A Java implementation of the RSA algorithm

CSE 5673 – Cryptology Project

Dr. Marius Silaghi

Group 7

Zubin Kadva

Zongqiao Liu

1. Introduction

The RSA algorithm was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and first published in 1978. The Rivest-Shamir-Adleman (RSA) scheme has since that time reigned supreme as the most widely accepted and implemented general-purpose approach to public-key encryption.

1.1. The RSA Algorithm

The algorithm is stated as follows:

1. Compute two large prime numbers p and q such that q is relatively prime to q
2. Compute $n = p * q$
3. Select e such that $gcd(\emptyset(n), e) = 1 \; and \; 1 < e < \emptyset(n)$
4. Compute $d \equiv e^{-1} \; mod \; \emptyset(n)$
5. Encryption $C = M^e \; mod \; n$
6. Decryption $M = C^d mod \; n$

Note: $\emptyset(n) = (p - 1) * (q - 1)$

Fig 1.1 The RSA algorithm

Public key: $\{n, e\}$

Private key: $\{n, e, p, q, d \; mod(p - 1), d \; mod \; (q - 1), q^{-1} mod \; p\}$

1.2. Fast decryption with the Chinese Remainder Theorem (CRT)

The steps include:

1. Compute $d_p = d \; mod(p - 1)$
2. Compute $d_Q = d \; mod(q - 1)$
3. Compute $q_{inv} = q^{-1} mod \; p$
4. Compute $m1 = C^{d_p} mod \; p$
5. Compute $m2 = C^{d_Q} mod \; q$
6. Compute $h = (m1 - m2) * q_{inv} mod \; p$
7. Compute $m = m2 + q * h$

Fig 1.2 Decryption using CRT

2. Padding

2.1. Encoding

RSAES-PKCS1-V1_5-ENCRYPT ((n, e), M)

Input: (n, e) recipient's RSA public key (k denotes the length in octets of the modulus n)

M message to be encrypted, an octet string of length mLen, where

$$mLen \leq k - 11$$

Output: C ciphertext, an octet string of length k

1. Length checking: If $mLen > k - 11$, output "message too long" and stop.
2. EME-PKCS1-v1_5 encoding:

   a. Generate an octet string PS of length $k - mLen - 3$ consisting of pseudo-randomly generated nonzero octets. The length of PS will be at least eight octets.

   b. Concatenate PS, the message M, and other padding to form an encoded message EM of length k octets as

$$EM = 0x00 \,||\, 0x02 \,||\, PS \,||\, 0x00 \,||\, M$$

3. RSA encryption as above
4. Output C

2.2. Decoding

RSAES-PKCS1-V1_5-DECRYPT (K, C)

Input: K recipient's RSA private key

C ciphertext to be decrypted, an octet string of length k, where k is the length in octets of the RSA modulus n

Output: M message, an octet string of length at most $k - 11$

1. Length checking: If the length of the ciphertext C is not k octets (or if k < 11), output "decryption error" and stop.
2. RSA decryption as above
3. EME-PKCS1-v1_5 decoding:

   Separate the encoded message EM into an octet string PS consisting of nonzero octets and a message M as

$$EM = 0x00 \,||\, 0x02 \,||\, PS \,||\, 0x00 \,||\, M$$

   If the first octet of EM does not have hexadecimal value 0x00, if the second octet of EM does not have hexadecimal value 0x02, if there is no octet with hexadecimal value 0x00 to separate PS from M, or if the length of PS is less than 8 octets, output "decryption error" and stop.

4. Output M

3. Java implementation

The RSA algorithm is implemented in the Java programming language. It makes use of the *BigInteger* library.

3.1. Invocation

1. *java –h*

2. *java RSA -K -p public_key_file -s secret_key_file -b bits -y Miller_Rabin_certainty*

3. *java RSA -e -m plaintext_file -p public_key_file -c ciphertext_file*

4. *java RSA -d -c ciphertext_file -s secret_key_file -m plaintext_file*

3.2. Functions

Some useful functions are as follows:

```
generateKeys (int bits, int certainty, String publicKeyFile, String secretKeyFile)
```

```
encrypt (String publicKeyFile, String plainFile, String cipherFile)
```

```
decrypt (String secretKeyFile, String plainFile, String cipherFile)
```

```
pad (BigInteger n, BigInteger message)
```

```
unpad (BigInteger n, BigInteger cipher)
```

4. References

1. William Stallings, "*Cryptography and Network Security: Principles and Practice Fifth Edition*", ISBN 10: 0-13-609704-9, ISBN 13: 978-0-13-609704-4, Prentice Hall, 2011

2. RSA Laboratories, "*PKCS#1 v2.2: RSA Cryptography Standard*", 2012