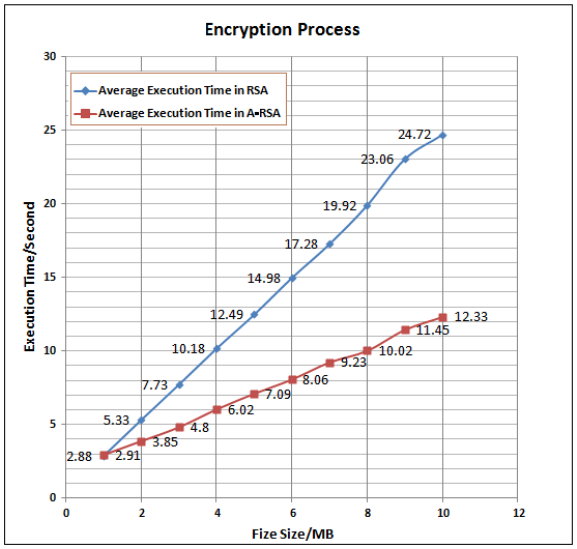

Fig. 1. Encryption Time for RSA and A-RSA Cryptosystems.

the average of (2.5,3.02,3.11) for *RSA*, and the point (1,2.91) represents the average of (2.82,2.65,3.26) for *A-RSA*.

Figure 1 shows that the encryption time directly proportional to the file size for both cryptosystems, but *A-RSA* is significantly faster than basic RSA. This speedup in the encryption time refers to the fact that *A-RSA* does not encrypt the entire message. It encrypts the header file and blinds the binary file, which is generated from compression phase. However, *A-RSA* is slower than basic RSA in encryption process for 1MB file. *A-RSA* encrypts 1MB file within 2.91s while it takes 2.88s using the basic RSA. The reason behind this fact is that *A-RSA* encryption process passes through four phases: statistical analysis, compression, encryption and blinding of the message. Since the 1MB is relatively small size, the analysis, compression, and blinding may take more time than the encryption itself.

The behavior of *RSA* and *A-RSA* in decryption process is depicted in Figure 2. *A-RSA* is faster than the basic RSA in decryption process because RSA needs to decrypt the entire message, but *A-RSA* just needs to decrypt the header file which is much smaller than the entire message.

Comparing Figure 2 with Figure 1, one can see that the speedup ratio between the *A-RSA* and the basic RSA in decryption process is much better than in the encryption. Also, there is a big difference in the execution time of encryption and decryption and this returns to the two reasons. First, private exponent d is larger than public exponent e. Second, when encrypting the file using RSA, the encrypted file (ciphertext) is greater than original file. In the basic RSA, the encryption is faster than decryption or may equally, but this is not the case for *A-RSA* system. *A-RSA* decryption process is faster than the encryption. The reason for this observation is existence of four phases in *A-RSA* encryption process (analysis, compression, encryption and blinding), but there are three phases in *A-RSA* decryption process that make its decryption faster (un-blinding, decryption and un-compression). Taking the overall average of encryption and decryption times of 10 files from the three PCs shows that *A-RSA* cryptosystem is faster than basic RSA by

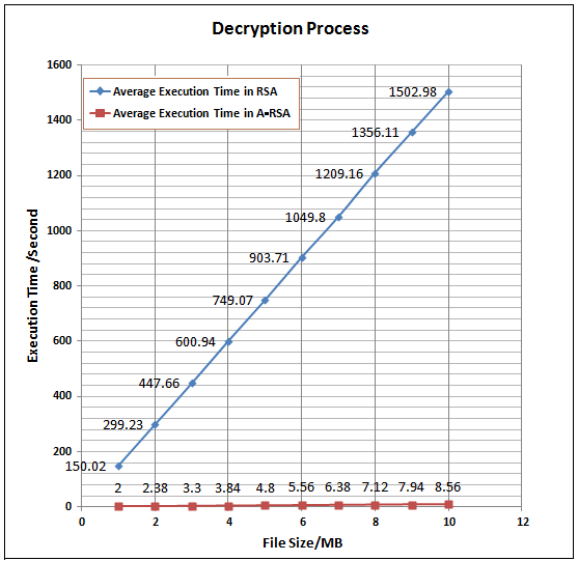

Fig. 2. Decryption Time for RSA and A-RSA Cryptosystems

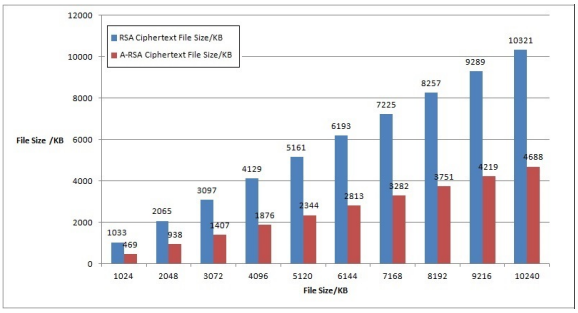

Fig. 3. RSA and A-RSA Ciphertext Size

45% in encryption process and 99% in decryption process.

A-RSA cryptosystem uses Huffman compression algorithm to reduce file sizes. Thus, the files generated from *A-RSA* system are smaller than the original files, which is helping for reducing resource usage, such as data storage space. Figure 3 shows the size of ten files after the encryption by using RSA and *A-RSA* cryptosystems. *A-RSA* cryptosystem results in an efficient use of the space. We found that RSA system increases the size of ciphertext by 1% compared to the original file size, while *A-RSA* cryptosystem reduces the file size by 54% from the original size. However, the percentage of file size reduction by using *A-RSA* cryptosystem depends on the number of occurrences of the symbols inside the file.

VII. Summary and Conclusion

RSA is popular and most widely used cryptosystem through the years until now, it is used in different applications and protocols. Even though, RSA seems to be robust and secure, attackers succeeded to exploit some properties of RSA algorithm and its implementation to carry out some attacks, such as *common modulus*, *known-plaintext*, *chosen-plaintext*, and *timing* attacks. Moreover, *RSA* does not solved block redundancy in the message. Besides, RSA is a time consuming algorithm, its speed is very slow comparing with symmetric key encryption algorithms like *AES* algorithm.

We suggested a new modifications to enhance *RSA* cryptosystem called *A-RSA*. This enhanced RSA combines the basic RSA with another cryptosystem called *Rabin*. A randomized component r is added to the RSA to make it semantically secure and this component is encrypted by Rabin. *A-RSA* uses Huffman coding algorithm to remove the redundancy in the message based on statistical analysis to generate Header H and Binary B files. *A-RSA* encrypts H and blinds B with r instead of the encrypting the entire message.

The enhanced version of RSA is more secure, faster encryption and decryption, and has shorter encrypted message size comparing with the basic RSA. Thus, *A-RSA* cryptosystem has the following characteristics:

1) Semantically secure compering to the basic RSA.
2) Secure against *frequency of block* attacks.
3) More secure against *brute-force* attack than the RSA.
4) Faster encryption and decryption compare with basic RSA.
5) Produce shorter cipher size.

A-RSA cryptosystem could be the better choice for encrypting large data in a secure and fast manner over a public network. Due the drastically improvement in the *A-RSA* encryption and decryption time, it might more convenient to use only the *A-RSA* for encryption and decryption without the use of another symmetric key for encrypting the entire message.

References

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[2] Z. Wang, Z. Jia, L. Ju, and R. Chen, "Asip-based design and implementation of rsa for embedded systems," in *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*. IEEE, 2012, pp. 1375–1382.

[3] A. Freier, P. Karlton, and P. Kocher, "The secure sockets layer (ssl) protocol version 3.0," August 2011, rFC6101.

[4] T. Dierks and E. Rescorla, "The transport layer security (tls) protocol version 1.2," August 2008, rFC5246.

[5] D. Boneh and H. Shacham, "Fast variants of rsa," *CryptoBytes*, vol. 5, no. 1, pp. 1–9, 2002.

[6] W. Diffie, "The first ten years of public-key cryptography," *Proceedings of the IEEE*, vol. 76, no. 5, pp. 560–577, 1988.

[7] H. Orman and P. Hoffman, "Determining strengths for public keys used for exchanging symmetric keys," 2004, rFC3766.

[8] D. Boneh, "Twenty years of attacks on the rsa cryptosystem," *Notices of the AMS*, vol. 46, no. 2, pp. 203–213, 1999.

[9] T. Takagi, "Fast rsa-type cryptosystem modulo p k q," in *Advances in Cryptology–CRYPTO'98*. Springer, 1998, pp. 318–326.

[10] R.-J. Hwang, F.-F. Su, Y.-S. Yeh, and C.-Y. Chen, "An efficient decryption method for rsa cryptosystem," in *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*, vol. 1. IEEE, 2005, pp. 585–590.

[11] S. Y. Yan, *Cryptanalytic attacks on RSA*. Springer, 2007.

[12] M. Bishop, "What is computer security?" *Security & Privacy, IEEE*, vol. 1, no. 1, pp. 67–69, 2003.

[13] W. Stallings, *Network security essentials applications and standards, 5th*. Pearson Education, 2013.

[14] W. Diffie and M. E. Hellman, "New directions in cryptography," *Information Theory, IEEE Transactions on*, vol. 22, no. 6, pp. 644–654, 1976.

[15] E. Rescorla, "Diffie-hellman key agreement method," June 1999, rFC2631.

[16] M. O. Rabin, "Digitalized signatures and public-key functions as intractable as factorization," Massachusetts Institute of Technology, Tech. Rep., 1979.

[17] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Advances in Cryptology*. Springer, 1985, pp. 10–18.

[18] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, Jan. 1987, american Mathematical Society.

[19] V. S. Miller, "Use of elliptic curves in cryptography," in *Lecture Notes in Computer Sciences; on Advances in cryptology—CRYPTO 85*, vol. 218. Springer-Verlag New York, Inc., 1986, pp. 417–426.

[20] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes," in *Advances in Cryptology–CRYPTO'99*. Springer, 1999, pp. 537–554.

[21] L. Harn, W.-J. Hsin, and M. Mehta, "Authenticated diffie–hellman key agreement protocol using a single cryptographic assumption," *IEE Proceedings-Communications*, vol. 152, no. 4, pp. 404–410, 2005.

[22] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval, "Key-privacy in public-key encryption," in *Advances in Cryptology–ASIACRYPT 2001*. Springer, 2001, pp. 566–582.

[23] R. S. Douglas, "Cryptography theory and practice," 1995.

[24] S. Sharma, P. Sharma, and R. S. Dhakar, "Rsa algorithm using modified subset sum cryptosystem," in *Computer and Communication Technology (ICCCT), 2011 2nd International Conference on*. IEEE, 2011, pp. 457–461.

[25] A. S. Wander, N. Gura, H. Eberle, V. Gupta, and S. C. Shantz, "Energy analysis of public-key cryptography for wireless sensor networks," in *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*. IEEE, 2005, pp. 324–328.

[26] G. HS, M. Seetha, A. K. Koundinya, and P. CA, "Comparative study and performance analysis of encryption in rsa, ecc and goldwasser-micali cryptosystems," *International Journal of Application or Innovation in Engineering & Management (IJAIEM)*, vol. 3, no. 1, pp. 111–118, January 2014.

[27] A. K. Srivastava and A. Mathur, "The Rabin cryptosystem & analysis in measure of chinese reminder theorem," *International Journal of Scientific and Research Publications*, p. 493, 2013.

[28] V. B. Andrea Pellegrini and T. Austin, "Fault-based attack of rsa authentication," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2010, pp. 855 – 860. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5456933

[29] I. K. Salah, A. Darwish, and S. Oqeili, "Mathematical attacks on rsa cryptosystem," *Journal of Computer science*, vol. 2, no. 8, p. 665, 2006.

[30] C. Cobb, *Cryptography for dummies*. John Wiley & Sons, 2004.

[31] W. H. Wong, "Timing attacks on RSA: revealing your secrets through the fourth dimension," *Crossroads*, vol. 11, no. 3, pp. 5–5, 2005.

[32] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '96. London, UK, UK: Springer-Verlag, 1996, pp. 104–113. [Online]. Available: http://dl.acm.org/citation.cfm?id=646761.706156

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

[34] G. J. Simmons, *Contemporary cryptology: the science of information integrity*. IEEE press, 1994.

[35] C. E. Shannon, "Communication theory of secrecy systems*," *Bell system technical journal*, vol. 28, no. 4, pp. 656–715, 1949.

[36] D. Salomon, *A concise introduction to data compression*. Springer, 2007.