Improved RSA Algorithm Counterparts by Exploiting Implementation Attacks

Karim Mohamed, STEM High School for Boys – 6th of October

Adham Ahmed, STEM High School for Boys – 6th of October

Abstract

RSA, being the first asymmetric-key algorithm, has caused an evolution in the science of cryptography, becoming one of the most used encrypting algorithms. These outcomes have resulted in the algorithm receiving a vast number of attacks. One type of these attacks, implementation attacks, poses a crucial threat to RSA. Moreover, due to having other algorithms (e.g., Diffie-Hellman key exchange and elliptic curves algorithm) that are formulated nearly similar to RSA, as well as the mechanisms of the implementation attacks, the dangers of these attacks are found in these other algorithms too. Since the popularity of RSA, most of the research, which provides an improved counterpart of the algorithm secured against this attack, is conducted for RSA. This paper examines two of the most known implementation attacks— timing attacks, and fault attacks— analyzing their methodology and investigating methods to counter them. Finally, the paper evaluates the possibility of having an improved variation of RSA that is feasible to be used in the present.

I. Introduction

Cryptography, the science of establishing techniques to ensure secure communication [1], is well thought-out as one of the human necessities. In fact, the initiation of cryptography is thought to be dated back to approximately 1900 B.C.; some Egyptian scribes formulated a deviant type of hieroglyphs as a form of communication that they only can understand [2].

Throughout history, earlier forms of cryptography were generally used only for military and political purposes, using non-advanced techniques for ciphering messages— yet it was decisive such as the Enigma machine (depicted in **Figure 1**)— requiring a single key to revert the ciphering. However, with the rise of modern machines, cryptography began revolutionizing and started to come out publicly to ensure the privacy of the communication, having the risk of the key being

Figure 1: Symmetric-key Enigma Machine

exposed while being transferred increased, thus another technique for cryptography was needed [3].

In 1976, Whitfield Diffie and Martin Hellman proposed a new method for key distribution. Diffie and Martin prospected replacing the single encryption-decryption key with two keys, one for encrypting, which is transferred and can be known to the public, while the other for decrypting, which is only known by the receiver, calling it asymmetric-key algorithm. Nevertheless, they faced a critical problem: not being able to implement the idea in a working algorithm [4].

A year later, in 1977, Ronald Rivest, Adi Shamir, and Leonard Adleman invented a working asymmetric-key algorithm, naming it RSA for the first letter of their names [3]. The RSA algorithm is recognized as a considerable advancement in cryptography, theorizing the field of modern cryptography and becoming the first asymmetric-key algorithm.

Being considered the most used encryption algorithm, the number of attacks done in an attempt to break the RSA algorithm is numerous. Attacks can be mainly classified into four types: elementary attacks, which are basic attacks that exploit the flagrant nature of the algorithm; low private exponent attacks, which are attacks that can break the systems that use low-value private key; low public exponents attacks, which are, similar to low private exponent attacks, attacks that can break systems that use low-value public key; and finally implementation attacks, which are attacks that aim at the blunders of the implementation of the algorithm [5].

Implementation attacks are the most powerful and complex, since they aim to exploit the system's application to the algorithm. Hence, they are often intricately designated, involving many mathematical attacks [6]. Examples of implementation attacks are timing attacks and fault attacks (which are our sole focus in this paper). Nevertheless, this type of attacks had a great role in the development of the RSA algorithm. As with the proper utilization of the fields of number theory, came better variations and implementations of the RSA algorithm, providing a counterpart for the earlier implementation attacks.

The following paper will describe comprehensively two of the most used implementation attacks: timing attacks and fault attacks, thoroughly explaining their idea and implementations with going through their counterparts and the number theory behind them.

II. RSA Algorithm

i. Overview

RSA is an asymmetrical encryption algorithm, meaning two keys are generated and used. The first key, the private key, is kept secret by the person generating the keys. The second key, the public key, can be freely distributed to anyone. When data is encrypted by one key, only the other key can decrypt it. For example, if I have my RSA keys set up and you have my public key, you can send me a message that's encrypted with my public key and only I can decrypt the message (because I hold my private key). Similarly, it can be used to prove authenticity. If I have a message that I want to prove is from me I can encrypt it with my private key. Now, anybody who has my public key can prove that I sent the message. This is called "message signing". These two concepts are combined while using the RSA algorithm to ensure that the message is coming from this sender and only the receiver can decrypt it. [11]. The main scheme of the RSA algorithm is based on how challenging is to prime factor huge numbers, even with the existing computational power and algorithms, it would take for an average workstation computer with two running CPUs roughly 5000 hours (208 days) to prime factor a 156-digit number [2].

ii. Mechanism

Suppose Bob and Alice want to communicate, but they have never met before and neither of them wants to spend the time to send a courier with a key. As a result, Eve might possibly learn everything Alice communicates to Bob. However, a message could still be transmitted so that Bob can read it, but Eve cannot; This would be impossible using any of the basic symmetric-key techniques. Diffie and Heilman's seminal study first made the feasibility of the current system known as a public key cryptosystem; However, they lacked a working implementation. at the time Several strategies were put forth during the following few years. The most effective one was introduced by Rivest, Shamir, and Adleman in 1977 and is known as the RSA algorithm. It is based on the point that the factorization of numbers into their prime components is difficult.

The RSA algorithm operates as follows, in order to create Bob chooses two unique huge primes p and q then multiplies them $n = pq$. He also chooses an encryption exponent e such that $\gcd\{e, p - 1 \times q - 1\} = 1$. While keeping the values of p and q secret, Bob sends Alice the pair n and e. Now Alice can communicate with Bob safely without ever needing to know p and q. Alice uses the letter m to convey her message. If m is greater than n, she divides the message into blocks that are each less than n. Alice then computes $C = m^e$ (mod n) and sends C to Bob. Since Bob knows p and q, he can compute $p - 1$ and $q - 1$, therefore he can find the decryption exponent d with $de = 1$ (mod $p - 1 \times g - 1$). As $m = C^d$ (mod n), Bob can read the message.

The algorithm can be summarized as follows:

1- Bob calculates $n = pq$ using two secret prime numbers p and q.

2- Bob chooses e with $\gcd\{e, p - 1 \times q - 1\} = 1$.

3- Bob computes d with $de = 1$ (mod $p - 1 \times g - 1$)

4- Bob makes n and e public and keeps p, q and d secret.

5- Alice encrypts m as $C = m$ (mod n) and sends C to Bob.

6- Bob decrypts m by computing $m = C^d$ (mod n).

III. Timing Attacks

Suppose that you could interfere in the middle of a system (a smartcard for example) that uses the RSA algorithm. Due to the design of this algorithm, you would be able to obtain only C and the public keys p and q, where, as explained in Section II, you can calculate the message M using $M \equiv C^d$(mod N). Yet, the only way to get the private key d is through brute-forcing or prime factoring n, which would take 300 trillion years for a 2048-bits RSA algorithm [7].

i. Mechanism

Paul Kocher proposed a type of attack that could break this encryption in [8]. By measuring the time needed to perform decryption and then perform series of calculation to get the private key d. This type of attack is now known as Timing attack.

First of all, to explain how the attack works, we will use the earliest form of the attack which is found in [8] and [5].

Let d in its binary form

$$d = \sum_{i=0}^{n} 2^i d_i \text{ where } d_i \in \{0,1\}$$

applying it in $M = C^d$ (mod N),

$$C = \prod_{i=0}^{n} M^{2^i d_i}$$

Now using the repeated squaring algorithms:

Set z equal to C and M equal to 1. Then for $i = 0, \dots, n$ do the following:

1- If $d_i = 1$, Set M to Mz (mod N)

2- Set z to z^2 (mod N)

Now, M has the value of C^d (mod N)

This will lead the equation to compute in less than or equal to $2n$ times, instead of d times [5].

Now to apply the attack, let the smartcard generate a number of messages k (i.e., $C_1, \dots, C_k$) and measure the time T_i needed to decrypt each message. If d is odd, then $d_0 = 1$, $M = C$, and $z = C^2$ (mod N). Now if $d_1 = 1$, the smartcard computes $Mz = C \times C^2$ (mod N). Else if $d_1 = 0$, it does not. Let t_i be the time needed to compute $C_i \times C_i^2$ (mod N). Since the time to compute $C_i \times C_i^2$ (mod N) depends on the value of C_i (calculating modulus takes different amount of time depending on the value of the number, hence each t_i takes different amount of time from each other.

As Kocher perceived, in [8], depicted in **Figure 2** if $d_1 = 0$, then t_i and T_i are independent from each other. While if $d_1 = 1$, t_i and T_i are correlated to each other. Hence, Kocher could determine if the value of d_1 is 0 or 1 depending on the

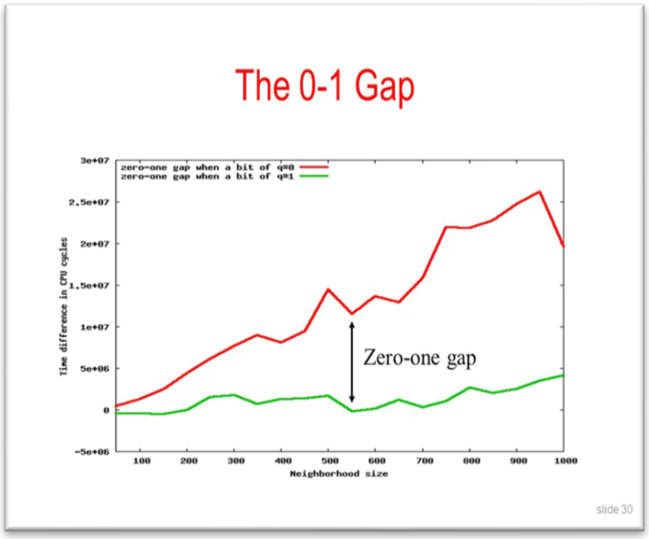

Figure 2: Graph showing computational power when $d_i = 0$ and $d_i = 1$

correlation of t_i and T_i. By this way, he can continue determining d_2, d_3, ..., d_n. Then by reverting the binary representation, Kocher could finally get the decryption key d.

ii. Countermeasures

A simple way to counter timing attacks is by giving all the numbers the same amount of time to do modular exponentiation (for example by delaying all the smaller numbers to have the same duration as the largest one), yet this would decrease the performance time of the algorithm. Another method is by using Blinding. Before calculating $M = C^d$ (mod N), choose random number r and s such that $s = r^e$ (mod n); compute $X = Cs$ (mod N) and $Y = X^d$ (mod N), hence $M = Y/r$ (mod n). Here the attacker can not use a timing attack, because the exponentiation process is done on an unknown random value X, instead of the known value C; However, Werner Schindler shows in [9] that an improved variation of timing attacks can still break a blinding RSA algorithm.

[10] introduces a better variation for RSA, which resists most of the attacks (including timing attacks); this algorithm works as follows:

Suppose you have A wants to send a message to B.

First: A will prepare public keys the same way as the standard RSA algorithm.

Second: B will do as following to encrypt the message:

1- Obtain A's public keys (N, e)

2- Represent the message in M such that $M < N$

3- Select a random integer k such that $\gcd\{k, n\} = 1$

4- Compute $c_1 = k^e$ (mod N)

5- Compute $c_2 = m^{\{e\}}k$ (mod N)

6- Send (c_1, c_2) to A

Finally, A should do the following to decrypt the message:

1- Use private key d and compute $c_1 = k$ (mod N)

2- Use the Euclidean algorithm and calculate the integer s such that $sk \equiv 1$ (mod N)

3- Compute $c_2 s = (m^e k)s = m^e (ks) = m^e$ (mod N)

4- Use the private key d to compute $(m^e)^d = m$ (mod N)

Using this algorithm, instead of the standard RSA algorithm, will provide more secure communication, as this improved variation counters timing attacks, since using k in encryption and decryption makes it challenging to distinguish between the time is taken for k and the time for the public key e or the private key d.

IV. Fault Attacks

From electrical devices, fault attacks retrieve secrets by taking advantage of hardware flaws. Boneh, DeMillo, and Lipton introduced fault-based attacks against CRT-RSA in the late 1990s. In situations when the message padding function is deterministic, these techniques factor the signer's modulus. The attack does not apply when the message is only partially known, such as when messages contain some randomness that can only be recovered when a valid signature is verified. A fault attack is an assault on a physical, electronic device in order to produce errors that cause the system to lose security (such as key recovery, an increase in an electronic purse balance, the acceptance of a false signature, or PIN code recovery) [12].

A fault attack is a live attack that compromises cryptographic hardware and allows secret information to be extracted. Attackers actively participate in fault assaults by providing other inputs in addition to the main input, such as fuzzing, radiation, heat, and vibration. By doing these, additional (typically incorrect) outputs are found that can provide further details about the algorithm and/or the secret. This procedure is explained in **Figure 3**

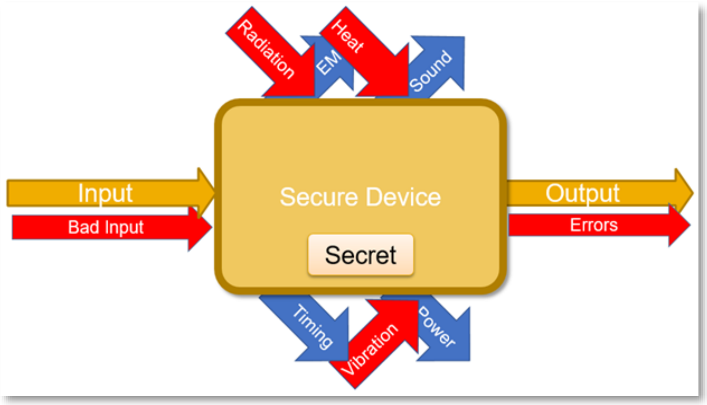

Figure 3: A schematic diagram of fault attacks and leakage types

i. Fault Methods

Power supply attacks: In case if the gadget isn't powerful enough, CMOS transistors are driven by power in electronic devices. If the device is marginally underpowered, some of the transistors may not be switched, leading to inaccurate computations. If the device is even slightly underpowered, it may have trouble entering an operational state (a boot loop), or it may even become completely broken. Injecting power spikes into the power supply is another attack strategy (to a similar effect).[1]

A device will become unstable and introduce problems if its power supply is underpowered or overpowered since some sections of a device are often more sensitive to it than others.

The most obvious situation for such an attack is when the device is owned by or under the attacker's control, as may be the case if they are looking at their own set-top box, etc. In such instances, the attacker is free to provide the gadget with any amount of power they desire.

Clock/Timing attacks: The clock, which is typically a bus shared by many of the system's components, synchronizes the propagation of calculations through the system; in other words, all inputs are prepared at the beginning, and when there is a rising edge on the clock bus, they begin propagating throughout the various computational components. When all calculations are complete, they all wait for the following rising edge on the clock bus to go on to the following phase. A rising edge would be injected on the clock bus at a random time during a clock glitching assault. In this manner, the device will be defective (unstable) since only part of the computations will have finished by that time while others are still being processed.

Temperature attacks: This attack strategy makes use of an electron's physical characteristic (current). Electrons "jump," and the hotter the environment, the more frequently and further they do so. Enough electrons can "jump" over the insulating layer in a transistor, for example, to switch it from logical 1 to 0, if a device becomes too hot. This leads to a flaw.

Due to the frequency of temperature-related device failures, temperature sensors are now incorporated into most modern electronics, causing them to shut down when they become too hot. By disconnecting the temperature sensor, a perpetrator can avoid detection. Another approach would be to fast change the device's temperature from extremely high to extremely low, resulting in an average temperature that is reasonable but faults during the extremes of the cycles.

A type-confusion attack is on the Java virtual machine (shown in **Figure 4**}. At initially, the memory was filled with little arrays (say of size one). It is typically impossible to access one of the memory regions using a pointer to another region because the Java Virtual Machine is type safe. The researchers heated the device's memory chip with a 50W light bulb in order to flip some of the bits and introduce a type-confusion problem. Because of this, a tiny number of the data structures that described the arrays in memory suddenly had incorrect values (for example, changed from size* = 1 to *size = 20). Currently, the attackers have read and write access to some impacted data structures since they contain a header from a separate data structure. The attackers gained access to the whole memory of the system by

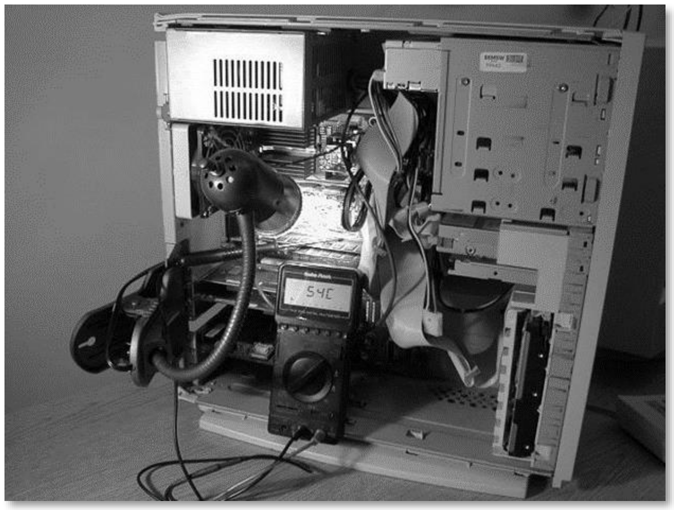

Figure 4: A light bulb flipping memory bits filled with safe Java structures

altering the second data structure's header to any value.

ii. Chinese Remainder Theorem

Before going through the mechanism of fault attacks, an important number theory theorem, Chinese Remainder Theorem, is going to be explained, as it is much involved in the mechanism of the attack.

The idea is that if we know both x (mod p) and x (mod q) then we can easily calculate x (mod n). So, given a message M, calculate M_p and M_q: $M_p = C^d$ (mod n) $= C^d$ (mod p), $M_q = C^d$ (mod n) $= C^d$ (mod q). To combine the values, we do: $M =$ CRT(M_p, M_q)= $M_p \times q \times (q^{-1}$ (mod p)) + $M_q \times p \times (p^{-1}$ (mod q)). It is easily provable that M (mod p) = M_q and M (mod q) = M_p, so by the Chinese Remainder Theorem, this value must be equal to M [13].

iii. Mechanism

The attacker has a decryption box (known plaintext scenario) with public key n and would like to recover d (the private key). The attacker is also aware that CRT is being used for decryption by the decryption box. Let's finally assume that the attacker can introduce any kind of flaw into the decryption procedure.

The attacker first gets $M = M_p \times q$ (q^{-1} (mod p)) + $M_q \times p$ (p^{-1} (mod q)) through the regular decryption process. Then, the attacker primes the device to re-calculate the message from the same cipher, this time injecting a transient fault during the calculation of Mp, resulting in the device erroneously producing M'_p instead of $M'_p \neq C^d$ (mod p). The device will then proceed to combine M'_p with the correct result of M_q, resulting in: $M' = M'_p \times q$ (q^{-1} (mod p)) + $M_q \times p$ (p^{-1} (mod q)). Now the attacker can calculate the value of $M - M'$: [$M = M_p \times q$ (q^{-1} (mod p)) + $M_q \times p$(p^{-1} (mod q))] - [$M'_p \times q$ (q^{-1} (mod p)) + $M_q \times p \times$ (p^{-1} (mod q))]. Finally, calculating the gcd of n and $M - M'$ yields: $gcd\ (n, M - M') = gcd\ (pq, (M\_p - M'_p) \times q \times (q^{-1}$ (mod p))) = q

The greatest common divisor of n and anything can be only p, q, n or 1. On the other hand, M_p and M'_p can never be multiples of p, otherwise, both would equal 0. So, by that reasoning, $gcd\ (pq, (M\_p - M'_p) \times q \times (q^{-1}$(mod p))) must equal q, and thus we have cracked the cipher using a single fault attack [12].

iv. Countermeasures

Usually, the fault attacks cannot be stopped at the cipher design level. The responsibility of ensuring that an adequate countermeasure is implemented falls on the circuit implementer/secure software developer rather than the cipher creator. Therefore, as we discuss here, countermeasures are created and examined independently of ciphers. We exclude the combined fault and side channel countermeasures for the sake of simplicity. To achieve protection against FA, all solutions that reduce fault effect make use of some redundancy.

In 2008, Vigilant put up an effective strategy for safeguarding modular exponentiation from fault attacks and applied this finding to the instance of CRT-RSA. Vigilant's countermeasure appears to be among the most cost-effective strategies considering all embedded device limits when compared to other methods. In fact, this solution does not require the public exponent, precomputation, additional arguments, or personalization that is incompatible with the JavaCard standard. Additionally, the countermeasure's performance and memory usage overhead is tolerable [16].

Vigilant's generic secure exponentiation: The principle of Vigilant's secure exponentiation method consists in computing m^d mod N in $\mathbb{Z}_{Nr^2}$ where r is a small random integer coprime with N. Then the base m is transformed into m' such that:

$$m' \equiv \begin{cases} m & \bmod N \\ 1 + r & \bmod r^2 \end{cases}$$

This implies that:

$$S' = m'^d \ mod\ \ Nr^2 \equiv \begin{cases} m^d & \bmod N \\ 1 + dr & \bmod r^2 \end{cases}$$

Therefore, a consistency check of the result S' can be performed modulo r^2 from d and r. If the verification $S' = 1 + dr\ \ mod\ r^2$ is successful, then the result $S = S'\ \ mod\ N$ is returned.

Vigilant's application to RSA with CRT: The application to RSA with CRT perform both half-exponentiations modulo pr^2 and qr^2, which is the basic idea to ensure that there were no errors during the computation of S_p or S_q during the recombination. Therefore, it is possible to carry out the last consistency check after recombination. A more detailed explanation of the procedures is provided in **Algorithm 1**, which is an exact replica of Vigilant's scheme [17].

Algorithm 1: Vigilant’s CRT-RSA scheme

```
Inputs: The message to sign $m$, the private key $(p, q, d_p, d_q, i_q)$, a 32-bit random integer $r$, four 64-bit random integers $R_1$, $R_2$, $R_3$, and $R_4$
Outputs: $S = m^d \pmod N$
Steps:
1. $p' = p r^2$, $m_p = m \pmod{p'}$
2. $i_{pr} = p^{-1} \ (mod\ r^2)$, $\beta_p = p i_{pr}$ and $\alpha_p = 1 - \beta_p \ (mod\ p')$
3. $\widehat{m_p} = \alpha_p m_p + \beta_p (1 + r)\ (mod\ p')$
4. if $(\widehat{m_p} \neq m\ (mod\ p))$ then
5.     return error
6. $d'_p = d_p + R_1(p - 1)$
7. $S_{pr} = \widehat{m_p^{d'_p}} \pmod{p'}$
8. if $(d'_p \neq d_p\ (mod\ p - 1)$ then
9.     return error
10. if $(\beta_p S_{pr} \neq \beta_p (1 + d'_p r)\ (mod\ p'))$ then
11.     return error
12. $S'_p = S_{pr} - \beta_p (1 + d'_p r - R_3)$
13. $q' = q r^2, m_q = m\ (mod\ q')$
14. $i_{qr} = q^{-1}\ (mod\ r^2)$, $\beta_q = q i_{qr}$ and $\alpha_q = 1 - \beta_q\ (mod\ q')$
15. $\widehat{m_q} = \alpha_q m_q + \beta_q (1 + r)\ (mod\ q')$
16. if $(\widehat{m_q} \neq m\ (mod\ q))$ then
17.     return error
18. if $(m_p\ (mod\ r^2) \neq m_q\ (mod\ r^2))$ then
19.     return error
20. $d_q^y = d_q + R_2(q - 1)$
21. $S_{qr} = \widehat{m_q^{d'_q}}\ (mod\ q')$
22. if $(d_q^v \neq d_q\ (mod\ q - 1))$ then
23.     return error
24. if $(\beta_q S_{qr} \neq \beta_q (1 + d'_q r)\ (mod\ q'))$ then
25.     return error
26. $S'_q = S_{qr} - \beta_q (1 + d'_q r - R_4)$
27. $S = S'_q + q(i_q(S'_p - S'_q)\ (mod\ p'))$
28. $N = pq$
29. if $(N[S - R_4 - q i_q (R_3 - R_4)] \neq 0\ (mod\ N r^2))$ then
30.     return error
31. if $(q \dot{x}_q \neq 1\ (mod\ p))$ then
32.     return error
33.     return $S\ (mod\ N)$
```

This method has the following advantages:

- The only requirements for the random integer r are that it be odd and have sufficient entropy.
- Precomputation is not required.
- Only p, q, d_p , d_q, i_q and the input message m are required for the calculation
- The countermeasure's implied overhead for performance and memory use is reasonable.

V. Conclusion

Since its invention, the RSA algorithm has become the most implemented communication and digital signature algorithm, due to being the first asymmetric-key algorithm. While this prominence of the RSA algorithm resulted in it being the center of attacks, one type of these attacks, implementation attacks, also affects algorithms related to the RSA algorithm. However, these multitudinous attacks have induced the generation of better variations of the RSA algorithm. This review paper aimed to scrutinize the potentiality of using the algorithm presently without facing any lack of security or performance. The paper has exhaustively investigated the aspects of two of the most popular implementation attacks: timing attacks and fault attacks, successfully observing improved RSA counterparts that are secured against these attacks. However, the critical question of using the RSA in the current time remains. As the paper addressed only the counterpart of the RSA against each of those two attacks, no variation of RSA that can simultaneously resist those two types of implementation attacks was clarified. Hence, more research is required.

VI. References

[1] W. Trappe and L. C. Washington, Introduction to cryptography: with coding theory, 2nd ed. Upper Saddle River, N.J: Pearson Prentice Hall, 2005.

[2] J. Katz and Y. Lindell, Introduction to modern cryptography. Boca Raton: Chapman \& Hall/CRC, 2008.

[3] J. F. Dooley, History of cryptography and cryptanalysis: codes, ciphers, and their algorithms. New York, NY: Springer Berlin Heidelberg, 2018.

[4] W. Diffie and M. Hellman, "New directions in cryptography," IEEE Trans. Inform. Theory, vol. 22, no. 6, pp. 644–654, Nov. 1976, doi: 10.1109/TIT.1976.1055638.

[5] D. Boneh, "Twenty Years of Attacks on the RSA Cryptosystem," Notices of the American Mathematical Society, vol. 46, issue 2, Feb 1999.

[6] T. Popp, "An introduction to implementation attacks and countermeasures," in 2009 7th IEEE/ACM International Conference on Formal Methods and Models for Co-Design, Cambridge, MA, USA, Jul. 2009, pp. 108–115. doi: 10.1109/MEMCOD.2009.5185386.

[7] C. Gidney and M. Ekerå, "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits," Quantum, vol. 5, p. 433, Apr. 2021, doi: 10.22331/q-2021-04-15-433.

[8] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in Advances in Cryptology — CRYPTO '96, vol. 1109, N. Koblitz, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 104–113. doi: 10.1007/3-540-68697-5_9.

[9] W. Schindler, "Exclusive Exponent Blinding May Not Suffice to Prevent Timing Attacks on RSA," in Cryptographic Hardware and Embedded Systems -- CHES 2015, vol. 9293, T. Güneysu and H. Handschuh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 229–247. doi: 10.1007/978-3-662-48324-4_12.

[10] S. Pradhan, "A New Design to Improve the Security Aspects of RSA Cryptosystem," International Journal of Computer Science and Business Informatics, vol. 3, issue 1, Jul 2013

[11] M. Mumtaz and L. Ping, "Forty years of attacks on the RSA cryptosystem: A brief survey," Journal of Discrete Mathematical Sciences and Cryptography, vol. 22, no. 1, pp. 9–29, 2019.

[12] Sung-Ming Yen, Seungjoo Kim, Seongan Lim, and Sang-Jae Moon, "RSA speedup with Chinese remainder theorem immune against hardware fault cryptanalysis," IEEE Transactions on Computers, vol. 52, no. 4, pp. 461–472, 2003.

[13] Y. Zheng and T. Matsumoto, "Breaking Real-World Implementations of Cryptosystems by Manipulating their Random Number Generation", Pre-proc. 1997 Symp. Cryptography and Information Security, 29 Jan.-1 Feb. 1997.

[14] D.P. Maher, "Fault Induction Attacks Tamper Resistance and Hostile Reverse Engineering in Perspective", Financial Cryptography, pp. 109-121, 1997.

[15] E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems", Advances in Cryptology CRYPTO '97, pp. 513-525, 1997.

[16] J.-S. Coron, C. Gira, N. Morin, G. Piret, and D. Vigilant, “Fault attacks and countermeasures on vigilant's RSA-CRT algorithm,” 2010 Workshop on Fault Diagnosis and Tolerance in Cryptography, 2010.

[17]] D. Vigilant. RSA with CRT: A New Cost-Effective Solution to Thwart Fault Attacks. In CHES 2008, vol. 5154 of LNCS, pp. 130–145. Springer, 2008.