Allegheny College
Department of Computer Science

Computer Science 202 - Algorithm Analysis

Final Project
Spring 2021

Cryptography

By: **Michael Ceccarelli**[1]
and **Rachael Harris**[2]

Under the Advision of Professor: **Aravind Mohan**[3]

Ref: , Second-year Computer Science and Mathematics Double Major and Economics Minor, Allegheny College: [1]
Ref: , Second-year Computer Science Major and Political Science and Philosophy Double-Minor, Allegheny College: [2]
Ref: , Assistant Professor of Computer Science, Allegheny College: [3]

Meadville, PA

May 12, 2021

Contents

Cryptography

Michael Ceccarelli and Rachael Harris

May 2021

1 Motivation

Our motivation for this project begins with the real-world applications of RSA encryption. RSA, which stands for Rivest, Samir, and Adelman, the three original authors of this system, is a cryptographic system involving the pairing of keys. These keys can be either public and called an encryption key, or private and called a decryption key. Ultimately, the security of this system is dependent upon how private the private key remains, while the public key can be shared without compromising security.

In industry, this system is used widely for data encryption of email and other digital transactions on the Internet. This algorithm is still used today with the standard of 2,048 total RSA keys. While the growth of key sizes has increased, the amount of security provided by them is not correlative to the amount of power required by the computer to process them.

Thus, when RSA is used to provide proper encryption, the server can provide confidentiality.

Therefore, we believe that the algorithm that we attempted to extend is useful, as this encryption solves the problems of key distribution and key security.

2 Background

2.1 RSA History

RSA was introduced by MIT colleagues Ron Rivest, Adi Shamir, and Leonard Adleman in 1977. The name for this algorithm comes from the surnames of these three colleagues, who first publicly described it.

The spark to their research came from the innovation of public-key cryptography (PKC), which was brought to fruition by Whitfield Diffie, Martin Hellman, and Raph Merkle in 1976. Rivest, Shamir, and Adleman sought to develop a cryptosystem that would enable secure message encoding and decoding between communicating parties. They divide and conquered the work among the three of them; Rivest and Shamir began working to develop an unbreakable key system while Adelman tried to break each one. Throughout this process, Adelman broke the system 42 times before he achieved success.

Before the invention of RSA, previous methods of encryption required securely-exchanged keys to encrypt and decrypt messages. With the invention of RSA, encryption and decryption could be done without both parties needing a shared secret key. The added benefits of RSA were that the algorithm could also mark messages with a digital signature and that originators could create messages intelligible only to their intended recipients. As such, any third parties intercepting these transmissions would find them indecipherable.

2.2 RSA Operation

In RSA, there are two types of keys, which is why the algorithm is known as asymmetric key cryptography. These two keys are known as encryption, or public keys, or decryption keys, or private keys.

The encryption keys are based on two large prime numbers and an auxiliary value. How secure the RSA is depends on the practical difficulty of factoring the product of two large prime numbers, which is known as the ”factoring problem.” Also, it is important to note in this discussion that the two prime numbers are kept a secret. Messages can be encrypted by anyone, via the public key, but can only be decoded by someone who knows the prime numbers.

3 Structured Overview

3.1 Proposed Implementation

RSA is a type of encryption that has been around for decades, and is still in use today with many applications. Named after its founders' initials, RSA showcases a problem humans currently do not have an understanding of: factoring large numbers. Forty years later, we are still stuck on the same problem, making RSA still relevant today.

Of course, our implementation would not actually be used by anyone for the purposes of encrypting messages, and all the operations will be found in the terminal window. The primary idea is to have a conversation between Alice and Bob, the two typical characters in any cryptography problem, and they will encrypt and decrypt their messages to each other using the RSA algorithm. Our current goal is to have a message typed in English, converted into an integer, encrypted, decrypted, then converted back into English. Currently, we plan to leave out Eve, the character always responsible for trying to intercept messages in typical cryptography examples, but if we have time to implement her, it will be interesting to see just how hard it is to crack RSA.

There are several algorithms we wrote, all of which are heavily rooted in complex mathematics dealing with prime numbers and modulus division. Our goal, which we successfully achieved, was to write pseudocode for the various algorithms, and implement them in Java afterwards. There is a class that holds all the methods to do these operations, serving as the core of the whole program.

However, it is important to note that in actual applications, RSA requires numbers that are simply beyond the scale of what we are able to do in a Java program. The current standard is 2048 bit prime number, or one that is around 600 digits long. This is impossible to comprehend, and using numbers of that size require programs that are specialized to do so, like Wolfram Mathematica. Our primes will be chosen on a much smaller scale, so even though they won't technically be secure by any means, they will work for our example.

By the end of the project, we have incorporated a solid program that utilizes RSA algorithms to encrypt messages that someone can type into the terminal window.

3.2 Proof

(Euler's Theorem) Take two prime numbers, p and q. Let g be an integer,

$$g = gcd((p-1),(q-1)).$$

Then,

$$a^{(p-1)(q-1)/g} \equiv 1 \pmod{pq}, gcd(a,pq) = 1.$$

Proof. Through basic principles of modulus division, we can safely assume that g divides $p-1$, so $(p-1)/g$ has to be an integer, and that p does not divide a, so

$$a^{(p-1)(q-1)/g} = \left(a^{(p-1)}\right)^{(q-1)/g}$$

Thanks to Fermat, we know that $a^{(p-1)} \equiv 1$, so we are able to replace that expression, written obviously above, with simply a 1. From here, the proof becomes very simple.

$$\equiv 1^{(q-1)/g} \pmod{p}$$
$$\equiv 1 \pmod{p}$$

$\square$

3.3 Successive Squaring

This is the algorithm to compute the large numbers (encryption keys) for the RSA algorithm that will aid in its security.

Algorithm: Successive Squaring

Input: A base, power, and a modulus

Output: A to the power of B mod M

```
Initialize $highestPower \leftarrow 0$
for $i = 0$ to 20 do
    if $powerOf(2, i) > B$ then
        $highestPower \leftarrow i$
    end if
end for
for $i = highestPower - 1$ to 0 do
    if $powerOf(2, i) <= B$ then
        $B \leftarrow B - powerOf(2, i)$
        $D[i] \leftarrow true$
    else
        $D[i] \leftarrow false$
    end if
end for
$M[0] \leftarrow A$
for $i = 0$ to $|M|$ do
    $M[i] \leftarrow powerOf(M[i-1], 2)\ \%\ n$
end for
Initialize $total \leftarrow 1$
for $i = 0$ to $|D|$ do
    if $D[i]$ then
        $total \leftarrow total * A[i]$
        $total \leftarrow total\ \%\ n$
    end if
end for
return $total$
```

3.4 Power Calculator

This is the algorithm that will allow us to stay within the scope of the Java programming language.

Algorithm: Power Calculator

Input: A base and a power.

Output: A to the power of B.

```
Initialize $total \leftarrow 1$
for $i = 1$ to $power$ do
    $total \leftarrow total * base$
end for
return total
```

3.5 Euclidean Algorithm

This is the algorithm to compute the greatest common divisor of two integers.

Algorithm: Euclidean Algorithm

Input: Two integers A and B

Output: Their greatest common divisor

```
Initialize $r \leftarrow a$
Initialize $x \leftarrow 0$
while $r <> 0$ do
    $x \leftarrow a/b$
    $r \leftarrow a\%b$
end while
if $r = 0$ then
    return $b$
else
    $a \leftarrow b$
    $b \leftarrow r$
end if
return $b$
```

4 Results

4.1 Example

Bob and Alice are trying to exchange messages on an insecure server. They are going to use RSA encryption to do so.

Bob chooses two large primes, p and q. This time, he chooses $p = 17$ and $q = 19$. Bob then computes a number that is hard to factor, N,

$$N = pq = 17 * 19 = 323.$$

He then needs an exponent, e, which he will make public. He chooses $e = 65$. The condition for this number is

$$gcd(e, (p-1)(q-1)) = gcd(65, 288) = 1.$$

Alice then turns her message into an integer that is less than N, 243. Using Bob's published N and e, Alice computes c, which looks like this

$$c \equiv m^e \pmod N, c \equiv 243^{65} \equiv 22.$$

Bob takes her ciphertext and, because he knows $(p-1)(q-1) = 288$, he is able to compute

$$ed \equiv 1 \pmod{288}, d * 65 \equiv 1 \pmod{288}.$$

He finds that $d = 257$. Now he computes the message from the formula

$$c^d \pmod N, 22^{257} \equiv 243 \pmod{323}.$$

4.2 Analysis

We did a run-time analysis on all of our algorithms, noting in the various reflection documents that our EEA still needs improved.

$$SuccessiveSquaring : O(n) \tag{1}$$

$$PowerMethod : O(n) \tag{2}$$

$$EuclideanAlgorithm : O(logn) \tag{3}$$

$$ExtendedEuclideanAlgorithm : O(n^2) \tag{4}$$

5 Conclusion

5.1 What We Learned

First and foremost, the actual algorithms needed to make RSA work are fully implemented and tested. You can find the pseudocode for all of them in the Structured Overview section. After writing this psudeocode, we turned the algorithms into Java code, with a little fine tuning for the actual implementation, as it is not a straight shot from pseudocode to high-level programming. Fortunately, they all seem to be in working order, as they have been tested thoroughly.

In this way, Michael's experience in the Cryptography course in the Mathematics Department at Allegheny College is extremely relevant. Here, he was able to transfer the skills that would help him write the three crucial methods for RSA cryptography. He was able to synthesize this knowledge with the experience in this Algorithms Course to write pseudocode for the algorithms for our RSA system.

When writing the pseudocode for the Successive Squaring method, we learned to work with a different algebraic system, modulus. When working with modulus, we learned that operations that seem way too big to compute are made possible by keeping the numbers within the scope of the problem. We learned to construct it carefully, as to raise large numbers to large powers successfully, it needed to be done with precision. We also learned to take a different approach than what might be intuitive or natural in this case. So, instead of getting a large number and then performing modulus division on it, we broke the multiplication up into steps that are manageable. This keeps the work from being done by the computer and, by extension, keeping it from taking up a lot of computing power.

When writing the pseudocode for the Euclidean Algorithm, we learned the fundamental idea of cryptography. This algorithm is essentially an efficient way to find the greatest common divisor for two numbers. While we do not directly use the ideas of this within our program, we were able to take the skills that we learned from writing this pseudocode to give us a strong basis for understanding what needed to be used, which is the Extended Euclidean Algorithm. This algorithm is an extension of the Euclidean Algorithm in such a way that it not only computes the greatest common divisor of two integers, a and b, but also the Bezout's identity, which are integers, x and y, such that *ax + by = gcd(a,b)*. This algorithm is a certifying algorithm, because the greatest common divisor is the only number that can simultaneously satisfy this equation and divide the inputs. It allows us to compute also, with almost no extra cost, the quotients of a and b by their greatest common divisor.

Finally, when writing the powerOf method, we learned the scope of the Java programming language. We learned that the language has a limitation for how large an integer can be, which means that when writing our RSA algorithm, we had to be very careful not to go past the scope of what Java could do, as not to give it too large of numbers. Moreover, powerOf is often used in successiveSquaring, which also gave us some challenges, as it forced us to pick from a pool of prime numbers that are much smaller and more restricted than we initially intended, but it did not take away from the core purpose or functionality of the program.

5.2 Challenges

One of the biggest challenges that we faced was getting the classes to communicate correctly. For the majority of our time working on this project, we had all of the classes in the same folder with the same package declaration, and could not figure out what exactly we were doing wrong. We guessed and checked and troubleshot with different resources available to us. Eventually, we met with Dr. Mohan to discuss this challenge and were able to overcome it by using the proper command, listed in the README, to run the program.

After we were able to get the classes to communicate with each other, we were able to begin debugging the program. While RSA is unbreakable, it is vulnerable and we were determined to find its vulnerabilities.

After facing that technical challenge, we challenged ourselves to add additional functionality to the program.

The first thing that we challenged ourselves to add was encrypting the public numbers, so that when Alice or Bob publish a number on the insecure channel, our program does not just show the number, but rather, a random sequence of characters that only the other knows how to decrypt. With this challenge, our goal is to add a second layer of encryption, for a more realistic effect. While this does not give the program any actual credibility in relation to being classified as a real cryptosystem, and while Alice and Bob have essentially no reason to be encrypting their messages, the idea is that we are simulating an environment in which they would communicate through an insecure channel and demonstrating what that would be like.

The second thing that we challenged ourselves to add was a better implementation for the Extended Euclidean Algorithm. Initially, it worked (and technically, at that), but it was very inefficient and time-consuming. We were satisfied with this at first, as we were happy to have something that worked. However, taking the time to formulate a better solution was a nice way to finish up the program, especially with a more advanced process for exchanging messages in the terminal window.

Finally, we challenged ourselves to calculate the algorithms' runtimes. This helped us in making the Extended Euclidean Algorithm more successful, as it had the worst run time. However, overall, this would aid in our success with the project, as it would allow us to understand the complexity of these algorithms and the project as a whole.

5.3 Rewards

As you could probably guess, one of our most immediate and biggest rewards was getting the classes to correctly communicate with each other. This was one of our biggest challenges and was causing us the most frustration. In remaining completely transparent, it was something that the both of us knew that we should know how to do, but could not figure it out. It was almost impressive that we could write an entire encryption algorithm, but could not figure out how to get the classes to communicate with each other. We were surprised that something as little as that could trip us up so much when working on this project. However, with the help of Dr. Mohan, we were able to come up with a solution to this problem! Once it was fixed, we were able to continue working on our RSA encryption, begin debugging, and address the challenges that we wanted to work on.

After that, our rewards came from meeting the goals that we had set for ourselves earlier; expanding on the Extended Euclidean Algorithm, making a more complex terminal window, and encrypting the public keys. These all carried with them their own rewards in different ways, but they were all incredibly exciting to see.

Ultimately, our biggest reward was running the program successfully for the first time and watching the program encrypt and decrypt messages in real time!

6 Bibliography

Smart, Nigel (February 19, 2008). ”Dr Clifford Cocks CB”. Bristol University.

Rivest, R.; Shamir, A.; Adleman, L. (February 1978). ”A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. Communications of the ACM.

Calderbank, Michael (2007-08-20). ”The RSA Cryptosystem: History, Algorithm, Primes”

Cooks, C.C. (20 November 1973). ”A Note on Non-Secret Encryption”. GCHQ.

Hirsch, Frederick J. ”SSL/TLS Strong Encryption: An Introduction”. Apache HTTP Server.

Ferguson, Niels; Schneier, Bruce (2003). Practical Cryptography.

Mit Colleagues. (N.D.). ”Leonard Adleman Made RSA Encryption History With His Invention.”

Kahn, David (1996). The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet. Simon and Schuster.