Proceedings of the 2023 International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/36/20230445

Probing RSA's weak spots: Dissecting timing attacks and Euclidean method exploitations

Siwei Tan[1] and Siyuan Tong[2, 3]

[1]College of Engineering, University of California, Davis, 95616, USA
[2]Facility of Applied Sciences, Macao Polytechnic University, Macao, 999078, China

[3]p2111377@mpu.edu.mo

Abstract. The RSA encryption system, a cornerstone of numerous cryptographic frameworks, has long enjoyed a reputation for robustness. However, its strength is inherently tethered to the meticulous selection and management of its foundational prime numbers, p and q. This study delves into a nuanced vulnerability that surfaces when p and q assume particularly large values. Within this context, we illuminate how the Euclidean method can be weaponized to swiftly decipher RSA-encrypted messages, unveiling the original plaintext with surprising efficiency. Intriguingly, our analysis also uncovers that harnessing parallel computation for the Euclidean method expedites decryption exponentially, accentuating this vulnerability. Such revelations cast a spotlight on a delicate balancing act between computational prowess and cryptographic fortitude. The insights gleaned from our research emphasize the paramount importance of judicious prime selection in the RSA framework. They also caution about the unforeseen pitfalls that might lurk behind algorithmic enhancements in cryptographic contexts. Through this investigation, we aspire to catalyse a critical re-evaluation of RSA's real-world deployments and champion a more circumspect, continually adaptive approach to designing cryptographic systems.

Keywords: Parallel Computing, RSA, Euclidean Algorithm, Timing Attacks, Cryptographic Security.

1. Introduction

Within the intricate tapestry of modern cryptographic systems, the RSA algorithm stands as a beacon of security. Yet, as we navigate the swift currents of technological evolution and escalating computational power, the imperative for perpetual scrutiny and refinement of RSA's safeguards becomes ever more pressing. Boneh's seminal work in 1999 charted a two-decade-long trajectory of attacks targeting the RSA framework, underscoring an unyielding necessity for vigilance and evolution in cryptographic paradigms [1]. At the heart of RSA's machinations lies the Euclidean algorithm, whose subtle intricacies play a pivotal role in shaping the encryption's overall resilience. A recent exploration by Lizy in 2021 shed light on potential enhancements to the RSA paradigm via adept utilization of the Euclidean technique, carving pathways to heightened security coupled with streamlined efficiency [2]. While existing scholarship has traversed myriad methodologies, a persistent quandary remains the pursuit of optimally parallelizing the Euclidean algorithm in a manner that amplifies efficiency yet remains staunchly unyielding to security breaches [3]. Further, a palpable void exists concerning a holistic

DOI: 10.54254/2755-2721/36/20230445

juxtaposition of diverse parallelization approaches—be it OpenMP, Threads, or Sequential—in relation to their real-world efficacy and scalability.

Our present endeavour dives deep into this chasm, rigorously examining the parallelized Euclidean algorithm within the context of RSA assaults [4]. Our aspiration is to distil for our audience an unambiguous blueprint of the premier parallelization trajectory, derived from meticulous empirical evaluations and discernment. Concurrently, we seek to cast a spotlight on lurking security vulnerabilities. Through this scholarly lens, we aim to both catalyse future inquiries into parallel processing's symbiosis with cryptography and furnish invaluable insights for the pragmatic deployment of RSA in our digital age.

2. Basics about RSA

The research begins by describing the basics of RSA. RSA is a widely used public-key cryptosystem, it utilizes the difficulty on factorization of large composite numbers. It is an asymmetric cryptographic algorithm [5].

2.1. Basic concept

In the encryption and decryption process of RSA, they cannot avoid these concepts. Euler's Totient Function and Euler-Fermat Theorem. The Euler Totient Function of n $\in$ N, denoted by $\phi(n)$, is the number of integers $k \in [1, n]$ [6] satisfying $gcd(n, k) = 1$. The value of $\phi(n)$ can be formally computed by the following:

$$\phi(n) = n \prod_{p|n} (1 - \frac{1}{p})$$

Euler-Fermat Theorem function of every positive $a, n \in \mathbb{Z}$, if $gcd(a, n) = 1$, then a ϕ(n) $\equiv$ 1 mod n. There are several references to the greatest common divisor (gcd). The gcd that for every number non-zero $a, b \in \mathbb{Z}$, the greatest common divisor of a, b, denoted by $gcd(a, b)$, is the largest integer $d \in \mathbb{Z}$ such that $a = ed$ and $b = fd$ for some $e, f \in \mathbb{Z}$. The research also used their Congruence, for every number $a, b \in \mathbb{Z}$, and for some $n \in \mathbb{Z}$, a and b are congruent under modulo n, written by $a \equiv b$ mod n, if and only if there exists some $k \in \mathbb{Z}$ such that $a = kn + b$.

2.2. RSA Cryptosystem

2.2.1. Key Generation. Suppose Alive wants to send a message $m \in \mathbb{Z}$ to Bob. Bob will set up the RSA cryptosystem via the following process: Pick two large primes p, q and compute $n = pq$ [7]. Pick an integer $d < \phi(n)$ such that $gcd(d, \phi(n)) = 1$. Find the multiplicative inverse e of d under modulo n, i.e., $ed \equiv 1 \bmod \phi(n)$ [8]. Publish (n, e) as public keys, and keep (p, q, d) as private keys.

2.2.2. Encryption and Decryption Process. If Alice wants to send message m to Bob, she encrypts m to the cipher c by the following: $c \equiv me \bmod n$. When Bob receives the cipher c, he decrypts it by the following: $m \equiv cd \bmod n$.

Proof of the Correctness for RSA Cryptosystem. Because $ed \equiv 1 \bmod \phi(n)$, so $ed = 1 + k\,\phi(n)$ for some $k \in \mathbb{Z}$. Therefore, we have

$$c^{ed} \equiv m^{ed} \equiv m^{1+k\phi(n)} \bmod n = m \cdot m^{k\phi(n)} \bmod n = m \cdot 1^k \bmod n = m \bmod n .$$

2.3. Problem Statement

In RSA cryptosystems, the goal of a time attack is to find the private key d for decryption. Before diving into the implementation of our attack, some details of the modular arithmetic algorithm used to optimize the RSA cryptosystem need to be elaborated. Modular arithmetic is expensive, so implementations of

DOI: 10.54254/2755-2721/36/20230445

RSA cryptosystems have developed several ways to speed up this arithmetic. In this case, the focus is on the Montgomery index [9].

Montgomery index operation is an efficient method to perform index operation on a finite field. It takes advantage of the factors contained in the exponent to speed up the operation by reducing the number of multiplications of the exponent. In RSA cryptosystems, people usually use modular exponentiation to calculate encryption and decryption operations. To improve efficiency, the Montgomery index computation can be used instead of the standard index computation [10].

In order to better understand the Montgomery index calculation, some basic concepts need to be known. Over a finite field, one can use the prime numbers p and q to represent the moduli $n = pq$. Exponential operation e mod n denotes the operation of e modulo n. For example, e mod $n = ek$ mod n, where k is an integer less than n and prime to n. Now back to the topic, how Montgomery index arithmetic optimizes modular arithmetic in RSA cryptosystems. The Montgomery index operation improves efficiency by reducing the number of multiplications of the index. In RSA cryptosystems, people usually use a smaller exponent r to compute the modular power operation d mod n. The research team gradually approach the final d value by repeatedly performing modular exponents. In the later context, the research team define the followings: Let n, the public key, be a k-bits integer. Let $r = 2k$, with its multiplicative inverse $r - 1$ in $\mathbb{Z}_n$, i.e., $r \cdot r^{-1} \equiv 1 \bmod n$. Let n' be an integer satisfying $r \cdot r^{-1} - \mathrm{n} \cdot n' = 1$ Then, our optimized Montgomery multiplication and exponentiation operations can be introduced. In the MontMul function there's a subtraction $u - n$, this is called extra reduction. The goal of timing attack is to detect the time difference caused by this extra reduction and guess the private key done bit by one bit.

2.4. Methodology

The experiment used the algorithm above to perform a timing attack. The attack starts in the while loop on line 4-23. The attack is first carried out on the private key d bits to generate a set of random messages S, and then RSA decryption is simulated for each element m ∈ S and the duration of decryption is recorded. Then, we divide S into two groups according to the time of decryption, modifies the guessed key according to the average time of those two groups. Finally, the guessed key is tested to see if it is the same as the private key d. To speed up the attack, we set up threading routines using pthread and OpenMP. In each routine, we perform random key generation, decryption simulation, and set up partitions. When all the routines are completed, the data are collected and the average is calculated. With this parallelization method, the private key d can be found much faster, thus improving the efficiency of the attack. This attack strategy is very targeted to the RSA encryption algorithm and can break the private key in a relatively short time.

2.5. Result

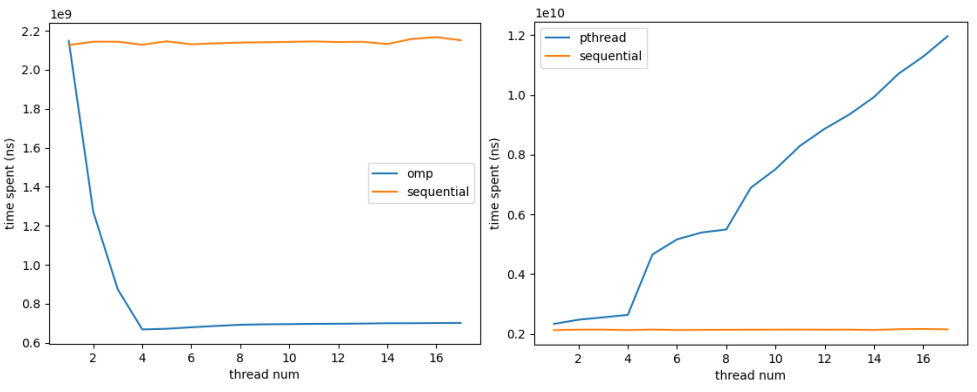

Figure 1. Experimental Result Graph (Photo/Picture credit: Original).

DOI: 10.54254/2755-2721/36/20230445

To obtain the experimental results, time was measured using *chrono* library to get the interval time, and the average running time was calculated by running the Euclidean method in different numbers of threads 100 trials. To explore parallelization, the research started with pthread. It is observed that as the number of pthread increases, the running time of the program also shows an upward trend. To get a deeper understanding of this phenomenon, the research team used the gperftools tool for performance analysis. Through analysis, it is found that this behavior is mainly due to the fact that most of the resources are wasted when passing parameters to the thread function. As shown in Figure 1.

In addition, the research team experimented with parallelization using OpenMP. Unlike pthread, parameter passing can be avoided in OpenMP, so our attack performance improves as the number of threads increases. The program runs fastest when the number of threads is set to 4. By comparing the two parallelization methods, it is found that the running time increases as the number of threads increases in pthread. However, in OpenMP, the opposite is true, and the increase in the number of threads actually improves the performance of the program. This discovery gives us an important clue to optimize parallelization and help us improve attack performance. In summary, the OpenMP version can improve program performance more effectively than pthread version in terms of parallelization. By setting the number of pthread reasonably, the running time can be reduced while ensuring the effectiveness of the attack. These findings have important guiding significance for us to further optimize the parallelization strategy.

3. Attack on RSA Based on Euclidean Algorithm

3.1. Euclidean Algorithm Definition

The Euclidean algorithm, a time-tested method rooted in ancient mathematics, offers a systematic procedure for determining two integers' greatest common divisor (GCD). Named after the renowned Greek mathematician Euclid, who documented it in his seminal work "Elements" around 300 BC, this algorithm operates on a foundational principle: The GCD of two numbers also divides their difference. In its modern adaptation, the algorithm employs a more efficient division-based approach rather than relying on repeated subtraction. Given two integers, the larger number is divided by, the smaller, and the remainder then replaces the smaller number. This process is iterated until a remainder of zero is achieved. At this point, the divisor from the preceding step is identified as the GCD. The Euclidean algorithm's elegant simplicity has remained a cornerstone in number theory and computational mathematics.

3.2. Methodology

Given two positive integers, a and b, where $a > b$. Compute the remainder of a divided by b. Replace a with b and b with the remainder obtained in the previous step. Repeat the process until b becomes 0. The non-zero remainder is the GCD of a and b. If the GCD is not 1, then the numbers are not co-prime, which can be a potential vulnerability in RSA if the numbers are part of the public key.

3.3. Implications for RSA Attack

The RSA encryption algorithm is based on the selection of two large prime numbers, the product of which forms both the module of the public key and the module of the private key. The security of RSA lies in the fact that the generation process of its public and private keys is random and theoretically impossible to calculate. However, if an attacker can efficiently find the greatest common divisor of the public key modulus and a series of numbers, then they may be able to find the non-trivial common factor of those numbers, and thus potentially expose the prime factor of the modulus. This will compromise the security of the RSA key pair. Therefore, understanding the potential vulnerabilities associated with Euclidean algorithms is critical to ensuring the robustness of RSA encryption. Euclid algorithm is a classical algorithm to find the greatest common divisor of two numbers. Its application in RSA encryption enables us to find possible vulnerabilities by solving the public key modulus and the greatest common divisor of a series of numbers. This requires the research team to have a deep understanding of

DOI: 10.54254/2755-2721/36/20230445

the principle and implementation of the Euclidean algorithm so that when the research team encounters a potential vulnerability, it can be found and fixed in time.

In general, for the RSA encryption algorithm, it is usually necessary to pay attention to its potential security vulnerabilities, especially those related to the Euclidean algorithm. Only by understanding and mastering these vulnerabilities, can we better ensure the security of RSA encryption, so as to protect our information security.

3.4. Result

3.4.1. Performance Table of three versions of Euclidean Algorithm Implementation

To get the results, time measurements were performed using the *high_resolution_clock()* function from the *chrono* library, and the average running time was calculated by running each version with different threads 100 times in a loop. To explore and highlight parallelization, the study starts with Thread, contrasted with the most basic serial. It can be observed from Table 1 that as the number of threads increases, the running time of the program also shows a decreasing trend. In addition, the research team conducted parallelization experiments using OpenMP. We found that their running time decreases as the number of threads increases, both in Thread and OpenMP. However, when the number of threads exceeds 25, the running time of threads does not decrease steadily, showing unstable fluctuations. However, OpenMP shows a relatively steady downward trend. This finding gives us an important clue to optimize parallelization, which helps to improve the attack performance. In summary, OpenMP can improve the program performance more effectively than Thread in parallelization. By setting the number of threads reasonably, the running time can be reduced while ensuring the effectiveness of the attack. These findings have important guiding significance for further optimization of parallelization strategies. As shown in Figure 2.

Table 1. Performance table across different thread configurations.

THREADS	TIME_SEQ(s)	TIME_OMP(s)	TIME_THREAD(s)
1	14.2894	21.4223	21.4256
2	14.2894	10.874	10.8385
3	14.2894	7.258	7.17554
4	14.2894	5.45106	5.44056
...	...	...	...
51	14.2894	0.429787	0.549034
52	14.2894	0.493095	0.52779

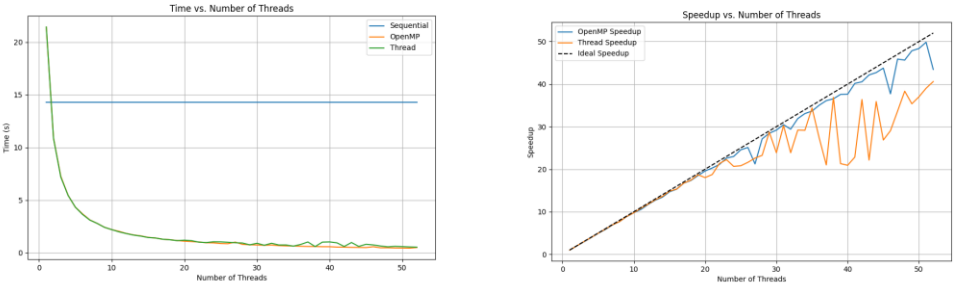

Figure 2. Experimental Result Graph (Photo/Picture credit: Original).

4. Conclusion

In an era marked by rapid advances in cryptography, our intensive research endeavoured to unravel the potential of the Euclidean algorithm in deciphering RSA encryption, scrutinizing it across three distinct paradigms: OpenMP, Threads, and Sequential. A keen examination revealed palpable variations in both scalability and performance metrics across these methodologies. Notably, the OpenMP rendition

DOI: 10.54254/2755-2721/36/20230445

emerged as the clear frontrunner, consistently eclipsing its counterparts in terms of both agility and computational efficiency. This revelation not only magnifies the pivotal role of strategic algorithm selection in applied cryptographic domains but also earmarks OpenMP as an exceptionally promising contender for scenarios where peak performance is non-negotiable. The fluid dynamics of the cryptographic realm demand an unwavering commitment to the regular evaluation and honing of our cryptographic tools. It's paramount that these algorithms stand resilient against emerging vulnerabilities while simultaneously pushing the envelope in computational efficiency. Our findings, shedding light on the pre-eminence of OpenMP, serve as a beacon for forthcoming research endeavours. The challenge lies in diving deeper into the intricate lattice of OpenMP's inner workings, probing its untapped potential, and aligning it further with the dual objectives of fortified security and augmented computational prowess.

Authors Contribution

All the authors contributed equally and their names were listed in alphabetical order.

References

[1] Aldaya, A. C., García, C. P., Tapia, L. M. A., & Brumley, B. B. (2018). Cache-timing attacks on RSA key generation. Cryptology ePrint Archive.

[2] Chen, C., Wang, T., Kou, Y., Chen, X., & Li, X. (2013). Improvement of trace-driven I-Cache timing attack on the RSA algorithm. Journal of Systems and Software, 86(1), 100-107.

[3] Acıiçmez, O., Gueron, S., & Seifert, J. P. (2007). New branch prediction vulnerabilities in OpenSSL and necessary software countermeasures. In Cryptography and Coding: 11th IMA International Conference, Cirencester, UK, December 18-20, 2007. Proceedings 11 (pp. 185-203). Springer Berlin Heidelberg.

[4] Zhu, X., Xu, H., Zhao, Z., & others. (2021). An Environmental Intrusion Detection Technology Based on WiFi. Wireless Personal Communications, 119(2), 1425-1436.

[5] Weiser, S., Spreitzer, R., & Bodner, L. (2018, May). Single trace attack against RSA key generation in Intel SGX SSL. In Proceedings of the 2018 on Asia Conference on Computer and Communications Security (pp. 575-586).

[6] Aldaya, A. C., & Brumley, B. B. (2020). When one vulnerable primitive turns viral: Novel single-trace attacks on ECDSA and RSA. Cryptology ePrint Archive.

[7] Thirumalai, C., Mohan, S., & Srivastava, G. (2020). An efficient public key secure scheme for cloud and IoT security. Computer Communications, 150, 634-643.

[8] Aslam, M., & Bilal, A. (2014). Implementation of rsa algorithm secure against timing attacks using fpga. International Journal of Engineering Research & Technology (IJERT), 1.

[9] Cabrera Aldaya, A., Cuiman Marquez, R., Cabrera Sarmiento, A. J., & Sánchez‐Solano, S. (2017). Side‐channel analysis of the modular inversion step in the RSA key generation algorithm. International Journal of Circuit Theory and Applications, 45(2), 199-213.

[10] Strenzke, F. (2010). A timing attack against the secret permutation in the McEliece PKC. In Post-Quantum Cryptography: Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings 3 (pp. 95-107). Springer Berlin Heidelberg.