Modified SHA-3 Side-Channel using Yarrow-Based Random Delay on Secure Communication

Ronald Christian C. Fausto III [1], Marion Cedric Emanuel V. Tapang [1], Dan Michael Cortez [2]

[1] Student, College of Engineering and Technology, Pamantasan ng Lungsod ng Maynila, Philippines.

[2] Professor, College of Engineering and Technology, Pamantasan ng Lungsod ng Maynila, Philippines.

Corresponding Author: rccfausto18@gmail.com

Abstract: - Secure Hash Algorithm-3 or SHA3 is a hashing algorithm that, when not appropriately implemented, can exhibit vulnerabilities to side-channel attacks. One type of this attack is a timing analysis attack. It utilizes timing information gained to obtain the key required for the algorithm. In this study, the researchers proposed a Random Delay implementation with a CSPRNG, specifically Yarrow CSPRNG, as its number generator, thus creating high-quality randomness that is highly secured from side-channel attacks, unique timings, and exhibits less predictability. Statistical tests such as the Avalanche Test evaluated the algorithm's performance. The test yielded 51.18 percent, and the performance analysis regarding CPU Load and Memory Usage is not far from a standard implementation of SHA3 and a standard implementation of SHA3 with Random Delay. The results proved that the proposed performance is cryptographically secure and efficient, as all tests yielded favorable results.

Key Words: —*SHA3, Side-Channel, Yarrow, Pseudorandom Number Generator, Random Delay, Performance, Cryptography.*

I. INTRODUCTION

Secure communication has become a vital aspect of modern communication due to the general use of the web and an increasing need for data privacy and security. The previous encryption algorithms have been exploited for various attacks, thus searching for new, more secure methods of protecting sensitive information [1][2]. One such technique involves using cryptographic hash functions, mainly used on various security-related applications, including authentication and digital signatures [3].

Manuscript revised June 15, 2023; accepted June 16, 2023. Date of publication June 18, 2023.

This paper available online at www.ijprse.com

ISSN (Online): 2582-7898; SJIF: 5.59

Secure-Hash Algorithm 3 (Keccak) was the declared winner of the NIST Competition in 2012 [4]. Moreover, it is proven to be secure and to have solved the major issues of its predecessors. Despite the impressive enhancements over its predecessors, studies have been conducted on the Algorithm, and when implemented incorrectly, it could be prone to side-channel attacks. One of these side-channel attacks is the Timing Analysis Attack.

Timing Analysis Attack uses the timing information gained on the side channel to obtain sensitive information such as the key. One countermeasure against these Timing Analysis Attacks is the insertion of a Random Noise or Random Delay in the side channel. Random Delay insertion disrupts the information gained in the side channel by inserting randomized noises [5]. The quality of these Random Delays depends on the number generator used, as the Random Number Generator determines the randomness, predictability, and security of these random delays. Using a Pseudorandom Number Generator (PRNG) to generate the unexpected delay can be a viable solution. Still, a PRNG is known to exhibit a randomness pattern, thus making it predictable and not made with security in mind. The previous concerns can be addressed using a Cryptographically Secure

Pseudorandom Number Generator (CSPRNG). CSPRNG exhibits highly reliable randomness and security and is less predictable than PRNG [6]. Aside from this, inserting an additional operation in a cryptographic operation such as SHA3 can affect the algorithm's performance and cryptographic strength [7][8].

II. Review of Literature and Studies

In the field of cryptography, one should prioritize the protection of user confidentiality, data privacy, and security. It must not disclose unintended information other than those mainly involved in the conversation [9]. This information can be revealed using various kinds of attacks. One of those attacks is called a side-channel attack. The side-channel attack exploits the physical characteristics of cryptographic systems to extract secret information [10]. The SHA3 Algorithm is built to be resistant to common cryptographic attacks. It is also designed to address the previous problems of its predecessors. It demonstrates high unpredictability and is resistant to collision attacks. While the algorithm is cryptographically secure, incorrect implementation of the problem could result in information leaks through timing analysis, power consumption, and other cyberattacks [11][12].

The insertion of Random Delay can bring an attacker an inconsistent timing analysis result. Random Delay obscures the timing information gained, thus making the information gained by the attacker somehow irrelevant [13]. These Random Delays must be significant enough to obscure these timing patterns. The source of randomness used to generate these delays should be robust and unpredictable to prevent an attacker from predicting the delay [14]. Depending on the implementation of these random delays, this can cause transmission errors that can harm the performance and introduce vulnerabilities. This can be inhibited and manipulated with a good source of delay as its deterministic, systematic, and data-dependent [15].

Using a Pseudorandom Number Generator (PRNG) is pivotal in numerous cryptographic procedures as it introduces high randomness and unpredictability, thus ensuring high security [16]. Random Delays are created using generated numbers to determine the delay intervals. However, PRNGs are deterministic and can be reproduced if the seed state is known. A Cryptographically Secure Pseudorandom Number Generator (CSPRNG) is recommended as the one requirement for a reliable random delay are the unpredictable random numbers [17][18]. CSPRNGs exhibit high randomness and unpredictability compared to PRNGs. Also, CSPRNGs are created to resist various cryptographic attacks while generating numbers close to true randomness and being least predictable. One example of these CSPRNGs is the Yarrow Algorithm. The Yarrow Algorithm uses two entropy pools: Fast and Slow Entropy Pools, mixed for its reseeding function, making it a highly secured random number generator [19].

Table.1. Comparison of PRNG and CSPRNG

Features/ Properties	**PRNG (Pseudorandom Number Generator) (As per Von Neumann, 1951)**	**CSPRNG (Cryptographically et al. Number Generator) (As per Blum, Blum & Shub, 1986)**
Predictability	After some time, the sequence starts repeating itself, making it somewhat predictable.	Infeasible to predict the following output given all the previous ones.
Seeding	Requires an initial value or seed to start the generation process.	Similar to PRNG, it requires an initial seed. However, CSPRNG might be frequently reseeded to increase security (Yarrow).
Quality of Randomness	It might not have a high level of randomness.	Ensures a higher level of randomness than PRNGs, as per the next-bit-unpredictability test and the state compromise extensions.
Backtracking Resistance	No built-in mechanism for backtracking resistance.	It has built-in backtracking resistance, i.e., compromise of current output does not compromise past outputs.
Period Length	Period Length Often has a fixed period length after which the sequence repeats.	It has a substantial, if not infinite, period before any repetition occurs.
Mitigation against Side-channel Attacks	It does not offer any specific feature to mitigate side-channel attacks.	Some CSPRNGs provide mechanisms for mitigating side-channel attacks.

A requirement to modify these cryptographic mechanisms should be a complete understanding of the cryptographic algorithm, as slight changes can weaken a cryptographic algorithm. Any negligence when implementing these changes

can introduce a vulnerability that an attacker can exploit [20][21]. The implementation of Random Delay counter-timing attacks as it adds a computation process to the operation. This information must be significant enough to obscure the timing patterns without excessively impacting the algorithm's performance [22][23]. Using a well-designed CSPRNG as a component for Number generation can be an effective defense against side-channel attacks. Its emphasis on the quality of pseudorandom numbers reminds us of the importance of robust randomness in securing cryptographic systems against various forms of attack.

Table.2. Comparison of Random Delay, PRNG Random Delay, and CSPRNG Random Delay

Features/ Properties	**Standard Random Delay**	**PRNG Random Delay**	**CSPRNG Random Delay**
Predictability	Highly predictable as the delay might be somewhat random.	Less predictable due to using a PRNG, but it can still show a repeating pattern after some time.	Least predictable due to the use of a CSPRNG, making it highly secure.
Uniqueness of Delay	Delay times might only be unique if the randomness source is genuinely random.	More unique delay times due to the PRNG, but the sequence may still repeat after some time.	Most unique delay times are due to the CSPRNG, with a substantial, if not infinite, period before any repetition occurs.
Resilience to Timing	Attacks Vulnerable to timing attacks due to its predictability	More resistant to timing attacks than a simple random delay, but still vulnerable due to repeating patterns.	Best resistance to timing attacks due to its unpredictability and backtracking resistance.
Complexity	Simplest to implement as it does not require a sophisticated random number generator.	More complex to implement due to the need for a PRNG.	Most complex to implement due to the need for a CSPRNG.
Use Case	Suitable for non-critical applications where security is not a high priority.	More suited to applications requiring a higher level of security than simple random delay offers, but not as high as cryptography applications.	They are primarily used in high-security applications where the risk of timing attacks is a concern.

III. Research Methodology

3.1 SHA3 Operations

The SHA3 is known to be secured against most attacks as its one of the criteria of NIST for choosing the winning algorithm. However, a poorly implemented SHA3 can introduce potential attack vulnerabilities, e.g., Fault Analysis and Timing Analysis. To evaluate the side channel from Timing Attacks, a timing analysis attack must be introduced to the side channel. A timing analysis attack is a side-channel attack that aims to collect information about the time it takes for a system to process functions and inputs. This timing information, when measured, can reveal sensitive information about the data being processed. The information gained through this attack is often used to unravel the secret keys. Timing Analysis Attacks get the timing information produced during the application's runtime where SHA3 is implemented. The values produced during that time will be visualized using a statistical tool to determine if implementing Random Delay to the SHA3 will improve its side channel.

The SHA3 operates through the use of a function called the *sponge* function. This function consists of three primary operations: Initialization, Absorption, and Squeezing, wherein the Random Delay will be inserted.

Initialization

In SHA-3, the algorithm's state is represented as a three-dimensional array of bits. This state is often visualized as a 5x5 matrix of 64-bit "lanes" for 1600 bits (this is the case in SHA-3, other Keccak implementations may use different sizes).

Given a message M and a rate parameter r, split M into blocks of size r bits, denoted as $M[1]$, $M[2]$,..., $M[n]$, where n is the number of blocks. Then, for each block, I from 1 to n, update the state S as:

$$S = S \oplus pad\ (M[i])$$

Equation 1: SHA-3 Initialization Phase

Where $\oplus$ represents the bitwise XOR operation, and *pad* is a padding function that expands $M[i]$ to the size of the state S.

3.2 Absorption

In the absorption phase, the input message is divided into blocks, each of the same size as the state (1600 bits for SHA-3). These blocks are XORed into the state, one block at a time. After each block is XORed, a permutation function, denoted as f, is applied to the state. The state S is transformed through the Keccak-*f* permutation function, wherein it can be denoted as:

$$S = Keccak - f(S)$$

Equation 2: SHA-3 Initialization Absorption Phase

3.3 Squeezing

During the squeezing phase, output blocks are read from the state. The first output block is the first 'd' bits of the state (where 'd' is the desired length of the output). If more output bits are needed, the permutation function 'f' is applied to the state again, and the next 'd' bits are read. This continues until enough output bits have been generated.

Given an output length d, extract and output z blocks of r bits from the state S, where z is the smallest integer such that $r \cdot z \geq d$.

$$Z[i] = S[i] \text{ for } i \text{ in } [1,z]$$

Equation 3: SHA-3 Initialization Squeezing Phase

Each time a block is extracted, the state S is updated via the Keccak-*f* permutation:

$$S = Keccak - f(S)$$

The final output is concatenating all $Z[i]$ blocks, truncated to d bits.

While more output bits are needed:

- Zi = first 'd' bits of state (Zi is the i-th output block)

state = f(state)

3.4 Yarrow Algorithm Operations

Yarrow is a Cryptographically Secure Pseudorandom Number Generator known for its high-quality randomness, ease of implementation, and other excellent features. The Yarrow algorithm generates these cryptographically secure pseudorandom numbers for the improved quality of these random delays. Yarrow has a four-part process that will be used in this study.

Initialization

$$P_0 = getrandbits(256)$$

Equation 4: Yarrow Initialization

Entropy Gathering

$$P = P \oplus CryptGenRandom(32)$$

$$P = P \oplus \text{i=1n} TimingKeyboard_i$$

$$P = P \oplus \text{i=1n} MovementMouse_i$$

Equation 5: Yarrow Entropy Source Gathering

Where:

- P represents the current state of the pool
- $\oplus$ represents the bitwise XOR operation.
- *CryptGenRandom(32)* represents 32 bytes of random data generated by the CryptGenRandom function
- *TimingKeyboard*, represents the time between the i-th and (i-1)-th keypress.
- *MovementMouse*, represents the i-th recorded mouse movement.

Reseeding:

$$P = P \oplus os.randoms(32)$$

Equation 6: Yarrow Reseeding

Random Number Generation:

$$Bytes_{required} = \text{length32}$$

$$RandomBytes = \oplus \text{i=1} Bytesrequired\ P.to\_bytes(32,'big')$$

$$P = P \oplus int(P.to\_bytes(32,'big'))$$

Equation 7: Yarrow Random Number Generation

Where:

- $Bytes_{required}$ represents the number of 32-byte blocks required to meet the requested length
- *RandomBytes* represents the generated random bytes
- $P.to\_bytes(32,'big')$ represents the current state of the Pool converted to bytes
- $int(P.to\_bytes(32,'big'))$ represents the current state of the Pool converted to an integer

Yarrow is a type of CSPRNG that relies heavily on its high and low entropy pool to generate random numbers. These entropy sources come from a variety of unpredictable and difficult-to-manipulate real-world events. Both entropy sources are mixed into one pool, and Yarrow utilizes this pool for its randomness. One highly reliable high entropy source is the CryptGenRandom Function, which exhibits high-quality randomness. It derives its entropy from various stochastic system sources, such as system performance counters, system state data, and cryptographic sources. The authors also suggested the usage of combined Keyboard Timings, the precise timing between keys you press on the keyboard, and Mouse Movement timings, the precise movements of your mouse, which can be used as a source of entropy for a reliable low entropy source pool. These entropy sources will then be mixed into a pool for the Yarrow Algorithm. These two could be a reliable low entropy source for the Yarrow. [24]

3.5 Random Delay

Random Delay is an implementation in which a random delay is set within a specific range to create confusion for attackers utilizing time information.

Formulaic Representation:

Here, D represents the delay time, RD represents the Generated Random Delay, and W represents Wait for the Delay duration:

$$D = RD + W.$$

Equation 8 Random Delay

Generated Random Delay

This is part of the computation wherein the random delay time will be generated within the predefined range. It can be represented as follows:

Let RD be the Random Delay function that takes min_delay and max_delay as inputs.

$$RD(min\_delay, max\_delay) = D$$

Equation 9: Random Delay Generation within the Pre-Defined Range

D is the delay time, defined as a randomly generated number between min_delay and max_delay.

Wait for the Delay

After the delay time was chosen from all the random values, the program will now wait for the chosen duration. It can be represented as follows:

$$Wait(D)$$

Equation 10: Wait for Delay for the Random Delay

The delay time *D* is determined randomly within the range of values given in the *min_delay* and *max_delay*. This randomness is what gives the random delay its security properties.

3.6 Testing and Evaluation

This implementation of Yarrow-Based Random Delay to the SHA3 will be analyzed for performance and security to ensure that the implementation can be deployed and implementation will not compromise existing and working frameworks. The performance analyzers that can help us analyze if this implementation can truly help are Cryptanalysis, memory usage, and CPU load. A memory usage test is one software test that aims to know the application's memory usage efficiency. This ensures that the application is not utilizing unnecessary resources which other applications could use. Memory usage testing may examine issues such as memory leaks or memory that is no longer utilized but was not released. It can also examine this excessive memory usage, which could impact performance and memory fragmentation. CPU load testing tests the system under the highest load it will ever need to handle. This involves running tests that require significant processing power and monitoring how well the system handles the workload. CPU load testing ensures the system can handle its maximum expected load without performance issues or failures. It is a form of stress testing that pushes the system to its limits to verify its robustness and reliability under extreme conditions. Avalanche Test tests the "avalanche effect" property. Avalanche Effect assumes that an excellent cryptographic function should drastically change its output, even at the slightest change. When compared, it should have a new output that cannot be easily correlated with the old output.

3.7 Yarrow-Based Random Delay

The usage of a simple random delay is highly predictable, has non-unique delay times, and is vulnerable to attacks due to these factors. These weaknesses are unsuitable for implementing sensitive applications such as communications applications. These kinds of implementations compromise confidentiality in case an attack happens. The studies mentioned above about introducing random delay using a PRNG answer all the concerns above; it is less predictable as it is pseudorandom, has more unique delays, and is resistant to attacks. The underlying problem of using a PRNG as its source of randomness is that a PRNG can repeat a given time. This is a significant problem that an attacker might utilize to scour data

in the side channel. However, this is solved through the use of CSPRNG. CSPRNG is a Cryptographically Secure PRNG, which is a PRNG that is built with security in mind. This has the properties of a PRNG, but it is more secure. It is the least predictable among them due to its generation of random numbers close to True Randomness; it generates unique delays and will take a very long time before a repetition happens. Since it is created with security in mind, it resists most known attacks, including side-channel attacks. Integrating the Yarrow CSPRNG as the source of randomness for the Random Delay will significantly increase the security and randomness of the Random Delay. The main objective of this implementation is to increase the randomness of the generated random delay, which will disrupt an attacker performing timing analysis attacks. As a result, it will exhibit a highly randomized noise during the execution of the SHA3.

It will start the Yarrow Entropy Gathering process as soon as the application starts. It will then start mixing the entropy gained during that process, and as soon as the SHA3 process starts, the Random Delay will call Yarrow to generate its Random Numbers for the Random Delay timings. Finally, it will start generating Random Delays during the SHA3 function, which will then disrupt the timings for the timing analysis attack. The whole Yarrow-Based Random Delay process can be seen in Figure 1 below.

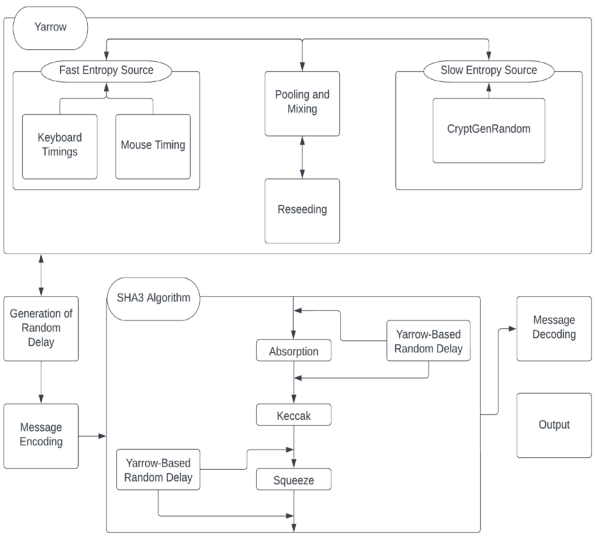

Fig.1. Proposed Yarrow-Based Random Delay on SHA3 Side-Channel

IV. Simulation and Results

4.1 Timing Analysis

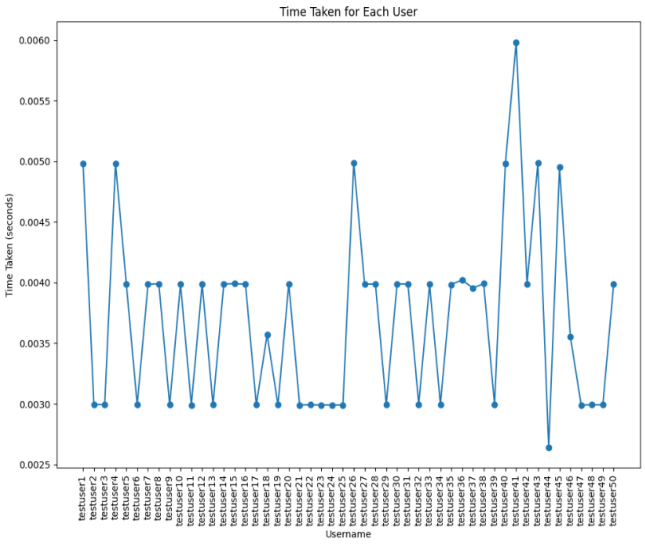

Fig.2. Simulation of Timing Analysis using the Yarrow-Based Random Delay on SHA3 Side-Channel on the communications application

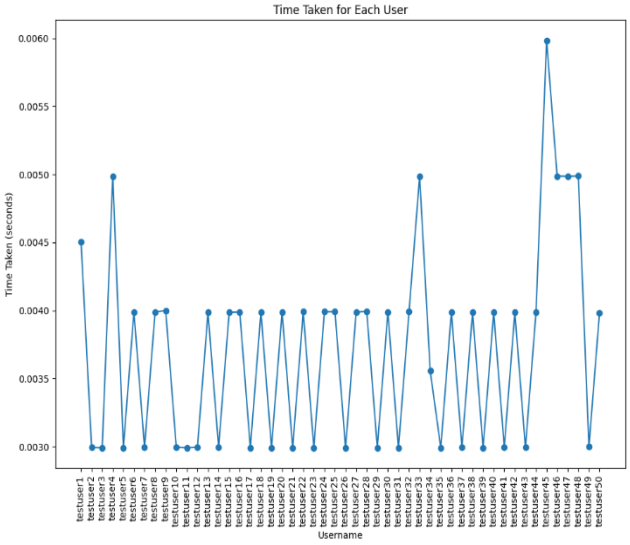

Fig.3. Simulation of Timing Analysis using a simple Random Delay on SHA3 Side-Channel on the communications application

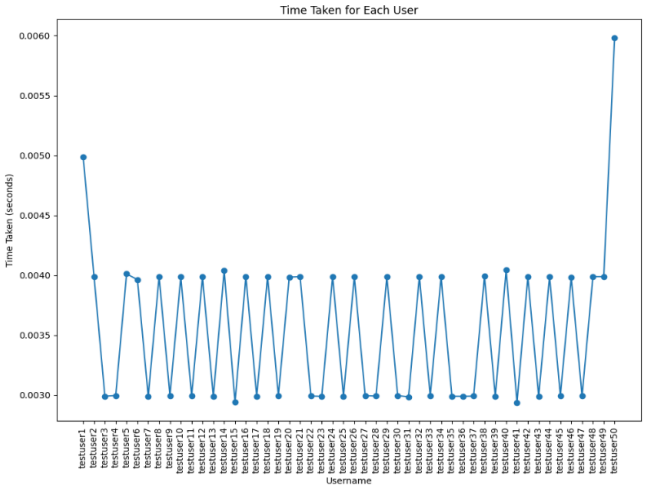

Fig.4. Simulation of Timing Analysis on SHA3 Side-Channel on the Communications application

Figure 4 shows that the SHA-3 Algorithm alone exhibits a pattern, which can be easily recognized as it repeats itself in a given period. In Figure 3, the SHA-3 Algorithm with a random delay integrated into its side channel alone shows a slight disruption from the pattern. These disruptions, however, are common, as their results are only occasional. In Figure 2, the SHA-3 Algorithm with the Yarrow-Based Random Delay integrated into its side channel exhibits frequent disruption from the pattern in the previous figures. The timing information gained can hardly be recognized in the Yarrow-Based Random Delay implementation on SHA-3 Side-Channel. The Yarrow-Based Random Delay can disrupt the timing information obtained through timing analysis attacks.

4.2 Performance Analysis

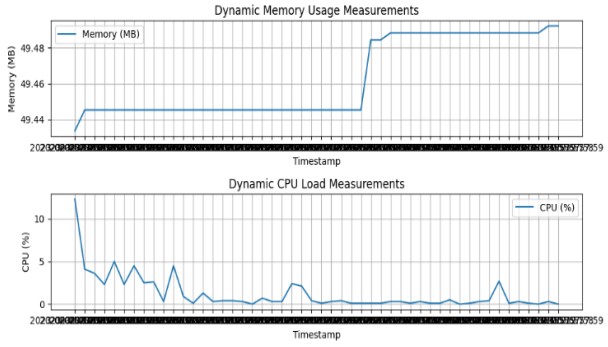

Fig.5. CPU Load and Performance Analysis of the Communications application with the Yarrow-Based Random Delay on SHA3 Side-Channel

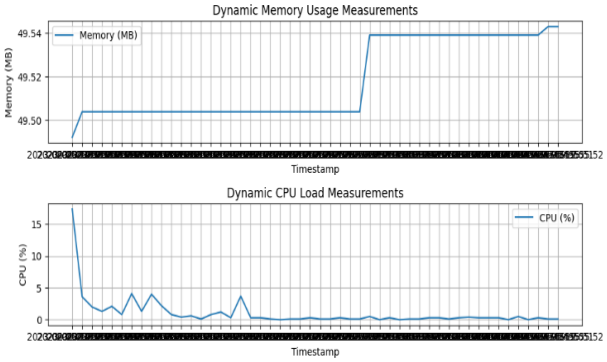

Fig.6. CPU Load and Performance Analysis of the Communications application with the Random Delay on SHA3 Side-Channel

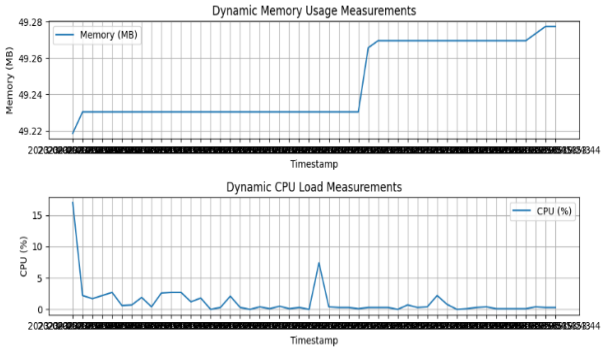

Fig.7. CPU Load and Performance Analysis of the Communications application on SHA3 Side-Channel

Figure 7 shows that the SHA-3 Algorithm's Performance in terms of memory usage and CPU load is not far from the Performances of Figures 6 and 5. A sharp difference in the spike levels in Figure 6 and Figure 5 can be observed in the Memory Usage compared to Figure 7. A behavior change can be observed in the CPU Load seen in Figure 7. It only spikes when the SHA3 Operation is called. In Figure 5 and Figure 6, the CPU load spikes early, even before the SHA3 Operation. The effect of the implementation in terms of CPU Load and Memory Usage is close to the standard implementation of SHA3 on secure communication.

4.3 Cryptanalysis

Cryptanalysis is needed to identify and analyze potential risks and trade-offs associated with introducing Yarrow-Based Random Delay in the SHA-3 algorithm, focusing on possible vulnerabilities to other side-channel attacks or unintended consequences. Performing a Cryptanalysis is necessary to test the cryptographic strength of the SHA3 algorithm with the

Yarrow-Based Random Delay. Although no change was introduced to the SHA3 operations regarding changed parameters or values, adding a Random Delay between these operations disrupts the regular operation by introducing this random timing information that could open vulnerabilities when incorrectly implemented.

Table.3. Avalanche Test Results

Test Scenario	SHA3 with Yarrow-Based Random Delay	Standard SHA3
Average Avalanche Effect	0.51	0.53

The required result for the Avalanche Effect to be considered successful is 50% above. As shown in the test performed, the SHA3 with Yarrow-Based Random Delay has an avalanche score of 0.51 or 51% compared to the 0.52 or 52%, thus showing that while it improves the security of the SHA3 Side-Channel, it still retains its cryptographic strength.

V. Conclusion and Recommendation

5.1 Conclusion

The SHA3 Algorithm, when not implemented correctly, can exhibit patterns that can be utilized to gain sensitive information about the used cryptographic algorithm. Implementing Random Delay to the SHA3 Side-Channel adds a layer of security but is inconsistent. It can be easily interpreted as a distraction as the simple Random Delay still exhibits the pattern created by the SHA3 Algorithm. The enhancement performed in this study significantly increased the quality of the randomness, and it cannot be easily discerned as the Random Delay generated thru the Yarrow-Based Random Delay is a high-quality randomness. The Memory Usage and the CPU Load of the implemented enhancement are significantly close to the performance of the application that uses the SHA3 alone without any Random Delay inserted in its operations, and it also passed cryptanalysis. The Yarrow-Based Random Delay is a highly secured implementation that can provide countermeasures against a timing analysis attack on a secure communication that uses the SHA3 algorithm as its Cryptographic Algorithm.

5.2 Recommendation

The researchers recommend using constant time implementation, masking techniques, and other implementations alongside the random delay further to increase the security of the SHA3 side channel. The researchers also recommend performing other attacks against the SHA3 side channel to test if the implementation is suitable for other side-channel attacks.

Acknowledgments:

The researchers would like to acknowledge and warmly thank our advisor and professors in PLM who made this study possible. The researchers would also like to express our deepest gratitude to our family for their continuous support. We dedicate this study to the Almighty God for giving us the strength to overcome our difficulties.

References

[1]. Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). Handbook of Applied Cryptography.
[2]. Rebeiro, C., Mukhopadhyay, D., & Bhattacharya, S. (2011). Timing Channel Attacks: A Survey on Cache Based Side Channel Attacks (Cryptology et al. 2011/578).
[3]. Rivest, R. (1992). The MD5 Message-Digest Algorithm. Internet Engineering Task Force (IETF), RFC 1321.
[4]. Daemen, J., & Bertoni, G. (2011, January). The Keccak SHA-3 Submission [Online submission to NIST (Round 3)].
[5]. Kocher, P. (1996). Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Advances in Cryptology – CRYPTO '96 (pp. 104–113). Springer-Verlag.
[6]. Ferguson, N., Schneier, B., & Kohno, T. (2010). Cryptography Engineering: Design Principles and Practical Applications.
[7]. Bucci, M., Luzzi, R., Guglielmo, M., & Trifiletti, A. (n.d.). A Countermeasure against Differential Power Analysis Based on Random Delay Insertion. 2005 IEEE International Symposium on Circuits and Systems.
[8]. Mangard, S., Oswald, E., & Popp, T. (2007). Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer.
[9]. Saribekyan, H., & Margvelashvili, A. (2017, May 18). Security Analysis of Telegram.
[10]. Oren, Y., Kemerlis, V. P., Sethumadhavan, S., & Keromytis, A. D. (2015). The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications.
[11]. Daemen, J., & Rijmen, V. (2010). Understanding Cryptography: A Textbook for Students and Practitioners. In C. Paar & J. Pelzl (Eds.), Cryptographic Hash Functions (pp. 107-128). Springer.
[12]. Paar, C., & Pelzl, J. (2010). Understanding Cryptography: A Textbook for Students and Practitioners.
[13]. Bhunia, S., & Tehranipoor, M. M. (2017). Hardware Security: Design, Threats, and Safeguards.

[14]. Bloemer, J., & Otto, M. (2006). Securing RSA against Fault Analysis by Double Addition Chain Exponentiation.
[15]. Hancock, J. (2004). Jitter—Understanding it, Measuring It, Eliminating It: Part 1: Jitter Fundamentals. Agilent Technologies.
[16]. Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). Handbook of Applied Cryptography.
[17]. Gentle, E. (1998) Random Number Generation and Monte Carlo Methods.
[18]. Markantonakis, K., & Mayes, K. (2017). Secure Smart Embedded Devices, Platforms and Applications.
[19]. Schneier, B. (1994). Applied Cryptography: Protocols, Algorithms, and Source Code in C.
[20]. National Institute of Standards and Technology (NIST), (2012). Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised).
[21]. Tillich, S., & Großschädl, J. (2010). SHA-3 candidates and side-channel attack security.
[22]. Zhang, L., Kong, Y., Guo, Y., Yan, J., & Wang, Z. (2018). Survey on network flow watermarking: model, interferences, applications, technologies and security.IET Communications, 12(14), 1639–1648.
[23]. Bloemer, J., & Otto, M. (2006). Securing RSA against Fault Analysis by Double Addition Chain Exponentiation.
[24]. Kelsey, J., Schneier, B., & Ferguson, N. (1999). Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator. In Sixth Annual Workshop on Selected Areas in Cryptography.