Efficiency of FIPS-Compliant RSA Cryptosystem Across Various Systems

George Wood
Truman State University
Kirksville, United States
glw3638@truman.edu

Chetan Jaiswal
Truman State University
Kirksville, United States
cjaiswal@truman.edu

***Abstract*—The RSA cryptosystem has an established position as one of the most secure methods of secret sharing. However, it is also known to be a rather slow algorithm. As a result, the feasibility of its use on large amounts of data depends heavily on the components of the system it is run on. This work will analyze and discuss the efficiency of an RSA-CRT implementation on a range of computer systems. We will discuss the hardware and software components present on each system and compare several measures of the efficiency of execution. Given the increasingly critical nature of efficient and effective cybersecurity, the goal of this study is to contribute towards finding ideal conditions for the execution of this powerful cryptosystem.**

***Index Terms*—RSA, cryptography, complexity, efficiency.**

I. Introduction

As the scope of data stored in computer systems has expanded, so has the need for effective cybersecurity systems. The RSA cryptosystem, short for Rivest-Shamir-Adleman after its creators, has proven to be one of the most reliable systems available for secret sharing. Due to the notorious factoring problem, breaking RSA encryption is practically and computationally unfeasible using any currently known means. The system is not without faults, however, as the time it requires to operate is well known to be quite large. As a result, the size of messages that it can practically encode is quite small, reducing its use largely to sharing of keys used in other encryption schemes, or identity authentication.

RSA is utilized in several manners that are essential to common digital procedures. For secret sharing, the system can be used to ensure that confidential information, such as a key used for AES, cannot be discovered even if the secret is intercepted by a third party. In terms of average internet use, RSA allows for common users to be assured that the parties they interact with are who they claim to be. Even though the size of data such as these is quite small, these operations have to be performed extremely frequently, making the importance of efficient execution extremely high. As a result, research towards finding ideal running environments for the system is of considerable value.

The work performed for this study aims to discover environments and elements that are conducive to best-case efficiency of RSA. In section 2, we will discuss the basics of the RSA cryptosystem, followed by a description of the implementation used in section 3. Section 4 will give insight on what will be used to measure execution efficiency, and section 5 will provide comparison of the data that is found.

II. The RSA Cryptosystem

The RSA cryptosystem is a public-key cryptography system that facilitates secret sharing. It makes use of a public key, denoted by *e*, a private key, denoted by *d*, and a modulus value, denoted by *n*. *n* is determined with two distinct large prime numbers p and q. Depending on which of the two keys are used for encoding, either encryption/decryption or signing/authentication can be accomplished. Standard encryption and decryption is accomplished with two equations. A plaintext message *m* that is encrypted to a ciphertext *c* message with *e* as shown in eqn. 1 can only be decrypted with *d* as shown in eqn. 2.

$$[1] c = m^e \ mod \ n$$
$$[2] m = c^d \ mod \ n$$

Signing and authentication can be accomplished via switching the keys used in the equation, as shown in eqn. 3 and eqn. 4 respectively.

$$[3] c = m^d \ mod \ n$$
$$[4] m = c^e \ mod \ n$$

An alternative mechanism based on the Chinese Remainder Theorem (CRT) can be used to increase efficiency both for decryption and, as it is analogous to the decryption process, for signing as well. Using this scheme, intermediate values determined using p and q in addition to the inverse of $q \ mod \ p$ are utilized. These values are determined as follows:

$$[5] dP = (1/e) \ mod \ (p-1)$$
$$[6] dQ = (1/e) \ mod \ (q-1)$$
$$[7] qInv = (1/q) \ mod \ p$$

These values can then be used to compute the output message without such costly operations as $modn$. Using decryption as an example, this is accomplished as follows:

$$[8] m_1 = c^d P \ mod \ p$$
$$[9] m_2 = c^d Q \ mod \ q$$
$$[10] h = qInv(m_1 - m_2) \ mod \ p$$

$^{[11]}m = m_2 + (h)(q)$

Signing with CRT can be accomplished in the same manner, simply swapping the positions of m and c.

III. Implementation

The specific implementation of RSA was coded in C++11, making use of several classes. All RSA-specific operations are performed within the RSACipher class, which in turn makes use of two other classes. The SHAHash class performs any and all hashing operations, and can utilize various forms of SHA, including the commonly used SHA-224 and SHA-256. Lastly, the RandGen class generates random numbers for use in the creation of the primes p and q, and the private and public keys.

Given that RSA makes use of prime numbers that are thousands of bits large, the 32- or 64-bit arithmetic currently utilized by many common systems is nowhere near large enough to accommodate the requirements of RSA. To handle this issue, the implementation makes use of the GNU Multiprecision arithmetic library (GMP), which allows for arithmetic operations on numbers of arbitrary bitlength.

To the greatest extent possible (and necessary), the implementation is compliant with federal guidelines for RSA established across several Federal Information Processing Standards (FIPS) publications created by the National Institute of Standards and Technology (NIST), primarily FIPS 186-4. A complete list of the relevant documentation followed can be found in the references section. The sole exception to compliance is in the RandGen class, which does not make use of a sufficiently strong external entropy/noise source. Given that the focus of this research is on efficiency rather than security, and the difficulty inherent in establishing such a source of randomness, this element was foregone in favor of research progress.

All interaction with the program is accomplished via a simple user interface, allowing for encryption, decryption with or without CRT, signing with or without CRT, authentication, and key generation. Although slight changes were necessary to ensure compatibility across various systems, the core of this interface remained largely the same across all data collection. Attempts were made at a implementation-independent GUI, but were eventually foregone in favor of research progress.

Although there are numerous further guidelines that establish padding schemes and other various additions to RSA such as those provided in PKCS #1, the focus of this work is on the base cryptosystem, with no such additional systems added.

IV. Methodology

In order to gather data on the efficiency of RSA, our code was ran on various systems enumerated in Appendix A. Additional code was utilized to record a number of aspects of the program's execution. The data types recorded includes those listed below.

A. Data Measured

1) Computational Complexity
2) Space Complexity
3) Execution Time
4) CPU Time Utilized
5) System Resources Utilized/Held

B. Data Analysis

V. Results

VI. Conclusion

Conclusion text

Appendix A
Hardware Utilized

1) Desktop 1
   - Processor: Intel Core i7-4790K
   - Cores: 4 Physical, 8 Virtual
   - Clock Speed: 4.00 GHz, up to 4.40 GHz w/ turbo
   - RAM: 8 GB
   - OS: Windows 7 Home Premium, Service Pack 1, 64-bit
2) Desktop 2
   - Processor: Intel Core i5
   - Cores: 2 Physical, 4 Virtual
   - Clock Speed: 2.7 GHz
   - RAM: 8 GB DDR3 1333 MHz
   - OS: OS X 10.7.5
3) Laptop 1
   - Processor: Intel Core i5-3427U
   - Cores: 2 Physical, 4 Virtual
   - Clock Speed: 1.80 GHz, up to 2.80 GHz w/ turbo
   - RAM: 8 GB
   - OS: Ubuntu 16.04 LTS
4) Laptop 2
   - Processor: Intel Core i5-8250U
   - Cores: 4 Physical, 8 Virtual
   - Clock Speed: 1.6 GHz, up to 3.4 GHz w/ turbo
   - RAM: 8GB LPDDR3 1866MHz
   - OS: Windows 10, 64-bit
5) Laptop 3
   - Processor: Intel Core i5-4258U
   - Cores: 2 Physical, 4 Virtual
   - Clock Speed: 2.4 GHz
   - RAM: 8 GB 1600MHz DDR3
   - OS: OS X Yosemite 10.10.5
6) Laptop 4
   - Processor: Intel Core i5-m7200U
   - Cores: 2 Physical, 4 Virtual
   - Clock Speed: 2.50 GHz Nominal, 2.70 GHz Actual, up to 3.10 GHz w/ turbo
   - RAM: 8 GB DDR4
   - OS: Windows 10 Home, 64-bit
7) Laptop 5
   - Processor: Intel Core M3-6Y30

- Cores: 2 Physical, 4 Virtual
- Clock Speed: 0.9 GHz, up to 2.2 GHz w/ turbo
- RAM: 8 GB
- OS: Windows 10 Home, 64-bit

8) Laptop 6
   - Processor: Intel Core i5-5200U
   - Cores: 2 Physical, 4 Virtual
   - Clock Speed: 2.2 GHz, up to 2.7 GHz w/ turbo
   - RAM: 8 GB DDR3
   - OS: Windows 7 Home, Service Pack 1, 64-bit
9) Laptop 7
   - Processor: Intel Core i7-7820HQ
   - Cores: 4 Physical, 8 Virtual
   - Clock Speed: 2.9 GHz, up to 3.9 GHz w/ turbo
   - RAM: 16 GB
   - OS: OS X 10.13.2
10) Laptop 8
    - Processor: Intel Core i5-4260U
    - Cores: 2 Physical, 4 Virtual
    - Clock Speed: 1.4 GHz, up to 2.7 GHz w/ turbo
    - RAM: 8 GB DDR3 1600 MHz
    - OS: OS X 10.12.6
11) Raspberry Pi
    - Processor: 64-bit ARMv8 A53 CPU
    - Cores: 4
    - Clock Speed: 1.2 GHz
    - RAM: 1 GB LPDDR2
    - OS: Raspbian Stretch 4.9

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

[National Institute of Standards and Technology, 2001] National Institute of Standards and Technology (2001). *FIPS PUB 140-2: Security Requirements for Cryptographic Modules*. National Institute for Standards and Technology, Gaithersburg, MD, USA. Supersedes FIPS PUB 140-1 1994 January 11.

[National Institute of Standards and Technology, 2013] National Institute of Standards and Technology (2013). *FIPS PUB 186-4: Digital Signature Standard (DSS)*. National Institute for Standards and Technology, Gaithersburg, MD, USA. Supersedes FIPS PUB 186-3 2009 June.

[National Institute of Standards and Technology, 2015a] National Institute of Standards and Technology (2015a). *FIPS PUB 180-4: Secure Hash Standard*. National Institute for Standards and Technology, Gaithersburg, MD, USA. Supersedes FIPS PUB 180-4 2012 March.

[National Institute of Standards and Technology et al., 2016] National Institute of Standards and Technology, Meltem Sönmez Turan, Barker, E., Kelse, J., and McKay, K. (2016). *NIST SP 800-90b: Recommendation for the Entropy Sources Used for Random Bit Generation*. National Institute for Standards and Technology, Gaithersburg, MD, USA.

[National Institute of Standards and Technology, 2016a] National Institute of Standards and Technology, E. (2016a). *NIST SP 800-57 Part 1 Revision 4: Recommendation for Key Management*. National Institute for Standards and Technology, Gaithersburg, MD, USA.

[National Institute of Standards and Technology, 2015b] National Institute of Standards and Technology, Elaine Barker, J. (2015b). *NIST SP 800-90a: Recommendation for Random Bit Generator (RBG) Constructions*. National Institute for Standards and Technology, Gaithersburg, MD, USA.

[National Institute of Standards and Technology, 2016b] National Institute of Standards and Technology, Elaine Barker, J. (2016b). *NIST SP 800-90c: Recommendation for Random Bit Generator (RBG) Constructions*. National Institute for Standards and Technology, Gaithersburg, MD, USA.

[National Institute of Standards and Technology, 2012] National Institute of Standards and Technology, Q. (2012). *NIST SP 800-107 Revision 1: Reccomendation for Applications Using Approved Hash Algorithms*. National Institute for Standards and Technology, Gaithersburg, MD, USA. Supersedes FIPS PUB 800-107 2009 February.