IJEEC

International Journal of Electrical Engineering and Computing
Vol. 9, No. 1 (2025)

Original research paper
UDC 528.23:[005.22:658.51
DOI 10.7251/IJEEC2501027P

Residue Number System-Based Hash Function: Design and Cryptographic Analysis

Milija Pavlović[1], Tijana Talić[2], Boris Damjanović[3], Negovan Stamenković[1]

[1] Faculty of Natural Sciences and Mathematics, University of Priština, Kosovska Mitrovica, Serbia
[2]Pan-European university "Aperion" Banjaluka, Republic of Srpska, BiH
[3]Faculty of Information Technologies and Engineering, Union- Nikola Tesla University, Belgrade, Serbia

E-mail address: milija.pavlovic@pr.ac.rs, tijana.z.talic@apeiron-edu.eu, boris.damjanovic@fpsp.edu.rs, negovan.stamenkovic@pr.ac.rs

Abstract— **In this paper, we propose a novel cryptographic hash function based on the Residue Number System (RNS). The design leverages the inherent parallelism of RNS to enhance computational efficiency and resilience against certain classes of attacks. Through modular processing across independent residues, the function achieves notable improvements in hardware acceleration capabilities, making it particularly suitable for resource-constrained environments. To assess the robustness of the proposed function, we conduct a series of tests including the distribution of Hamming weights, bit balance evaluation, and avalanche effect analysis. The results demonstrate that the hash function produces uniformly distributed outputs, exhibits strong avalanche characteristics, and maintains high resistance to input structure correlation. These properties confirm the potential of the proposed RNS-based approach for secure and efficient hashing in modern cryptographic applications.**

Keywords-hash functions;residue number system; parallelism;avalanche effect;bit balance;

I. INTRODUCTION

In modern cryptography, hash functions are considered one of the most significant components, and their application encompasses a wide range, including ensuring security in communication, data integrity, and search optimization [1]. Although they have a seemingly simple foundation, the implementation and design of hash functions can represent an extremely complex process, for which a deep understanding of mathematical and computational principles is essential [2]. Hash functions can be described as methods that transform messages of varying lengths into hash values of fixed length [3]. The most significant properties of a hash function are:

- A hash function can be used on a block of data of varying sizes.
- The hash function produces an output of fixed length.
- The function H(x) is easy to compute for any given x, which makes it practical for implementation in both hardware and software (where x represents the input given to the hash function).
- For any given hash value h, it is computationally impractical to determine x such that H(x) = h. This characteristic is referred to in the literature as one-wayness.
- For any given block x, it is computationally impractical to identify $y \neq x$ such that H(y) = H(x). This property is known as weak collision resistance.
- It is computationally impractical to find any pair x and y such that H(x) = H(y). This property is called robust resistance to collisions [4].

Hash functions are categorized into two types: keyed and unkeyed. Keyed hash functions utilize both a message and a confidential key, whereas unkeyed hash functions depend exclusively on the input message [5]. Keyed functions render it exceedingly challenging for adversaries to produce identical hash results without possessing the secret key. Nevertheless, due to the enhancement of computational capabilities and cryptanalytic instruments, antiquated hash algorithms have grown progressively susceptible to assaults. Ensuring security standards and safeguarding against threats necessitates the creation of advanced, more resilient cryptographic algorithms [6].

The functionality of widely utilized hash algorithms, including notable examples like MD4, MD5, SHA-1, SHA-2, and SHA-3, relies on the execution of diverse arithmetic, logical, and algebraic operations. These algorithms have demonstrated unreliability over time, revealing specific vulnerabilities to collision-based attacks [7]. Moreover, the application of linear processes in some hash function algorithms may augment their susceptibility to particular cryptographic assaults [8].

Hash functions are predominantly utilized in cryptography, where they are essential for maintaining data confidentiality and validity. Cryptographic hash functions, specifically SHA-2

This paper is a revised and expanded version of the paper presented at the XXIV International Symposium INFOTEH-JAHORINA 2025

and SHA-3, are engineered to satisfy rigorous standards. A primary criterion is collision resistance, signifying that it is virtually infeasible to identify two distinct inputs yielding identical hash values [9]. Due to these characteristics, hash functions are essential in secure communication protocols, digital signatures, and message authentication.

Furthermore, hash functions are extensively employed in the organization and efficient retrieval of big databases. Hash tables facilitate rapid data retrieval and management, hence conserving time and computational resources greatly. This use is particularly significant in domains such as search engines, data repositories, and time-critical applications [10].

The advancement and refinement of hash functions encounter difficulties due to the exponential increase of data in contemporary society. The growing demand for data processing in distributed systems and cloud contexts underscores the significance of scalability and efficiency in hash functions. Furthermore, the rise of novel security vulnerabilities necessitates the continual enhancement of cryptographic algorithms to address the challenges presented by sophisticated assaults [11].

Currently, advancements in hash functions transcend their use in conventional systems, encompassing emerging technologies like blockchain and cryptocurrencies. Hash functions underpin processes such as "proof of work" in cryptocurrencies like Bitcoin, ensuring the irreversibility and verifiability of transactions within a distributed framework. In machine learning and artificial intelligence, hashing techniques are utilized to compress data and enhance algorithmic speed [12].

Current hash algorithms, including SHA-2 and SHA-3, have constraints on computational cost, vulnerability to side-channel attacks, and efficiency in hardware applications. The suggested methodology utilizing the Residue Number System (RNS) promotes efficiency through simultaneous execution of modular activities and bolsters resilience against certain attack vectors. Additionally, RNS-based hash algorithms can be refined for hardware acceleration, rendering them appropriate for resource-limited settings [13].

II. HASHING BASED ON TRADITIONAL MATHEMATICAL PRINCIPLES

Hash functions are among the fundamental algorithms in cryptography and computer science in general. Their primary role is to transform input messages of arbitrary length into output values of fixed length. Some of the most popular hash functions that have been used throughout history up to the present day include: MD4, MD5, RIPEMD, SHA-1, SHA-2, and SHA-3. Some of these functions are derived from one another, while others are based on entirely different approaches [14].

When it comes to the operations used to process messages in hash functions, the most common ones include:

- Bitwise operations – such as AND, OR, or XOR, are used to manipulate bits in a way that is optimized for processor efficiency.
- Bit rotations and shifts – help distribute information evenly throughout the hash.
- Message padding – adding bits to the end of the message so that its length becomes divisible by the block size.
- Mixing and permutations – the order of bits in the message is shuffled across different rounds to increase resistance against analytical attacks.
- Arithmetic operations – used to combine message components in an efficient and nonlinear manner.
- Compression functions – specialized mathematical functions that reduce the size of data to a fixed length [15].

III. RESIDUE NUMBER SYSTEM

The Chinese Remainder Theorem, which states that an integer is represented as a set of remainders with respect to a set of pairwise coprime moduli, is the foundation of the unique, unconventional, but incredibly effective Residue Number System (RNS) [16]. Although this method of representing numbers has been around since ancient Chinese mathematics, it wasn't until the 18th and 19th centuries that it was formally expressed mathematically [17] [18]. The need for parallel data processing and faster arithmetic operations without carry propagation, which are features of RNS, sparked a lot of interest in RNS in the second half of the 20th century, when it came to its application in computing and digital signal processing.

A significant benefit over the traditional binary system is the residue number system's high degree of parallelism, lower latency, and faster speed, which are made possible by the independent execution of operations like addition, subtraction, and multiplication on integers within each modulus. Because of these advantages, RNS is being used more and more in high-performance processors, embedded systems, digital signal processing systems, and image processing processors [19].

In cryptography, this system is especially significant because it can speed up modular arithmetic, which is essential for algorithms like RSA, ECC, and other public-key systems—especially in hardware implementations where side-channel attack resistance is critical [20]. Furthermore, RNS is ideally suited for implementation in hash functions because its parallel nature allows for efficient hardware implementation, which is crucial in systems where hashing must be quick and frequent (e.g., blockchain technologies, cryptographic databases, etc.), and modular arithmetic allows for the creation of a large number of distinct hash values with minimal collision risk.

The advantages of RNS in particular domains, like secure and embedded systems, make it a useful tool in contemporary digital arithmetic, despite some of its drawbacks, such as more complicated comparison operations, sign determination, and conversion between number systems [21].

IV. RESIDUE NUMBER SYSTEM HASH FUNCTION

Let a message $\boldsymbol{M}$ of length N be given. First, the message is padded with zeros to ensure that its length n is divisible by 64. Formaly, if $\lceil N/64 \rceil$ is the smallest integer greater than or equal to, then:

$$n = 64 \cdot \lceil N/64 \rceil$$

The padded message $\boldsymbol{M}^*$ has length n, and its content is:

$$\boldsymbol{M}^* = \boldsymbol{M} \,||\, 0_{n-N}$$

where 0_{n-N} is a string of zeros of length $n-N$, and $||$ denotes concantenation.

Then, the message $\boldsymbol{M}^*$ is divided into t blocks $\boldsymbol{B}_i$ each of length 64 bits:

$$\boldsymbol{M}^* = \boldsymbol{B}_1 \,||\, \boldsymbol{B}_2 \,||\, \ldots ||\, \boldsymbol{B}_t, \qquad t = \frac{n}{64}$$

A. Block processing using moduli

For each block $\boldsymbol{B}_i$, a set of moduli $\{x_1, x_2, ..., x_8\}$ is used, where:

$x_1 = 211, x_2 = 223, x_3 = 227, x_4 = 229, x_5 = 233, x_6 = 239,$
$x_7 = 241, x_8 = 251,$
and all moduli are pairwise coprime

After selecting of moduli, the remainder and quotient are computed for each modulus:

$$a_{i,j} = \boldsymbol{B}_i \mod x_i, \; A_{i,j} = \left\lfloor \frac{\boldsymbol{B}_i}{x_i} \right\rfloor, \; j \in \{1,2,...,8\}$$

B. Iterative processing

After computing $\{a_{i,j}, A_{i,j}\}$, further iterative processing is performed through the following steps:

1. **Computation of new values for remainders and quotients**:

- For remainders:
  $b_{i,j} = \left(A_{i,j} + a_{i,j}\right) \mod x_j, \; j \in \{1,2,...,8\}$.
- For quotients:
  $$\boldsymbol{B}_{i,j} = \left\lfloor \frac{\left(A_{i,j} + a_{i,j}\right)}{x_j} \right\rfloor, \quad j \in \{1,2,...,8\}.$$

2. **Continuation of processing with previous results**:

From the expressions above, further processing yields:

$$c_{i,j} = \left(\boldsymbol{B}_{i,j} + b_{i,j}\right) \mod x_j, \; C_{i,j} = \left\lfloor \frac{\left(\boldsymbol{B}_{i,j} + b_{i,j}\right)}{x_j} \right\rfloor.$$

The process is repeated for

$\{d_{i,j}, D_{i,j}\}$, $\{e_{i,j}, E_{i,j}\}$, $\{f_{i,j}, F_{i,j}\}$, $\{g_{i,j}, G_{i,j}\}$, $\{h_{i,j}, H_{i,j}\}$.

C. Formal expression for the general step

For each step k (where $k \geq 1$), the following is computed :

$$x_{i,j}^{(k)} = \left(X_{i,j}^{(k-1)} + x_{i,j}^{(k-1)}\right) \mod x_j,$$

$$X_{i,j}^{(k)} = \left\lfloor \frac{\left(X_{i,j}^{(k-1)} + x_{i,j}^{(k-1)}\right)}{x_j} \right\rfloor,$$

where $x_{i,j}^{(0)} = a_{i,j}$ and $X_{i,j}^{(0)} = A_{i,j}$.

This process continues until the final step, where we obtain the set of results $\{h_{i,j}, H_{i,j}\}$ for each block $\boldsymbol{B}_i$.

D. Algorithm I part

INPUT: Message M, length N

SET: x = [211, 223, 227, 229, 233, 239, 241, 251] // Moduli

SET: n = CEIL (N / 64) * 64 // Adjust message length to be divisible by 64

APPEND zeros to the end of message M until its length becomes n

DIVIDE message M into t blocks B[1], B[2], ..., B[t] of 64 bits each

// Initialization of values for computation

FOR each block B[i], where i is from 1 to t:

FOR each modulus x[j], where j is from 1 to 8:

// Step 1: Compute initial values

a[i][j] = B[i] MOD x[j]

A[i][j] = FLOOR(B[i] / x[j])

// Iterative computation

b[i][j] = (A[i][j] + a[i][j]) MOD x[j]

B1[i][j] = FLOOR((A[i][j] + a[i][j]) / x[j])

c[i][j] = (B1[i][j] + b[i][j]) MOD x[j]

C[i][j] = FLOOR((B1[i][j] + b[i][j]) / x[j])

```
d[i][j] = (C[i][j] + c[i][j]) MOD x[j]
D[i][j] = FLOOR((C[i][j] + c[i][j]) / x[j])

e[i][j] = (D[i][j] + d[i][j]) MOD x[j]
E[i][j] = FLOOR((D[i][j] + d[i][j]) / x[j])

f[i][j] = (E[i][j] + e[i][j]) MOD x[j]
F[i][j] = FLOOR((E[i][j] + e[i][j]) / x[j])

g[i][j] = (F[i][j] + f[i][j]) MOD x[j]
G[i][j] = FLOOR((F[i][j] + f[i][j]) / x[j])

h[i][j] = (G[i][j] + g[i][j]) MOD x[j]
H[i][j] = FLOOR((G[i][j] + g[i][j]) / x[j])

OUTPUT: h[i][j] and H[i][j] for all i in {1, ..., t} and j in {1, ..., 8}
```

E. Computing final residues using XOR operation

For each modulus x_j $(j = 1,...,8)$ and the corresponding values $a_j, b_j, c_j, d_j, e_j, f_j, g_j, h_j, H_j$, the final residue $ostaci_j$ is defined as:

$$ostaci_j = a_j \oplus b_j \oplus c_j \oplus d_j \oplus e_j \oplus f_j \oplus g_j \oplus h_j \oplus H_j$$

Here, $\oplus$ denotes the bitwise XOR logical operation.

F. Converting the result to an 8-bit binary representation

Each residue $ostaci_j$ is converted into a binary representation with exactly 8 bits, by adding leading zeros if necessary. This representation is denoted as b_ostaci_j:

$$b\_ostaci_j = bin\left(ostaci_j\right),$$

with leading zeros added to reach a total length of 8 bits.

G. Concatenation of binary representations

The final hash $hesF$ is obtained by concatenating the binary representations:

$$hesF = b\_ostaci_1 \,||\, b\_ostaci_2 \,||\, b\_ostaci_3 \,||\, b\_ostaci_4 \,||$$

$$||\, b\_ostaci_5 \,||\, b\_ostaci_6 \,||\, b\_ostaci_7 \,||\, b\_ostaci_8 .$$

Here, $||$ denotes the operation of concatenating binary strings.

H. Result

The final hash $hesF$ is a binary string of legth 64 bits, representing the concatenation of all individual binary residues.

I. Algortihm II part

```
// 1. Computing final remainders using XOR operation
FOR j ← 1 TO 8 DO
    remainders[j] ← a[j] XOR b[j] XOR c[j] XOR d[j] XOR e[j] XOR f[j] XOR g[j] XOR h[j] XOR H[j]

// 2. Converting results to 8-bit binary representation
FOR j ← 1 TO 8 DO
    b_remainders[j] ← ConvertTo8BitBinary(remainders[j])

// 3. Concatenating binary representations
binary_combined ← ""
FOR j ← 1 TO 8 DO
    binary_combined ← binary_combined + b_remainders[j]

// 4. Assigning the final hash
finalHash ← binary_combined

// Function to convert a number to an 8-bit binary representation
FUNCTION ConvertTo8BitBinary(number):
    binary_rep ← ToBinary(number)
    IF Length(binary_rep) < 8 THEN
        binary_rep ← AddLeadingZeros (binary_rep, 8 - Length(binary_rep))
    RETURN binary_rep
```

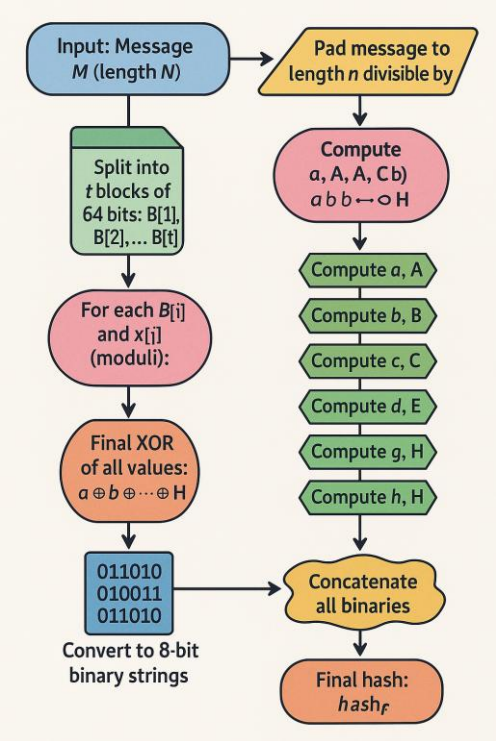


Figure 1. Graphical representation of the RNS hash function algorithm

A graphical representation of the complete algorithm is given in Fig. 1.

V. MEASURING EFFICIENCY

A key component of assessing cryptographic hash functions is their efficiency and output transformation quality measurement. Maintaining security characteristics like confusion and diffusion depends on several tests and measures used to guarantee that tiny input changes result in unpredictable and extensive output changes.

A. Distribution of Hamming weights in the outputs

The distribution of Hamming weights in the output hashes serves as one of the indicators of the hash function's quality.

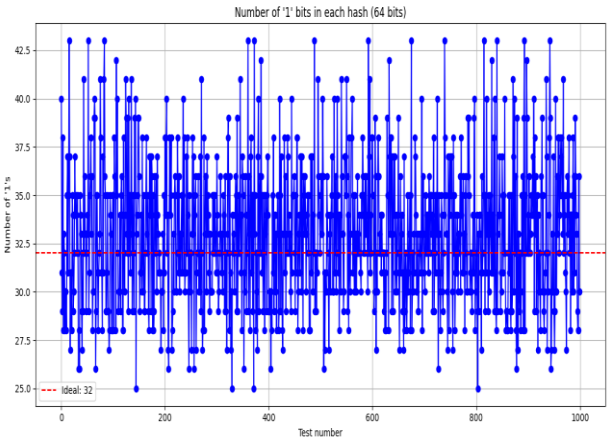


Figure 2. Distribution of Hamming weights in the outputs (64-bit)

Ideally, the output should contain approximately equal numbers of ones and zeros, which indicates that the hash function is unbiased and evenly distributes the output bits.

Using a sample of 1000 independent hashings of random inputs, Fig. 2. examines the distribution of Hamming weights in the outputs of a 64-bit hash function. The horizontal axis indicates the test index, and the vertical axis shows the number of bits in each generated hash that are set to logical one ("1"). With a distribution that oscillates within the bounds anticipated for a binomial distribution with parameters n = 64 and p = 0.5, the visual representation displays scattered points grouped around the mean value. The graph's dashed red line indicates the expected mean in this instance, which is μ = 32. The standard deviation of such a distribution is given by the formula

$$\sigma = \sqrt{np(1-p)} = \sqrt{64 \cdot 0.5(1-0.5)} = \sqrt{16} = 4$$

which implies that most of the data (approximately 95%) should fall within the interval {24, 40}. The values vary from about 25 to 43, suggesting no notable systematic deviations, therefore supporting this expectation. The distribution of points suggests that the hash function shows good diffusion and that there is no deterministic relationship between input changes and the locations of set bits in the output since it seems random without any clear pattern. A well-designed hash function's desirable quality is such a level of randomness, which guarantees that tiny input changes produce radically different outputs, therefore improving the cryptographic strength of the function. Statistically speaking, the findings show consistency and fairness in the distribution of "1" bits, which is vital for both cryptographic uses and non-cryptographic use cases like hash tables. Though it does not offer a full picture without further tests like frequency analysis, avalanche tests, or collision resistance analysis, this kind of study is a qualitative indicator of entropy and bit-level balance in the function. The results of this experiment, however, point to the hash function satisfying the criterion of statistical balance in its output bit.

B. Bit balance test

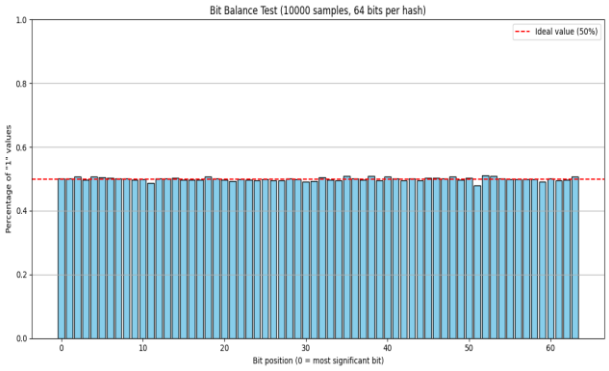


Figure 3. Bit balance test (10000 samples)

Closely related to the previous test is the bit balance test, which examines whether each individual bit in the output is

equally likely to be 0 or 1 across a large number of output values. A well-balanced hash function ensures that each bit behaves randomly, independent of the input structure.

The Fig. 3. presents the results of a bit balance test, a fundamental method for evaluating the entropy uniformity of hash functions. Specifically, the chart illustrates the percentage of bits set to logical one ("1") at each of the 64 bit positions within hash outputs, based on an analysis of 10,000 independently generated hashes. This analysis is grounded in the assumption that a well-designed hash function should produce outputs where each bit is equally likely to be "1" or "0," with a probability of 0.5.

Theoretically, this test relies on the hypothesis of uniform bit distribution, formally modeled as a set of independent binary random variables $X_i \sim Bernoulli(p)$, where $p = 0.5$ for each bit $i \in \{0, ..., 63\}$. Under this assumption, the expected number of "1" values at each bit position is $\mu = n \cdot p = 10{,}000 \cdot 0.5 = 5{,}000$. The standard deviation is calculated using the Bernoulli distribution formula:

$$\sigma = \sqrt{(n \cdot p \cdot (1-p))} = \sqrt{(10000 \cdot 0.5 \cdot 0.5)} \approx 50$$

which corresponds to a relative fluctuation tolerance of about ±1% (i.e., 100/10,000).

The significance of such an analysis lies in its ability to validate one of the key properties of cryptographic hash functions — diffusion — which refers to the desirable condition where output bits behave chaotically as a function of all input bits. The absence of structural asymmetries ensures that no exploitable statistical bias is present, which in turn strengthens resistance against cryptanalytic attacks such as collision finding or output prediction.

Beyond cryptographic contexts, bit balance is also important in non-security applications like hash tables and pseudorandom generators, where it ensures even distribution of values and minimizes clustering or collision probability.

Overall, the presented results indicate that the hash function under analysis exhibits strong entropy characteristics and uniformity, fulfilling a critical criterion for reliable and unbiased output distribution.

C. Avalanche effect

In order to evaluate the avalanche effect, the performance of the hash function was analyzed across input messages of different lengths. Key parameters describing the behavior of the function under minimal input perturbations were observed, including the minimum number of changed output bits (BMIN), the maximum number of changes (BMAX), the average number of changes (MEAN), the percentage of changed bits (P%), and indicators of deviation such as Average Bias (AB) and Average Probability deviation (AP).

TABLE 1. Key parameteres of Avalanche Test

Message length (bytes)	BMIN	BMAX	MEAN	P%	AB	AP
5	5	45	29.031000	45.360937	0.092781	0.625000
6	5	48	29.715833	46.430990	0.071380	0.671875
7	5	45	30.082857	47.004464	0.059911	0.625000
8	6	45	30.333437	47.395996	0.052080	0.609375

The results in Table 1. indicate that, for input messages ranging from 5 to 8 bytes in length, the minimum number of changed output bits varies between 5 and 6. Although this value is slightly lower than the ideal (where a larger number of changes would be preferred for stronger diffusion), it remains within acceptable limits for an initial security assessment. The maximum number of changes (BMAX) consistently falls within the range of 45 to 48 bits, representing solid coverage relative to the total 64 output bits. These results suggest relatively high sensitivity of the hash function to input modifications.

The average number of changed bits (MEAN) increases with the input length, from 29.03 for 5-byte messages to 30.33 for 8-byte messages. Similarly, the percentage of changed bits (P%) rises from 45.36% to 47.40%, approaching the ideal 50% expected for a well-designed avalanche effect. This trend indicates that the function distributes bit changes across the output more uniformly as the input length increases.

A more detailed analysis of deviations from ideal behavior, through the Average Bias (AB), reveals a decreasing trend as the message length increases, suggesting greater stability and reduced deviation from the expected bit-change probability. AB values decrease from 0.0928 for 5 bytes to 0.0521 for 8 bytes. Simultaneously, the Average Probability (AP) remains within a narrow range of 0.6094 to 0.6719, indicating moderate variation between minimum and maximum bit changes, without extreme deviations.

Overall, the tested hash function demonstrates a strong avalanche behavior, with the bit change percentage close to the ideal and low bias across the output bits. Although the BMIN is slightly lower than desired in some cases, the overall quality of bit diffusion indicates good cryptographic robustness for the analyzed message lengths.

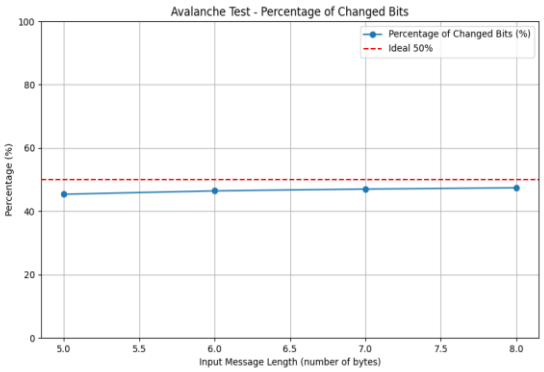


Figure 4. Avalanche Test – Percentage of Changed Bits

Fig. 4. presents the results of an avalanche effect analysis for a custom-designed hash function. The graph specifically illustrates the percentage of output bits changed as a function of the input message length, measured in bytes. The x-axis represents different input message lengths (5, 6, 7, and 8 bytes), while the y-axis shows the corresponding percentage of changed bits in the hash output upon flipping a single input bit.

The solid blue line with circular markers denotes the empirically observed percentage of changed bits. It can be observed that across all tested input lengths, the percentage remains relatively stable, ranging between approximately 45.36% and 47.40%. This indicates a consistent behavior of the hash function when subjected to small perturbations in the input.

For reference, a red dashed horizontal line is drawn at the 50% mark, representing the theoretical ideal for a perfect avalanche effect, where flipping one input bit would, on average, flip half of the output bits. The proximity of the empirical results to this ideal line suggests that the tested hash function exhibits a strong, though not perfect, avalanche property.

Importantly, the slight deviation below the ideal 50% indicates a minor bias in the bit transformation process; however, such deviations are common and acceptable in practical hash function design, especially given the randomness of input sampling and finite sample size (500 samples per input length). Additionally, the relatively small variance across different message lengths further supports the robustness of the function's design against input size variation.

Overall, the graph demonstrates that the hash function maintains a high level of diffusion across varying input sizes, which is a desirable property for ensuring unpredictability and resistance against differential cryptanalysis.

VI. CONCLUSION

From protecting digital signatures to verifying data in distributed systems, hash functions are an essential tool in contemporary cryptography that guarantee authenticity, integrity, and data security in a variety of applications. This analysis focuses on the RNS (Residue Number System) hash algorithm, which is straightforward, quick, and incredibly effective at producing hash values because it is based on binary operations, modular arithmetic, and multi-stage processing. By using RNS, numbers can be distributed effectively through smaller, mutually prime moduli, improving parallel computation and enhancing resistance to some attacks.

This algorithm's combination of binary formatting, merging, and modular operations is especially noteworthy because it produces a hash value with a high degree of entropy. Stronger resistance to reverse engineering attempts and collision finding is made possible by high entropy, which decreases the predictability of the output value even with small changes in the input. The algorithm is very adaptable for a range of applications, from tiny sensor systems to massive distributed networks, because it is made to accommodate varying input data sizes.

The algorithm's ease of use on various hardware and software platforms is further enhanced by its simplicity of implementation. Because of its architecture, it is simple to modify for a variety of processors, including those with limited resources. In the context of Internet of Things devices, where efficiency and speed are crucial considerations, this is especially crucial.

A thorough analysis of the suggested algorithm's defense against contemporary cryptanalytic attacks, such as collision and brute-force attacks, as well as a study of the uniformity of the hash value distribution, are required for a thorough evaluation. To objectively evaluate the algorithm's benefits and potential drawbacks, it's also critical to compare its security and performance features with those of current standards like SHA-2 and SHA-3.

Its use in distributed consensus systems and blockchain technologies, where speed, security, and attack resistance are essential, may be an especially intriguing area of application. In this regard, further refinement of the algorithm for hardware implementations (like FPGA and ASIC platforms) may greatly expand its applicability, allowing for the development of security solutions that are quicker and use less energy.

The suggested RNS-based hash algorithm, in summary, shows great promise for use in fields like change detection systems, digital signatures, user authentication, safe password storage, and data integrity verification. It is a strong contender for additional development and use in contemporary cryptographic solutions due to its simplicity, high entropy, and modular flexibility. This algorithm may make a significant future contribution to the fields of cryptography and information security with additional optimization and thorough security assessment.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support from the Serbian Ministry of Science, Technological Development and Innovation (Contract No. 451-03-65/2024- 03/200124). This work has been partially funded by the University of Pristina in Kosovska Mitrovica, Faculty of Science and Mathematics, Serbia, under the project: IJ-2302 (Optimization of neural network).

REFERENCES

[1] C. Paar and J. Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners*, 2nd ed., Springer, 2010.

[2] [2] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 3rd ed., CRC Press, 2020.

[3] S. M. S. Eldin, A. A. Abd El-Latif, S. A. Chelloug, M. Ahmad, A. H. Eldeeb, T. O. Diab, W. I. Al Sobky, and H. N. Zaky, "Design and analysis of new version of cryptographic hash function based on improved chaotic maps with induced DNA sequences," IEEE Access, vol. 11, pp. 101694–101709, 2023.

[4] W. Stallings, Network Security Essentials: Applications and Standards, 6th ed. Pearson Education, 2016.

[5] M. Bellare and P. Rogaway, *Introduction to Modern Cryptography*, Lecture Notes, University of California, Davis, 2005.

[6] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, Handbook of Applied Cryptography. Boca Raton, FL: CRC Press, 1997.

[7] A. Sadeghi-Nasab and V. Rafe, "A Comprehensive Review of the Security Flaws of Hashing Algorithms," *Journal of Computer Virology and Hacking Techniques*, vol. 19, no. 1, pp. 1–16, 2022.

[8] M. Hassan, J. Vliegen, S. Picek, and N. Mentens, "A systematic exploration of evolutionary computation for the design of hardware-oriented non-cryptographic hash functions," in Proc. Genetic and Evolutionary Computation Conf., 2024, pp. 1255–1263.

[9] X. Wang and H. Yu, "How to break MD5 and other hash functions," in *Advances in Cryptology – EUROCRYPT 2005*, R. Cramer, Ed. Berlin, Heidelberg: Springer, 2005, pp. 19–35.

[10] A. Kirsch and M. Mitzenmacher, "Less hashing, same performance: Building a better bloom filter," in *Proceedings of the 14th Annual European Symposium on Algorithms (ESA)*, Zurich, Switzerland, 2006, pp. 456–467.

[11] J. Black, P. Rogaway, T. Shrimpton, and M. Stam, "An Analysis of the Blockcipher-Based Hash Functions from PGV," Journal of Cryptology, vol. 23, no. 4, pp. 519–545, Jul. 2010, doi: 10.1007/s00145-010-9071-0.

[12] Y. Yang and X. Zhang, "A novel hash function based on multi-iterative parallel structure," Wireless Pers. Commun., vol. 127, pp. 2979–2996, 2022.

[13] D. Schinianakis and T. Stouraitis, "Residue Number Systems in Cryptography: design, challenges, robustness," in Springer eBooks, 2015, pp. 115–161. doi: 10.1007/978-3-319-14971-4_4.

[14] A. K. Lenstra and E. R. Verheul, "Selecting cryptographic key sizes," Journal of Cryptology, vol. 14, no. 4, pp. 255–293, Aug. 2001, doi: 10.1007/s00145-001-0009-4.

[15] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, Aug. 1987 [Digests 9th Annu. Conf. Magnetics Japan, p. 301, 1982].

[16] N. Stamenković, "Isomorphic transformation and its application to the modulo (2n + 1) channel for RNS based FIR filter design," University Thought - Publication in Natural Sciences, vol. 9, no. 2, pp. 63–68, Jan. 2019, doi: 10.5937/univtho9-21821.

[17] P. V. A. Mohan, Residue number systems: Theory and Applications. Birkhäuser, 2016.

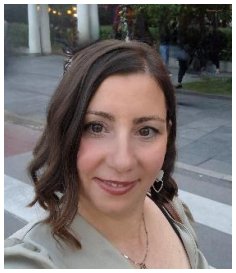
[18] N. S. Szabó and R. I. Tanaka, Residue arithmetic and its applications to computer technology. 1967.

[19] K. Isupov, "High-Performance computation in residue number system using Floating-Point arithmetic," Computation, vol. 9, no. 2, p. 9, Jan. 2021, doi: 10.3390/computation9020009.

[20] D. M. Schinianakis, A. P. Fournaris, H. E. Michail, A. P. Kakarountas, and T. Stouraitis, "An RNS Implementation of an Fp Elliptic Curve Point Multiplier," IEEE Transactions on Circuits and Systems I Regular Papers, vol. 56, no. 6, pp. 1202–1213, Nov. 2008, doi: 10.1109/tcsi.2008.2008507.

[21] J. -c. Bajard and L. Imbert, "a full RNS implementation of RSA," IEEE Transactions on Computers, vol. 53, no. 6, pp. 769–774, Apr. 2004, doi: 10.1109/tc.200

Milija Pavlović received his BSc degree in 2022 and MSc degree in 2024 from the Faculty of Natural Sciences and Mathematics in Kosovska Mitrovica, Serbia. He is currently a first-year PhD student and a teaching assistant at the Department of Informatics, Faculty of Natural Sciences and Mathematics, in Kosovska Mitrovica. His research interests include cryptography and modular arithmetic.



Tijana Talić graduated in 2008 from the Faculty of Natural Sciences and Mathematics in Banja Luka (Bosnia and Herzegovina), Department of Mathematics and Informatics. She received her PhD in 2019 from the Faculty of Information Technology of the Pan-European University Apeiron, Banja Luka. Since 2019, she has been employed at the same faculty as a professor in the field of computer science and information science and bioinformatics.

Boris Damjanović received his BSc degree in 2008 from the College of Economics and Informatics in Prijedor, Bosnia and Herzegovina, and his MSc and PhD degrees in 2010 and 2016, respectively, from the Faculty of Organizational Sciences, University of Belgrade, Serbia. He is an associate professor at Union Nikola Tesla University in Belgrade. His main areas of interest are cryptography, computer security, and applied informatics.

Negovan Stamenković received his BSc degree in 2006 from the Faculty of Technical Sciences in Kosovska Mitrovica, Serbia, specializing in Electronics and Telecommunications. He obtained his PhD degree in 2011 from the Faculty of Electronic Engineering, University of Niš, Serbia. He is a full professor at the Faculty of Natural Sciences and Mathematics in Kosovska Mitrovica. His main research interests are in digital signal processing, computer engineering, and modular arithmetic.