Padding Schemes for RSA and their Security

Calvin Ryan D'Souza
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY 14586
cd5184@cs.rit.edu

***Abstract*—What is RSA? Why padding schemes? Paddings schemes and prime number details**

I. Introduction

Digital signatures are widely used on the internet for verifying the identity of websites. The Rivest-Shamir-Adleman encryption scheme commonly known as RSA is one of the main algorithms used for implementing digital signatures. RSA is based on the factorization problem. The security of RSA is based on the fact that the product of 2 large prime numbers is extremely difficult to factor. RSA is computationally expensive since it is implemented with 2048/4096-bit numbers. The primitive versions of RSA as mentioned in [1] are usually referred to as textbook implementations.

Textbook RSA is a deterministic cryptosystem (i.e., has no random component) which makes it susceptible to chosen-plaintext attacks. This means it is not semantically secure. Thus, in real-world scenarios, RSA is almost always implemented with a padding scheme. Padding Schemes add a random component to RSA and makes it probabilistic in nature. Before actually encrypting the message, the message is padded with some random string. This forms an embedded string which is then sent for RSA encryption. Commonly used padding schemes with RSA are PKCS #1 v1.5 (RSAES-PKCS1-v1_5) and Optimal Asymmetric Encryption Padding (RSAES-OAEP) [1]. The paper implements these schemes in accordance with [2] for experimental purposes.

Prime numbers generated are categorized as basic primes and safe primes for each RSAES padding scheme. These are generated using random number/bit generators and primality tests as mentioned in [2]. Safe primes are generated since prime numbers can often be factorized using Pollard's p-1 algorithm [3]. This does not usually occur in case of prime numbers generated for cryptographic purposes. But if it does occur, it will be detrimental to the RSAES implementations even though a padding scheme is employed. The paper provides insight into implementing RSA with safe primes versus basic primes and the computational complexities related to them.

RSAES-PKCS1-v1_5 has been broken as demonstrated in [4]. However, it is still recommended as usable by FIPS 186-4 [2] and FIPS 186-5 [5]. This paper tests the feasibility of this attack in practical scenarios and discusses the security of RSAES-PKCS1-v1_5. The paper also provides results on how RSAES-OAEP may reduce message length for RSA signatures, thus increasing computational costs since a higher bit RSA implementation is required. SHA-3 is recommended as a Mask generation function in [5]. The paper also shows experiments with SHA-3 used as a mask generation function instead of older version of SHA. The results show how SHA-3/SHAKE can be used a standalone mask generation function versus current implementations described in [1]. Moreover, the paper looks at OAEP+ [6] and other padding scheme modification literature and contrasts them with RSAES-OAEP and RSA-PKCS1-v1_5.

II. PREREQUISITES & PRIOR WORK

A. *Safe Primes Vs Basic Primes*

The RSA modulus used for encryption and decryption is the product of two randomly generated large prime numbers. RSA security is based on the premise that this modulus is hard to factorize. However, for a prime number p if p-1 has many small factors, it is possible to find p given p-1. This is called Pollard's Algorithm. This introduces the concept of safe primes.

For a prime p, if p-1 can be expressed as 2 * prime (m), then p is called a Safe prime. The prime number m is known as a Sophie Germain prime. Some examples of Sophie Germain primes are 2, 3, 11, 23 and their respective Safe primes are 5, 7, 23, 47. Prime numbers for RSA are generated using cryptographically safe random number generators. Thus, the primes numbers used for generating the RSA modulus are often not susceptible to Pollards Algorithm. However, if the prime numbers are chosen with care, it could be detrimental to RSA.

As part of this project, we have compared the cost of implementing RSA with basic primes and safe primes. The prime number generation algorithm used as part of the project generates a set of primes and then filters them for safe primes.

B. *Key Generation Scheme for RSA and Textbook RSA*

As mentioned earlier, the basic RSA encryption scheme without padding is known as Textbook RSA. Textbook n-bit RSA functions as follows:-

1) Randomly generate two n/2-bit prime numbers (p and q). Compute $n = p*q$.
2) n is the modulus for RSA.
3) Compute Φ (n) = (p-1)(q-1).

4) Choose public-key exponent e. e is chosen such that GCD(e, Φ (n)) = 1.
5) Compute private-key exponent, $d \equiv e^{-1}$ *mod* $\Phi(n)$.
6) The public-key pair for RSA = (e, n) and private-key pair = (d, n).

Consider plaintext (p) and corresponding ciphertext (c). Since we have the key-pairs, encryption and decryption in textbook RSA is as follows:-

- Encryption: $c = m^e$ *mod n*
- Decryption: $m = c^d$ *mod n*

III. Padding Schemes

RSA in the real-world is used with Padding Schemes given its deterministic nature. FIPS 186-4 [2] which outlines Digital Signature Standards (DSS) recommends PKCS #1 v1.5 and OAEP as viable Padding schemes with RSA. Padding schemes modify the input message to generate encoded messages which are then used for encryption by RSA. A basic schematic for encryption and decryption using a Padding scheme is shown below:-

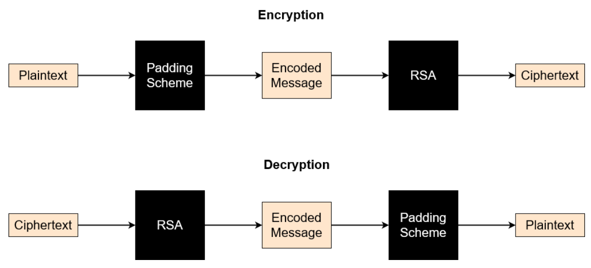

Fig. 1. Padding Scheme Architecture

Thus, on the sender's end, the input message is encoded by the padding scheme and then encrypted using RSA. On the receiving end, the ciphertext is decrypted using RSA and then decoded using the padding scheme.

A. PKCS #1 v1.5

Public-Key Cryptography Standard (PKCS) #1 v1.5 is one of the Padding Schemes which is employed with RSA. It was officially published in RFC2313 [7] in March 1998. It is a simple padding scheme but susceptible to timing attacks. PKCS #1 v1.5 is considered broken, since there is a known attack against it called Bleichenbacher's [4] attack. Despite being broken, FIPS 186-4 [2] still recommends PKCS #1 v1.5 as a viable Padding Scheme with RSA. As explained above, padding schemes are used for encoding the input message before it is sent for encryption.

The input string for RSA is usually the same byte length as the RSA modulus. Let this byte length be k. PKCS #1 v1.5 has a minimum padding length of 11 bytes. Out of these 11 bytes, the starting 2 bytes (0x00 and 0x02) and last byte (0x00) of the padding are fixed. Hence, the randomized padding is at least 8 bytes long. This results in the message length being restricted to a maximum of k-11 bytes. Encoding using PKCS #1 v1.5 has the following steps:-

1) Let the number of bytes in the RSA modulus be k.
2) Length checking the input message (m): If $len(m) > k$ - 11 bytes, report error and stop.
3) Create encoded message as follows:-
   a) Add *0x00* as 1st byte
   b) Append *0x02* as 2nd byte
   c) Append (k - 8 - $len(m)$) non-zero bytes as randomized padding bytes.
   d) Append *0x00* as separation byte
   e) Append input message m to form final encoded message.

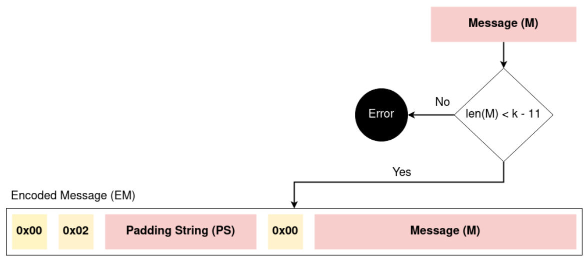

Fig. 2. Encoding plaintext with PKCS #1 v1.5

The encoded message is then encrypted using public-key pair and RSA to generate the ciphertext. On the receiver's end, the ciphertext is decrypted using private-key pair and RSA to generate a possible encoded message. As mentioned earlier, PKCS # 1 v1.5 is susceptible to attacks and thus the encoded message needs to be validated for correctness. After the encoded message is successfully validated, the original plaintext is recovered. The encoded message is decoded and validated as follows:-

1) Length checking encoded message (EM): If $len(EM)$!= k or $len(EM) < 11$, report error and stop.
2) If 1st byte is not *0x00*, report error and stop. Else, discard byte.
3) If 1st byte is not *0x02*, report error and stop. Else, discard byte.
4) Traverse bytes until *0x00* is encountered. If number of bytes traversed < 8, report error and stop. Else, discard bytes.
5) If 1st byte is not *0x00*, report error and stop. Else, discard byte.
6) Since all padding bytes are discarded, original input message can now be extracted.

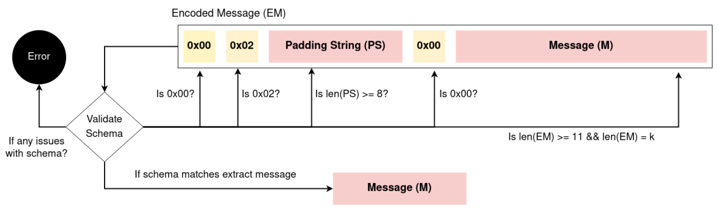

Fig. 3. Encoding EM with PKCS #1 v1.5

B. Optimal Asymmetric Encryption Padding (OAEP)

Optimal Asymmetric Encryption padding (OAEP) is one of the Padding Schemes which is employed with RSA. It was officially published in RFC2437 [8] in October 1998. OAEP is the preferred padding scheme since it is considered to be IND-CCA2 secure. However, there have been studies which contradict this claim stating OAEP is only IND-CCA1 secure.

In comparison to PKCS#1 v1.5, OAEP also incorporates the use of a Hash function hash() and Mask Generation Function (MGF) mask(). Output length of the hash function is represented as hlen bytes.

OAEP also uses an optional label for each message. For this project, we consider the label to always be an empty string. As with PKCS#1 v1.5, the input message is encoded using OAEP before encryption on the sender's end. For a k- byte modulus RSA, the encoding process is as follows:

1) Length checking message(m): If *len(m)* > *k* - 2 - 2 * *hlen*, report error and stop.
2) Create data block (*DB*):
   a) Append *Hash(L)* as start of string.
   b) Append (*k* - *len(m)* - 2 * *hlen* - 2) *0x00* bytes.
   c) Append *0x01* as separation byte to mark end of *0x00* bytes. Append message to form *DB* of length (*k* - *hlen* - 1) bytes.
3) Generate random seed (*s*) where *len(s)* = *hlen* bytes.
4) Generate *DBmask* = *mask(s, k - hlen - 1)*
5) Generate *maskedDB* = *DBmask* $\oplus$ *DB*
6) Generate *seedMask* = *mask(maskedDB, hlen)*
7) Generate *maskedSeed* = *seedMask* $\oplus$ *seed*
8) Generate Encoded message (*EM*) as:-

$$EM = 0x00||maskedSeed||maskedDB$$

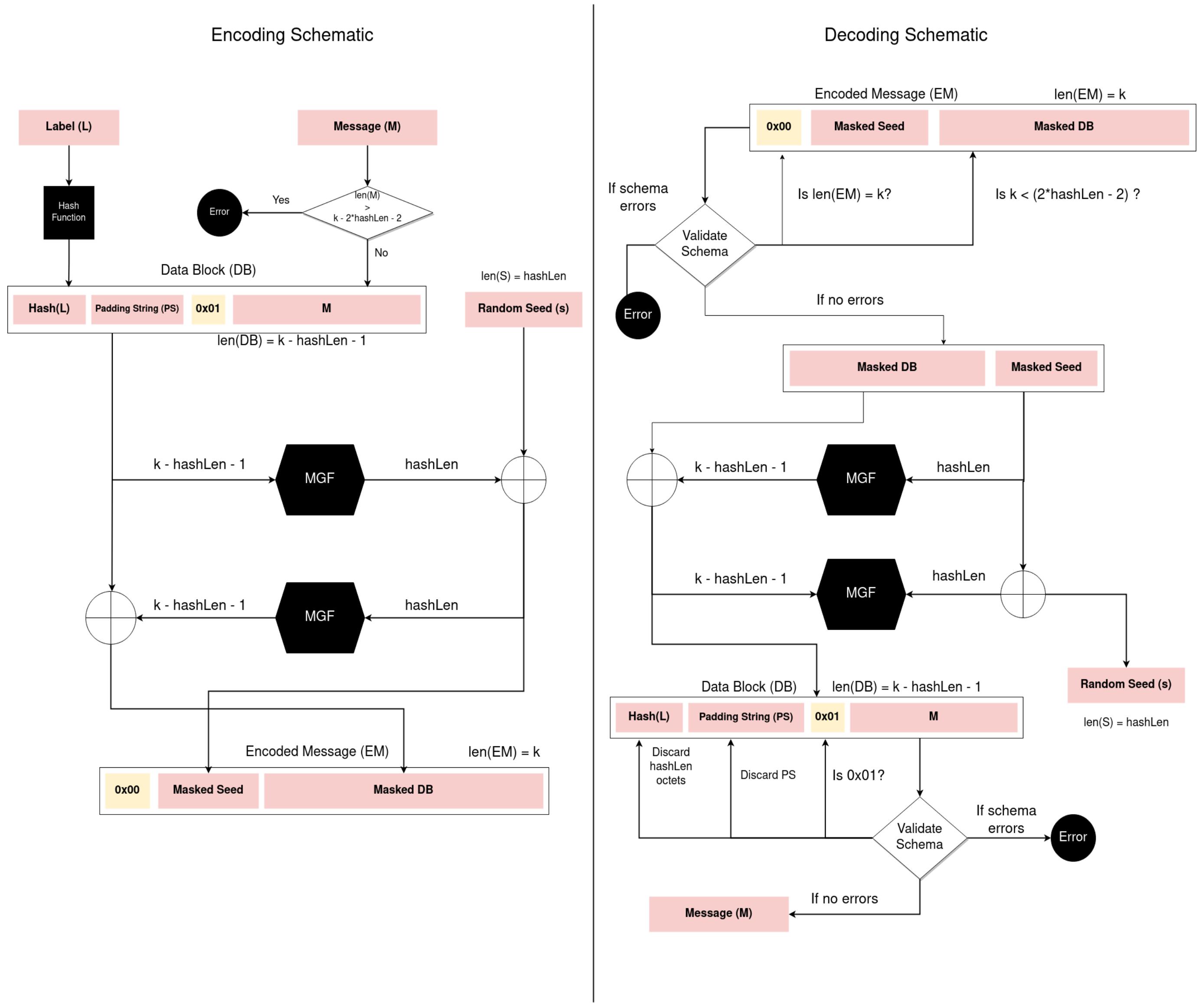

Fig. 4. Encoding and Decoding with OAEP

The MGF uses hash() along with a counter to produce an output string os of given length n for a given input string is. This can be represented as:

$$mask(is, n) = opwherelen(op) = n$$

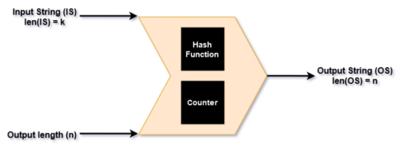

Fig. 5. Mask Generation Function

The encoded message is then ecnrypted with the public-key pair using RSA. The generated ciphertext is transmitted to the reciever who then decrypts this using the private-key pair for RSA. Since there are no known attacks against OAEP, the message structure isnt validated. Instead, before decryption, a basic length check for the ciphertext (*c*) is performed:- if *len(c) != k* bytes OR *len(c)* $<$ *2hLen + 2*, the ciphertext is discarded. Else, it is decrypted to acquire the encoded message (EM). The decoding process for retrieving the plaintext from EM is as follows:-

1) If *len(EM) != k* bytes OR *len(EM)* $<$ *2hLen + 2*, report error and stop.
2) Split *EM* as *0x00* || *maskedSeed* || *maskedDB*.
3) Retrieve Data Block (*DB*):-
    a) Let *seedMask = mask(maskedDB, hlen)*
    b) Let *seed = seedMask* $\oplus$ *maskedSeed*
    c) Let *DBmask = mask(s, k - hLen - 1)*
    d) Let *DB = DBmask* $\oplus$ *maskedDB*
4) Split *DB* to retrieve original plaintext:-
    a) Discard *Hash(L)* from the data block.
    b) Discard *0x00* bytes until separation byte *0x01* is encountered. If *0x01* is not encountered, report error and stop.
    c) Discard *0x01* and recover plaintext.

C. *Experiments with message length*

Experiment info

IV. Bliechenbacher Attack

Information regarding the attack

A. *Attack implementation in experimental scenario*

Some information along with a table

V. SHA-3/SHAKE Implementation with OAEP

Some introductory text

A. *Advantages of SHA-3 over earlier versions*

Information and images

B. *Implementation details and length-based scenarios*

Information and images

VI. Results

Results broken down as per experiments and tests. Not defined currently, thus no sections created.

VII. Conclusion

Cumulative conclusion based on research and experiments.

References

[1] K. Moriarty, B. Kaliski, J. Jonsson, and A. Rusch, "Pkcs# 1: Rsa cryptography specifications version 2.2," *Internet Engineering Task Force, Request for Comments*, vol. 8017, p. 72, 2016.

[2] C. F. Kerry and C. R. Director, "Fips pub 186-4 federal information processing standards publication digital signature standard (dss)," 2013.

[3] L. M. Adleman and K. S. McCurley, "Open problems in number theoretic complexity, ii," in *International Algorithmic Number Theory Symposium*. Springer, 1994, pp. 291–322.

[4] D. Bleichenbacher, "Chosen ciphertext attacks against protocols based on the rsa encryption standard pkcs# 1," in *Annual International Cryptology Conference*. Springer, 1998, pp. 1–12.

[5] N. I. of Standards and Technology, "Fips pub 186-5 (draft) federal information processing standards publication digital signature standard (dss)," 2013.

[6] V. Shoup, "Oaep reconsidered," in *Annual International Cryptology Conference*. Springer, 2001, pp. 239–259.

[7] B. Kaliski, "Pkcs# 1: Rsa encryption version 1.5," RFC 2313, March, Tech. Rep., 1998.

[8] B. Kaliski and J. Staddon, "Pkcs# 1: Rsa cryptography specifications version 2.0," 1998.