Simulation and Analysis of Various Attacks on the RSA Cryptosystem

Heather DeVal, Jordan Fok, Hannah Russell
University of Maryland, Baltimore County
Department of Computer Science and Electrical Engineering
CMSC 443

***Abstract*—This paper examines the RSA Cryptosystem and analyzes a few of the attacks that can be used against it. When attacking a cryptosystem it is possible to attack it by looking at the scheme and the mathematical problem behind the algorithm. Conclusions on the cryptosystem are drawn from both coded implementations and previous research. For exploiting the cryptographic scheme, the blinding attack is implemented and analyzed. For exploiting the mathematical problem, the attacks of factoring the public key and Wiener attack are analyzed. For each attack, the paper specifically looks at the difficulty of implementation, along with the reality of execution and the time required for the attack to be completed with a variety of chosen security parameters. The paper will also examine the ethics behind the RSA Cryptosystem and the attacks used against it.**

I. Introduction

The RSA Cryptosystem was first introduced in 1977 by Ron Rivest, Adi Shamir, and Len Adleman. When implemented correctly the cryptosystem is able to provide authenticity of digital data and privacy [1]. RSA has been used in emails, web browsers, credit card transactions, network traffic, remote sessions, and other implementations that make this cryptosystem still relevant today [1]. It is also one of the first cryptosystems to successfully use public-key cryptography and allow secure digital-signatures [3]. A depiction of how pubic-key cryptography works can be found in Figure 1.

The security of the RSA cryptosystem relies on the fact that it is difficult to factor the product of two large prime numbers, p and q. A basic RSA algorithm has the following steps:

- **Key Generation**: Key generation provides both the private and public key needed for the algorithm by using the following steps:
  - Calculate $n = pq$, where p and q are large prime numbers.
  - Compute $\phi(n) = (p-1)(q-1)$ and choose an integer e, where $\gcd(\phi(n), e) = 1$ and $1 < e < \phi(n)$.
  - Calculate the integer d using the equation $de \mod \phi(n) = 1$.

  The public key is the pair $\{e, n\}$ and the private key is the pair $\{d, n\}$.
- **Encryption**: The RSA encryption scheme uses the equation:

$$C = M^e(\mod n),$$

  where M is the plaintext, C is the ciphertext, and $M < n$.

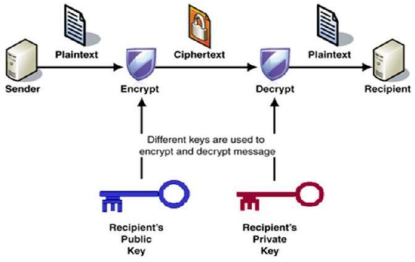

Fig. 1. This diagram explains the theory behind public key cryptography, which is used for the RSA cryptosystem. In public key cryptography, there are two keys: the public key and the private key. Anyone can view the public key and it is used to encrypt the message. The private key is only held by the recipient of the message and is used for decryption.

- **Decryption**: When decrypting, RSA uses the equation:

$$M = C^d(\mod n),$$

  in order to reveal the original message sent.

Since the encryption and decryption operations shown above are inverses of each other that means it is possible to perform the operation in reverse order in order to obtain a digital signature [2]. A message can be signed by using the following operations:

$$S = SIGN(M) = m^d \mod n.$$

Similarly, the message can be verified with the following:

$$M = VERIFY(s) = S^e \mod n.$$

RSA algorithms have been implemented in a variety of different languages, for the testing and research purposes of this project it was implemented using python and was able to complete all of the necessary steps previously listed.

The RSA implementation can be attacked in two different ways. In Section II, the approach of attacking the cryptographic scheme will be analyzed through the implementation of the blinding attack. In Section III, the approach of attacking the algorithm and mathematical problem are examined. This mathematical exploit will be demonstrated through the attacks factoring the public key and Wiener (guessing d) attack. Finally, Section IV will discuss whether or not the attacks on the RSA cryptosystem are deemed ethical.

II. Attacking the Cryptographic Scheme

The RSA scheme itself can not be considered completely secure. No matter how the security parameters of n, d, and e are adjusted, it is still possible to successfully retrieve confidential information.

A. Blinding

The RSA cryptosystem involves a signing process that allows for secure communication between parties by utilizing digital signatures. The blinding attack works by allowing the attacker to get a signature for a message from a party without the party viewing the true contents of the message. This attack is similar to signing a blank check without being able to see the amount it is for. Later verification checks will show the message and signature from the attacker to be genuine. The blinding attack process consists of the following steps:

- Blinding Phase: The attacker "blinds" a message by using a random blinding factor. The following equation is used to blind the message:

$$M^{'} = r^{c}M \mod n,$$

  where $M^{'}$ is the blinded message, r is the blind factor, and e and n are the variables previously discussed in the RSA algorithm.
- Signing Phase: The attacker sends the blinded message to the receiving party and requests them to sign it. The receiving party does not know any better, sees the message as harmless, signs it, and sends it back to the attacker. The following equation is used to sign the blinded message with the private key:

$$S^{'} = (M^{'})^{d} \mod n,$$

  where S' represents the sent signature and d is the private key exponent.
- Unblinding Phase: Finally, the attacker removes the blinding factor from the message and obtains the signature of the original message using the following calculations:

$$\frac{S^{'}}{r} = \frac{(Mr^{e})^{d}}{r} = \frac{M^{d}r^{ed}}{r} = \frac{M^{d}r}{r} = M^{d} \mod n.$$

The code for the blinding attack was created using the language python. The MD5 hash and hashlib library are used to create the blinded message, which is then sent to the party that holds the private key. After the party signs the message using the private exponent d, it sends the attacker it's public signature. The attacker then uses the python library libnum and the method above to remove the blinded factor and obtain the signature of the original message.

1) Time Required for Attack: When testing this attack a range of relatively small security parameters were used for e, d, and n. The blinding factor, r, was also relatively small and was set equal to 21 for each test. With the chosen security parameters the time the test took to complete varied from 0.01 seconds to 0.02 seconds.

2) Difficulty for Attacker and Reality of Implementation: A blinding attack is fairly simple for an attacker to implement. It relies on the party with the private key to unknowingly sign the blinded message from the attacker. Even if larger security parameters were chosen, the time for the attack to complete will not vary significantly. The blinding attack is really general and can easily work for a larger n that is around 2048 or 5096 bits. The attack works for these n values because the underlying signature scheme is not secure. One prevention that can be used to make the signature scheme more secure would be to use RSA-FDH instead of a plain RSA signature. Since it is simple to implement and does not take much time, it is quite possible that attackers will implement this attack to gain information. This attack further proves that the RSA cryptographic scheme is not completely secure.

III. Attacking the Mathematical Problem

For attacks on the mathematical problem behind the RSA cryptosystem, often times precautions can be taken in the security parameters to make the algorithm more secure and prevent an attack. In the following attacks, the thresholds of the security parameters were tested in order to determine when an attack with a mathematical approach will be successful.

A. Factoring the Public Key

The RSA cryptosystem relies on integer factorization for generation of the public keys. If the attacker can find the prime factors used in the RSA process, the attacker can compute the exponent d from any party's public key. Now that the attacker has the exponent d, they can decrypt the ciphertext. The factorization process is the following:

- Factor n: If n can be factored successfully, the two prime integers, p and q are obtained.
- Compute ϕ: Using the factors, p and q, ϕ can be computed by multiplying $(p-1)(q-1)$.
- Compute d: Now with ϕ the private key exponent, d, can be found with the equation:

$$e^{-1} \mod (p-1)(q-1) = d.$$

The code for the attack was completed in python and used the library libnum to calculate the private key. There are a few methods that can be used to factor the public key, including quadratic residues $\mod n$, elliptic curve factorization, difference of squares, and Pollard's method. The method chosen for this code was difference of squares because it is the simplest implementation that still works efficiently. The code utilizes a while loop to try all possible numbers and returns once the factors have been found. Finding the factors using difference of squares starts with the equation:

$$x^{2} - y^{2} = (x-y)(x+y).$$

The value of n is represented as:

$$n = x^{2} + y^{2}, \tag{1}$$

which means the factors of n will be $(x+y)$ and $(x-y)$ [5]. If equation 1 is rearranged, a new equation can be formed:

$$n + y^2 = x^2.$$

If the result of n and the values of y is a square, then the two factors, p and q have been found [5].

1) Time Required for Attack: When n was less than 512 bits, the time to complete the attack on average was 0.0003 seconds. With an n larger than 512 bits, the attack becomes infeasible because it will take longer than the attacker's lifetime for the factoring to complete.

2) Difficulty for Attacker and Reality of Implementation: Implementing this attack is moderately difficult because it involves mathematical computations and bitwise operations. Strong RSA cryptosystems will use an n that is larger than 2048 bits, making this attack unsuccessful for attackers. The program implemented for this project used the approach of trying all possible factors one at a time. However, there is research that shows there are methods out there that would take significantly less time and would take on the order of $\sqrt{n}$ steps [2]. Factoring a 1024-bit number would take approximately a year on a machine that would cost millions of dollars [2]. The reason that it is recommended to use a 2048-bit number is because it will require several billion times more work than factoring a 1024-bit number. The length that a factoring attack would take when this security parameter is used makes factoring the public key an infeasible option for many hackers. Inexperienced hackers are even more unlikely to attempt this attack due to the required bitwise operations, but more advanced hackers may attempt it and see if the system is using a small n.

B. Wiener Attack

In the wiener (guess the d) attack, the attacker attempts to find the private exponent d from the public n and e. The attack uses the continued fraction algorithm to guess d and break the encryption. Continued fractions are used to find the numerator and denominator of a fraction when an estimate of the fraction is known [4]. In the case of RSA, the public exponent and modulus is used to construct an estimate of a fraction that involves the private exponent. [4].

The code for this implemented attack was completed in python and an adaptation of previous code to execute the continued fractions.

1) Time Required for Attack: When wiener attack was successful in finding the exponent d, it took approximately 0.002 seconds to complete. This time was calculated on a program that used both a large n and a large e. In this case the private exponent that was found was relatively small. When the attack failed to find the private exponent, it took approximately 9 seconds. By running various tests, we were able to verify the theorem that the attack only works when the private exponent d is less than $n^{\frac{1}{4}}$.

2) Difficulty for Attacker and Reality of Implementation: This attack is moderately difficult to implement. It involves multiple mathematical computations and requires an understanding of continued fractions. Considering that the attack can be avoided by using a d that is larger than $n^{\frac{1}{4}}$, if the attack is used it is unlikely to be successful. Hackers still may attempt to try this attack to see if the system is outdated and is not using the proper security parameters to protect their confidential information.

IV. Ethical Analysis

The general rule of thumb for ethical hacking is to not hack people unless you have been given explicit consent to do so. Often times hacking can result in consequences that may include jail time. The attacks in this paper should not be used to hack others for any personal, monetary, or political gain. They can be used to test your own cryptographic systems and make sure that your information is secure. This would be considering ethical hacking and this field is very important in the Defensive Security Operations.

V. Conclusion

In conclusion, the RSA cryptosystem can not be deemed as completely secure. The system is vulnerable to attacks on both the mathematical algorithm and on the cryptographic scheme. While there are precautions a coder can make to protect against attacks on the mathematical algorithm, the RSA cryptosystem is still vulnerable to attacks on the scheme itself. This means that if a coder does decide to use the RSA cryptosystem to send messages, they should proceed with caution and make sure they have the proper security parameters in place. Hackers will still be able to gain information on the messages sent by using attacks that are only moderately difficult for them to create. There are also newer variations of the RSA cryptosystem that may make the cyrptographic scheme more secure and could be looked into in future work.

References

[1] D. Boneh, "Twenty Years of Attacks on the RSA Cryptosystem," Stanford University, pp. 1–16.

[2] B. Kaliski, "The Mathematics of the RSA Public-Key Cryptosystem," RSA Laboratories, pp. 1–9.

[3] E. Milanov, "The RSA Algorithm," Washingtown University, June 2009.

[4] M. Wiener, "Cryptanalysis of Short RSA Secret Exponents", BNR, August 1989.

[5] B. Buchanan, "Defining the Limits of RSA Cracking", August 2018.