A Constraint Satisfaction Approach for Estimating the RSA Prime Factors towards Known Bits Factorization Attacks

Daniel Asiedu[1,*], Patrick Kwabena Mensah[1], Peter Appiahene[2], Peter Nimbe[1]

[1]Department of Computer Science and Informatics, University of Energy and Natural Resources, Sunyani, Ghana

[2]Department of Information Technology and Decision Sciences, University of Energy and Natural Resources, Sunyani, Ghana

Emails: daniel.asiedu.stu@uenr.edu.gh; patrick.mensah@uenr.edu.gh; peter.appiahene@uenr.edu.gh; peter.nimbe@uenr.edu.gh

Abstract

The Rivest–Shamir–Adleman (RSA) cryptosystem is one of the most prevalently utilized public-key cryptographic systems in current practice. Prior investigations into vulnerabilities of this cryptosystem have concentrated on diminishing the complexity associated with the integer factorization challenge, which is integral to the RSA modulus, expressed as $N=pq$. Possessing partial knowledge about the least significant digits (LSDs) of both p and q is a common assumption attacker's advantage to enable the polynomial-time factorization of N, ultimately undermining the security of RSA. This paper presents a novel heuristic algorithm predicated on the Constraint Satisfaction Problem (CSP) principles, which estimates k-LSD pairs of the RSA prime factors, p and q. The proposed Generate and Test (GT) and Backtracking with Heuristic Variable Ordering (BHVO) solver guarantees polynomial-time factorization of known bits by iteratively refining candidate pairs and eliminating invalid combinations through effective constraint propagation. The proposed approach obviates the requirement for specialized hardware for side-channel attacks to reveal a portion of p and q. In our results, we have successfully estimated up to 5-LSDs of p and q with a reduced number of iterations and factored 2048 bits, N based on the known 4-LSDs of the prime in polynomial time. Our research lays the groundwork for factorization algorithms that require partial knowledge of the prime factors. We have highlighted the possible vulnerabilities linked to existing RSA key generation techniques. These may make RSA moduli susceptible to the attacks discussed in this study and proposed countermeasures to ensure secure prime generation.

Keywords: RSA cryptosystem; Constraint satisfaction problem; key exposure attacks; Factorization attacks; Cryptography; Attack mitigation

1. Introduction

The swift progress in computational capabilities and the growing necessity for secure communication have rendered the RSA cryptosystem an essential pillar of contemporary cybersecurity. Since its establishment, the RSA cryptosystem has been regarded as the most prevalent asymmetric key cryptosystem. Within its key generation algorithm, an RSA modulus, $N=pq$, is derived where p and q, referred to as RSA primes, are two distinct prime numbers such that $p<q<2p$ [1]. The security of RSA is fundamentally based on the computational challenge of decomposing N into p and q, with large prime numbers selected to guarantee the impracticality of factorization within a reasonable duration [2]. Nevertheless, the foundational dependence on the difficulty of factorizing large prime numbers has engendered apprehensions regarding the long-term sustainability of this extensively utilized algorithm [3], [4]. Should an individual possess the capability to factor these numbers efficiently, they could compromise the security of RSA and gain access to confidential information. Similarly, the increasing

DOI: https://doi.org/10.54216/JCIM.160104
Received: October 24, 2024 Revised: January 11, 2025 Accepted: February 09, 2025

computational capabilities of conventional computers have made brute-force attacks on RSA more attainable, mainly when there is partial information on the prime factors of the modulus N, heightening the associated security challenges. As numerous cryptographic frameworks depend on the difficulty of integer factorization [5], [6], the investigation of efficient algorithms for prime factorization that surpass existing methodologies or exploit specific attributes of the prime factors can exert a significant influence on the current applications of cryptographic algorithms that hinge on the complexity of factorization.

It remains unclear whether the prime factorization problem possesses polynomial computational complexity and, consequently, if it belongs to the complexity class P. However, most widely used cryptographic algorithms are based on the premise that factorization is a complex problem. Resources that depend on this assumption include website certificates and Bitcoin wallets. Moreover, without an explicit lower bound on computational complexity, many essential services may become susceptible to security threats in the longer term. Thus, exploring novel methods that provide computational benefits is encouraged [7].

When there is no knowledge of p and q, the factorization process becomes impractically complex as N increases, necessitating exponential time when employing traditional algorithms. Conversely, given some partial knowledge regarding p and q, the complexity of factorisation can be significantly reduced, and a solution may be obtained in polynomial time, even for a large RSA modulus [8]. This principle has important implications for the security of cryptographic systems. For example, suppose certain aspects of p and q, such as their most significant digits (MSDs) or particular modular relationships, are disclosed or guessed. In that case, the factorisation challenge can shift from being computationally prohibitive to manageable within reasonable timeframes. This vulnerability underscores a vital aspect of cryptographic design: safeguarding the prime numbers and any partial information about them.

Historically, the exposure of partial knowledge regarding p and q has mainly depended on side-channel attacks, which take advantage of physical emissions or processing behaviours in specific hardware setups [9], [10], [11], [12], [13], [14]. These methods demand precise instruments and specialized equipment, often involving complicated experimental setups. As a result, their practical implementation is cumbersome and highly dependent on the particular device being targeted.

The current study proposes a new heuristic algorithm based on the CSP that estimates a portion of p and q without relying on side-channel information. This method provides a software-oriented solution that is adaptable, scalable, and free from hardware constraints. This method extends the scope of prime factorization techniques through computational heuristics, thereby addressing the limitations of traditional side-channel methods.

This research contributes to the field of cryptology as follows;

- Given RSA modulus N, a new heuristic CSP-based algorithm that estimates the LSDs of the RSA prime factors, p and q.
- Laying the foundation for factorization algorithms that rely on known-bits prime, eliminating the need for specialized hardware devices to conduct side-channel attacks to reveal part of p and q.
- Highlighting patterns and vulnerabilities in RSA modulus construction that may inform the development of more secure key generation practices.

This study aims to develop a new heuristic algorithm inspired by CSP for estimating the LSDs of RSA prime factors. We strive to explore the implications of this additional information on the “known bits prime” factorization process in polynomial time.

This research expands on the idea that having partial knowledge of p and q may enable the polynomial-time factorization of N. We examine the practical applications and computational implications of utilizing partial prime information. The findings show the potential of these methods within cryptanalysis. Strategies within key generations to protect against the risk of exposing critical information are also included, together with the necessity of enhancing randomness in RSA prime generation to reduce the risks linked to the factorization of known bits prime.

2. Related Work

Several studies have shown that prime factorization of larger RSA module (exceeding 1024 bits) can be achieved with polynomial time complexity. Many of these approaches focus on the assumption that an attacker possesses knowledge of several bits of the prime factors p and q, thereby diminishing the difficulty of factoring N. In [15], the authors proposed partial key exposure attacks, positing that if certain bits of the prime numbers are known to the adversary, this can facilitate the factorization of N. They demonstrated that knowing 2/3 of the bits of either p

DOI: https://doi.org/10.54216/JCIM.160104
Received: October 24, 2024 Revised: January 11, 2025 Accepted: February 09, 2025

or q is adequate for this purpose. Subsequently, [16] improved upon this finding by reducing the necessary known bits to 1/2 using the LLL algorithm. The attack described by Herrmann and May later required that the known bits be organized into random blocks [17]. Heninger and Shacham's approach, inspired by the so-called cold boot attack, targets memory in electronic chips to recover bits of private keys, assuming those bits are sourced from random positions [18]. They successfully executed their attack when 0.57 bits of the primes were known. It is important to note that this fraction is significantly lower when assessing the random bits of the RSA private exponent, d (specifically, dp and dq in the context of CRT-RSA). [19]analyse the reconstruction algorithm by Heninger and Shacham from a combinatorial point of view. The algorithm is an innovative brute-force method on the total search space of unknown bits of p and q, which prunes the infeasible solutions. However, their approach requires some random bits of the primes. In contrast to existing methodologies, [20]leverage k number of least significant bits (LSBs) from the primes, where k is less than, which is a notably small value and factor N up to 2048 bits.

While the papers present encouraging results, the attacks discussed operate under the assumption of having partial knowledge of the bits of p and q to boost efficiency, requiring particular hardware for side-channel attacks to uncover segments of p and q. Such data is generally complicated to access in real-world attack scenarios, making attempts to factorize known prime bits impractical. The dependence on hardware for side-channel attacks underscores the need for different approaches to infer these values' characteristics. This approach creates new opportunities for cryptanalysis that bypass these impractical assumptions, which could improve and optimize the effectiveness of algorithms aimed at prime factorization with known bits prime. Our approach builds upon the studies of prime factorization with known bits primes (case of LSDs), given the knowledge of only N to enhance polynomial time factorization in practical settings.

3. Proposed methodology

Problem Formulation

Our method utilizes the column multiplication technique, commonly called long multiplication. In this technique, when multiplying multi-digit numbers, we carry out a sequence of single-digit multiplications and subsequently combine the outcomes according to their respective place values (refer to Figure 1). For a generalization, for multiplying two n-digit balanced numbers, as presented below:

$$X = p_1 p_2 p_3 \dots p_n$$
$$Y = q_1 q_2 q_3 \dots q_n$$

Where p_i and q_j are the digits of the numbers, X and Y respectively, with p_1 and q_1 being the most significant digits (leftmost) and p_n and q_n being the least significant digits (rightmost). The general form for a partial product is $Y_j \times X$, shifted $(n-j)$ places to the left. Ultimately, the final product is obtained by summing all the partial products column by column, progressing from right to left. Thus,

Let X represent an $(n+1)$-digit number:

$$X = p_n \,.\, 10^n + p_{n-1} \,.\, 10^{n-1} + \dots + p_1 \,.\, 10^1 + p_0 \,.\, 10^0$$

Let Y represent an $(m+1)$-digit number:

$$Y = q_m \,.\, 10^m + q_{m-1} \,.\, 10^{m-1} + \dots + q_1 \,.\, 10^1 + q_0 \,.\, 10^0$$

Where p_i and q_j are the digits of X and Y respectively.

We Compute partial products by multiplying each digit of $X(p_i)$ with each digit of $Y(q_j)$, weighted by their respective positional values:

$$PP_{ij} = p_i \,.\, q_j \,.\, 10^{i+j}$$

Where PP_{ij} is the partial product at $ij^{th} position$.

When computing each PP_{ij}, if the result of $p_i \,.\, q_j > 9$, carry digits are calculated and added to the subsequent positional place:

$$r_{ij} = PP_{ij} \bmod 10$$

$$c_{ij} = \left\lfloor \frac{PP_{ij}}{10} \right\rfloor$$

DOI: https://doi.org/10.54216/JCIM.160104
Received: October 24, 2024 Revised: January 11, 2025 Accepted: February 09, 2025

Where r_{ij} is a remainder for the current positional value, and c_{ij} is a carry to the next higher positional value. Then, add partial products (column-wise) for each positional value. If the sum of any column exceeds 9, compute the carry c and the remainder r as follows:

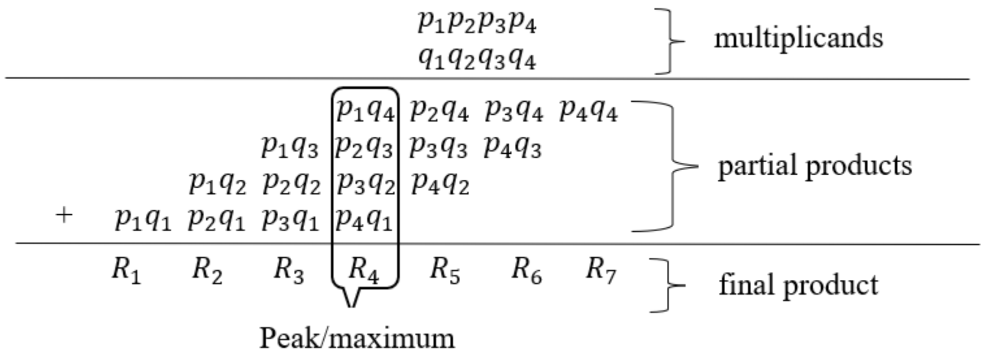
$$c_v = \left\lfloor \frac{s_d \ in \ column \ v}{10} \right\rfloor, r_v = (s_d \ in \ column \ v) \ mod \ 10$$

Where s_d is the sum of digits.

Add c_v to the next higher positional value

The result R is the concatenation of the remainders r_v from all positional values after carry propagation.

For simplicity, we would limit X and Y to a 4-digit balance number, as shown in Figure 1.



$$\begin{array}{cccccccc}
 & & & & p_1p_2p_3p_4 & & & \\
 & & & & q_1q_2q_3q_4 & & & \\
\hline
 & & & \boxed{p_1q_4} & p_2q_4 & p_3q_4 & p_4q_4 & \\
 & & p_1q_3 & p_2q_3 & p_3q_3 & p_4q_3 & & \\
 & p_1q_2 & p_2q_2 & p_3q_2 & p_4q_2 & & & \\
+ & p_1q_1 & p_2q_1 & p_3q_1 & p_4q_1 & & & \\
\hline
 & R_1 & R_2 & R_3 & R_4 & R_5 & R_6 & R_7
\end{array}$$

Figure 1. 4-digit balance multiplication process

When two numbers consist of the same number of digits, as illustrated in Figure 1 (4-digit numbers), the addition process reveals a distinct pattern when analysed column by column, resembling a parallelogram shape. The digits from the lowest partial product define the base of this parallelogram. At the same time, its sides are created by progressively incorporating additional digits as one moves leftward, commencing with a single digit in the unit's column and increasing towards the most significant digits. This pattern advances as the count of terms (p_iq_j) in each summation rises linearly until it hits a maximum (see Figure 1), which aligns with the number of digits in the multiplicand. Following this peak, there is a symmetrical decrease. This organized approach aids in structuring the summation, guaranteeing that each digit is incorporated according to its positional value, ultimately leading to the precise result.

In our attempt to reverse the multiplication process to reconstruct the original multiplicands, $p_1, p_2, p_3, \dots, p_n$ and $q_1, q_2, q_3, \dots, q_n$, we formulate the addition of the partial products as a CSP. We assume that the multiplicands, $p_1, p_2, p_3, \dots, p_n$ and $q_1, q_2, q_3, \dots, q_n$ are distinct prime factors of the RSA modulus N.

Definition:

Given

- set of domain variables $P = \{p_1, p_2, p_3, \dots, p_n\}$ and $Q = \{q_1, q_2, q_3, \dots, q_n\}$
- Finite set of domain values $D = \{d_1, d_2, d_3, \dots, d_n\}$, where $P_i, Q_i \in D_i$
- Set of constraints $C = \{c_1, c_2, c_3, \dots, c_n\}$ where each constraint C_i is a relation defined over a subset of the variables $P \ and \ Q$.

The objective is to find pairs $\{p, q\}$ such that $p \in D_i, q \in D_i$ for all $i = 0,1,2,3, \dots, n$ for each constraint C_i that $P \ and \ Q$ must satisfy.

Each column (sum of the partial products) can be represented as a constraint (from right to left), p_iq_j as term(s) in each constraint, and R_i as the sum of the column-base final product. The addition of partial products, where the number of digits contributing to the columns increases up to a number of terms (p_iq_j) equal to the number of digits in the multiplicand and then decreases, can be used to systematically generate an arbitrary number of columns and terms up to the length of the multiplicands. This pattern structure ensures the process can be extended to accommodate any number of digits in the multiplicands. To fully recover the multiplicands, we stop where the addition process reaches a peak (a constraint C_i having terms p_iq_j equal to the length of the multiplicands).

DOI: https://doi.org/10.54216/JCIM.160104
Received: October 24, 2024 Revised: January 11, 2025 Accepted: February 09, 2025

For n-digits number, we formulate the CSP as follows, starting from the LSD (one's column):

Variables: $p_1p_2p_3 \dots p_n$ and $q_1q_2q_3 \dots q_n$

Domains: $D = \{0,1,2,\dots,9\}\ \forall\ p_i, q_i \in D$, where $i = 1,2,3,\dots,n$

Constraints:

$$p_nq_n\ mod\ 10 = r_n$$

$$(p_{n-1}q_n + a_1) + p_nq_{n-1}\ mod\ 10 = r_{n-1}$$

$$(p_{n-2}q_n + a_2) + (p_{n-1}q_{n-1} + a_3) + p_nq_{n-2} + b_1\ mod\ 10 = r_{n-2}$$

$$(p_{n-3}q_n + a_4) + (p_{n-2}q_{n-1} + a_5) + (p_{n-1}q_{n-2} + a_6) + p_nq_{n-3} + b_2\ mod\ 10 = r_{n-3}$$

$$\vdots$$

$$(p_1q_n + a_n) + (p_2q_{n-1} + a_{n+1}) + (p_3q_{n-2} + a_{n+2}) + \cdots + p_nq_1 + b_n\ mod\ 10 = r_l$$

where,

p_i are the digits of p and q_i are the digits of q

p_iq_j are the sum of the product terms

p_nq_n are last digit position of $p\ and\ q$

r_i are digits of the final product starting from the LSD r_n

$a_1$is the quotient of $p_nq_n\ mod\ 10$ (the carry term in the first constraint)

a_2 is the quotient of $(p_{n-1}q_n + a_1)\ mod\ 10$ (the carry of the first term in the second constraint)

a_3 is the quotient of $p_nq_{n-1}\ mod\ 10$ (the carry of the second term in the second constraint)

a_4 is the quotient of $(p_{n-2}q_n + a_2)\ mod\ 10$ (the carry of the first term in the third constraint)

a_5 is the quotient of $(p_{n-1}q_{n-1} + a_3)\ mod\ 10$ (the carry of the second term in the third constraint)

a_6 is the quotient of $p_nq_{n-2}\ mod\ 10$ (the carry of the third term in the third constraint)

b_1 is the first carry of the sum of the terms in the constraint two

b_2 is the second carry of the sum of the terms in the constraint three

$a_n, a_{n+1}, \dots, b_n$ are the carries propagating to the peak constraint (see Figure 1)

r_l is the digit position of the final product that corresponds to the peak constraint (see Figure 1)

Note that each constraint (the sum of the partial products) corresponds to a digit position of the final product, which represents the RSA modulus beginning from the LSD (one's column). Additionally, the number of constraints is equal to the length of $p\ and\ q$ in the situation where we want to recover the entire p_i and q_j digits position.

To determine the length of $p\ and\ q$, we compute $\left\lceil \frac{k}{2} \right\rceil$, where k is the length of the RSA modulus N(digit-wise).

The objective is to find the feasible solution for $p_1, p_2, p_3, \dots, p_n$ and $q_1, q_2, q_3, \dots, q_n$ such that, $\frac{q_{n-j}(p_i)}{q_{n-j}} = p$ at any position of j and $\frac{N}{p} = q$, where $p, q \in \mathbb{Z}^+$, n is the length of $p\ and\ q$, N is the RSA modulus and $j = 0,1,2,..,n-1$.

For simplicity, we formulate values of $p\ and\ q$ with 4-digit balanced prime numbers as follows;

Variables: $p_1p_2p_3p_4$ and $q_1q_2q_3q_4$

Domains: $p_1, p_2, p_3, p_4 = \{0,1,2,\dots,9\}$ and $q_1, q_2, q_3, q_4 = \{0,1,2,\dots,9\}$

Constraints:

$$p_4q_4\ mod\ 10 = r_4$$

$$(p_3q_4 + a_1) + p_4q_3\ mod\ 10 = r_3$$

DOI: https://doi.org/10.54216/JCIM.160104
Received: October 24, 2024 Revised: January 11, 2025 Accepted: February 09, 2025

$$(p_2q_4 + a_2) + (p_3q_3 + a_3) + p_4q_2 + b_1 \; mod \; 10 = r_2$$

$$(p_1q_4 + a_4) + (p_2q_3 + a_5) + (p_3q_2 + a_6) + p_4q_1 + b_2 \; mod \; 10 = r_1$$

Algorithm 1 facilitates the CSP formulation, given the RSA modulus N and the number of digits to estimate. When the $k - LSDs$ is not set, the Algorithm 1 assumes all digit's position. Otherwise, it returns constraints up to $k - LSDs$ of $p \; and \; q$.

Algorithm 1: Constraints and Domain Generation

```
Input: RSA modulus (string of numeric digits), $\boldsymbol{k - LSDs}$
Output: constraints, domains
1  function Generate_ConstraintsDomains (RSA modulus N, $\boldsymbol{k - LSDs}$)
2    If $\boldsymbol{k - LSDs}$ exists do
3       $\boldsymbol{n} \leftarrow \boldsymbol{k - LSDs}$ // Generate constraints up to $\boldsymbol{k - LSDs}$
4    else
5       $\boldsymbol{length} \leftarrow \boldsymbol{len}(\boldsymbol{N})$ // Get the length of the input digits
6       $\boldsymbol{n} \leftarrow \left\lceil \frac{\boldsymbol{length}}{\boldsymbol{2}} \right\rceil$ // Generate constraints up to the length of p and q
7    end if
8    R_values ← [int($\boldsymbol{d}$) for $\boldsymbol{d}$ in $\boldsymbol{N}$] // Parse digits into R values
9    constraints ← [ ] // Initialize an empty list to store constraints
10   domains ← { } // Initialize an empty dictionary to store domains
11    for $\boldsymbol{k}$ from $\boldsymbol{n}$ to 0 step -1 do
12       equation ← "" // Initialize an empty string for the current equation
13       $\boldsymbol{t} \leftarrow 1$ // Initialize carry index counter
14       for $\boldsymbol{i} \leftarrow (\boldsymbol{n - k + 1})$ to $\boldsymbol{n + 1}$ do
15          $\boldsymbol{j} \leftarrow \boldsymbol{2n - k + 1 - i}$ // Calculate the corresponding j value
16          if $\boldsymbol{1 \leq j \leq n}$ do
17             if $\boldsymbol{i = n}$ do
18                term ← $\boldsymbol{p_i} * \boldsymbol{q_j}$ // Current term
19                equation.append(term) // add current term to equation
20             else
21                term ← ($\boldsymbol{p_i} * \boldsymbol{q_j} + \boldsymbol{a_t}$) // Current term with carry
22                equation.append(term) // add current term to equation
23                domains[$\boldsymbol{a_t}$] ← (0, 9) // Add carry to the domains
24                $\boldsymbol{t} \leftarrow \boldsymbol{t + 1}$ // Increment carry index
25             end if
26             end if
27       Set equation_str to join elements of equation using " + "
28       R_value ← R_values[$\boldsymbol{-k}$] // Get the R value for this equation
         #adding the partial sum carry terms
```

DOI: https://doi.org/10.54216/JCIM.160104
Received: October 24, 2024 Revised: January 11, 2025 Accepted: February 09, 2025

```
        if $k > 2$ do
          full_equation ← (equation_str + $b_k$) mod 10 = R_value  // Add carry term ($b_k$)
           constraints.append(full_equation)  // Add the completed equation to the constraints
          domains[$b_k$] ← (0, 9)  // Add carry variable to domains
        else
          full_equation ← equation_str mod 10 = R_value  // Complete the equation
          constraints.append(equation)  // Add the completed equation to the constraints
        end if
        domains[$p_k$] ← (0, 9)  // Add domain values of $p$ at position $k$
        domains[$q_k$] ← (0, 9)  // Add domain values of $q$ at position $k$
    end for
    constraints.reverse  // Reverse constraints to ensure proper order
    return {constraints, domains}  // Output the constraints and domains
end function
```

Algorithm 1 transforms a sequence of digits from the RSA modulus into a well-organized set of constraints and domains suitable for solving as a CSP. Each digit from the input is regarded as a constant, while variables are created for the terms in equations that model relationships between the digits. It analyzes the input digits, formulating modular equations that include variables p_i , q_j, a_t, b_k. To handle carryovers and modular arithmetic, remainder terms a_t and b_k are incorporated. The constraints are generated sequentially, beginning with the least significant digit (LSD) and progressing backwards. Each constraint integrates modular arithmetic and carryover terms as necessary. The construction of constraints occurs in reverse order to optimize processing and maintain alignment between the equations and the input structure. This results in a framework of constraints and domains that systematically addresses digit-based problem-solving challenges.

Solving the Formulated CSP

A solution to a CSP involves assigning a distinct value to each variable in such a way that all imposed constraints are fulfilled. The structure of our formulated CSP necessitates using a systematic search algorithm that explores a problem instance's search space, ensuring that either an optimal solution is identified or the absence of a solution is definitively established. This essential characteristic of systematic search algorithms is referred to as completeness.

Algorithm 1 presents a systematic approach for constraint satisfaction (CS) to estimate the digit pairs of p and q. The algorithm formulates constraints based on the lengths of p and q, or the k-LSDs, where k is the number of LSDs positions required, facilitating an organized retrieval of (p, q) pairs. Consequently, our approach integrates two systematic search techniques: Generate and Test (GT) and Backtracking with Heuristic Variable Ordering (BHVO). Initially, the GT algorithm proposes a potential solution, followed by testing its validity to determine whether it meets the original constraints using the BHVO. Within this framework, every possible combination of variable assignments is systematically generated and evaluated against the constraints. The first combination that meets all constraints is deemed the solution. This approach not only ensures the identification of a solution when one exists but also often does so more efficiently when k is small. The following section elaborates on our method utilizing GT and BHVO.

LSDs pair (p_i, q_i) estimation based on the GT and BHVO

We examine the digit pairs (p_i, q_i) from a combinatorial perspective, beginning with the LSD. We employ combinatorial generation methods to investigate all potential pairs of digits (p_i, q_i) that meet the specified modular and arithmetic conditions. The primary objective is to systematically evaluate viable combinations while utilizing the constraints to filter out those that are not feasible. Our analysis will concentrate on the conventional RSA relationship $(N = pq)$, particularly the factorization techniques outlined in [21]. The success of our algorithm only necessitates the knowledge of N.

DOI: https://doi.org/10.54216/JCIM.160104
Received: October 24, 2024 Revised: January 11, 2025 Accepted: February 09, 2025

Case of k-LSD pairs:

Step 1: Algorithm 3 begins by leveraging the outputs of Algorithm 1, which defines the lengths of p and q, generates the constraints, and establishes their respective initial domains.

Step 2: Since the first constraint (one's column) involves modular multiplication with only one term, guided by Algorithm 2, all the valid combinations of the LSD (p_n, q_n) and the first a_1(carry propagation) are generated based LSD of N. Note that (p_n, q_n) are known to have specific properties ending in 1, 3, 7, or 9 for primality.

Algorithm 2 serves as a modulo 10 multiplication lookup table. Given the LSD of N, for every match in the modulo table, the associated p_i, q_j, and $quotient[i][j]$ values are captured. The values above the diagonal (upper triangular part) are mirrored in the lower triangular part of the table. Hence, we preserved only the upper triangular part for fast computation. This is because $(i \times j)$ mod 10 is the same as $(j \times i)$ mod 10. In other words, multiplication is commutative, and the modulo operation preserves this property.

Step 3: Generate and test k-LSDs of p and q pair for all feasible combinations of p_k and q_k whose last digits are in (p_n, q_n).

Step 4: Apply Local Constraints for iterative refinement:

- The method selects values for p_k sequentially from its domain and checks all possible combinations in the q_k domain. Note that the domain here is the feasible combinations by step 3. The approach avoids recomputing previously tested combinations for earlier values of p_k.
- The last k digits of N can be derived from $N \; mod \; 10^k$ for increasing values of k to test consistency. This modular result provides a relationship between the LSDs of N and the possible LSDs of its factors. We test factors p_k and q_k such that:
$$(p_k \, . q_k) \; mod \; 10^k = N \; mod \; 10^k$$
- Each valid pair (p_k, q_k) is further tested against the constraints provided by Algorithm 1 to eliminate invalid combinations, significantly reducing the solution space.
- If no valid combinations exist, the algorithm backtracks to the previous step to try a different pair (p_k, q_k)
- Use constraints and previously selected digit pairs at each step to narrow down possible combinations for the current digit.

Step 5: After validating combinations for all digit positions, the algorithm assembles the digits to reconstruct p and q:

- Gather all pairs (p_k, q_k) that satisfy the constraints for each position k.
- If multiple solutions are possible, return all valid combinations.

Algorithm 2: Generating the first LSDs of p and q

```
Input: LSD of $N$
Output: first LSDs of $p$, $q$ and $a_1$ (first carry progation)
1 function First_LSDs_$P_nQ_n$ ($LSD\_N$)
2     Initialize $p_n$ ← [1, 3, 7, 9]  // initial $p_n$ values
3     Initialize $q_n$ ← [1, 3, 7, 9]  // initial $q_n$ values
4     Initialize $p_n\_values$ ← []  // final $p_n$ values
5     Initialize $q_n\_values$ ← []  // final $q_n$ values
6     Initialize $a_1\_values$ ← []  // final  $a_1$ values
7     mod_table ← 2D array of size |$p_n$|x|$q_n$|
8     quotient_table ← 2D array of size |$p_n$|x|$q_n$|
9     for each $i$ in $p_n$ do
```

DOI: https://doi.org/10.54216/JCIM.160104
Received: October 24, 2024 Revised: January 11, 2025 Accepted: February 09, 2025

```
10         for each $\boldsymbol{j}$ in $\boldsymbol{q_n}$ do
11             if $\boldsymbol{i} \geq \boldsymbol{j}$ then
12                 mod_table[$\boldsymbol{i}$][$\boldsymbol{j}$] ← ($\boldsymbol{i}$ * $\boldsymbol{j}$) % 10
13                 quotient_table[$\boldsymbol{i}$][j] ← ($\boldsymbol{i}$ * $\boldsymbol{j}$) // 10
14             else
15                 mod_table[$\boldsymbol{i}$][$\boldsymbol{j}$] ← 0
16                 quotient_table[$\boldsymbol{i}$][$\boldsymbol{j}$] ← 0
17             end if
18         end for
19     end for
20     for each row $\boldsymbol{i}$ in mod_table do
21         for each column $\boldsymbol{j}$ in row do
22             if mod_table[$\boldsymbol{i}$][j] == $\boldsymbol{LSD\_N}$ then
23                 Append $\boldsymbol{p}$[$\boldsymbol{i}$] to $\boldsymbol{p_n\_values}$
24                 Append $\boldsymbol{q}$[$\boldsymbol{j}$] to $\boldsymbol{q_n\_values}$
25                 Append quotient_table[$\boldsymbol{i}$][$\boldsymbol{j}$] to $\boldsymbol{a_1\_values}$
26             end if
27         end for
28     end for
29     return $\boldsymbol{p_n\_values}$, $\boldsymbol{q_n\_values}$, $\boldsymbol{a_1\_values}$
30 end function
```

Algorithm 3: LSDs pair $(\boldsymbol{p_i}, \boldsymbol{q_i})$ estimation

```
Input: RSA modulus $\boldsymbol{N}$, $\boldsymbol{k}$ (number of LSDs to estimate)
Output: $\boldsymbol{p_k}, \boldsymbol{q_k}$ LSDs pairs
1 function $\boldsymbol{k\_LSDsPrimeEstimation(N, k,)}$
     // Initialization using Algorithm 1, 2
2    constraints, domains ← Algorithm 1 ($\boldsymbol{N}$)
3    $(\boldsymbol{p_n}, \boldsymbol{q_n})$_combinations, $\boldsymbol{a_1}$ ← Algorithm 2 ($\boldsymbol{LSD\_N}$) // Get the list of all possible
      $(\boldsymbol{p_n}, \boldsymbol{q_n})$ pairs and $\boldsymbol{a_1}$ (the first carry propagation)
4    k-LSDs_combinations ← [$\boldsymbol{n}$ for $\boldsymbol{n}$ in range $(\boldsymbol{10^k})$ if $\boldsymbol{n}$ mod 10 in
     {$(\boldsymbol{p_n}, \boldsymbol{q_n})$_combinations}]  // GT combinations ending in valid LSD
       // Apply Local Constraints and Refinement based on the BHVO
5    feasible_solutions ← [ ]  // Store final valid combinations
6    for $\boldsymbol{i}$, $\boldsymbol{p}$ in enumerate(k-LSDs_combinations) do
7        for $\boldsymbol{q}$ in k-LSDs_combinations [$\boldsymbol{i}$:] do  // Ensure pairing starts from current index
```

DOI: https://doi.org/10.54216/JCIM.160104
Received: October 24, 2024 Revised: January 11, 2025 Accepted: February 09, 2025

```
8        if $(\boldsymbol{p.q})$ mod $(\mathbf{10^k})$ = $\boldsymbol{N}$ mod $(\mathbf{10^k})$ do  // Test modular consistency
9          // Check for duplicate pairs in the solution space
     if not exists pair in feasible_ solutions where ($\boldsymbol{p}$ = pair[0] and $\boldsymbol{q}$ = pair[1]) or ($\boldsymbol{p}$
   = pair[1] and $\boldsymbol{q}$ = pair[0]) do
11             for each $\boldsymbol{a_1}$ in domains["$\boldsymbol{a_1}$"] do  // Iterate over possible values of the first
               carry
12                 if satisfies_constraints($\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{a_1}$) then
13                    feasible_ solutions.append($\boldsymbol{p}, \boldsymbol{q}$)
14                 else
15                    Backtrack to line 14
16                 end if
17             end for
18          end if
19       end if
20  reconstructed_p ← concatenate_digits(feasible_ solutions, "$\boldsymbol{p}$")  // Reconstruct p
21  reconstructed_q ← concatenate_digits(feasible_ solutions, "$\boldsymbol{q}$")  // Reconstruct q
22  return reconstructed_p, reconstructed_q  // Output reconstructed numbers
23 end function
```

Improving the known bits prime factorisation of $N = pq$ in polynomial time

In this section, we show the primary attack introduced in [20].

Theorem 1. Let $a, b \in \mathbb{Z}^+$ and $m \geq 2$ be an even integer such that $a < b < (2a^m + 1)^{1/m}$. Consider $N = pq = (a^m + r_p)(b^m + r_q)$ is a valid RSA modulus. Let $r_p \equiv p \ (mod\ 2^m)\ and\ r_q \equiv q \ (mod\ 2^m)$ where $r_p < 2a^{m/2}$ and $r_q < 2b^{m/2}$ such that $max\{r_p, r_q\} < 2^k$. If $2^{k-1}(2^{\frac{m}{2}} + 1)$ is sufficiently small and 12 $LSBs$ of p and q are known then N can be factored in polynomial time.

The authors characterize a sufficiently small value as the maximum feasible value of the lowest security level that can be subjected to brute-force attacks using contemporary computing technology. Currently, the lowest security level is established at 112 bits.

Our improved attack

Algorithm 3 pre-computed LSBs of p and q, which are the main requirement of Theorem 1, with computational efficiency as input to the attack proposed by [20]. It overcomes the barriers posed by specific hardware to conduct a side-channel attack to reveal a portion of p and q.

Algorithm 4: Improved Factorisation of $N = pq$

```
Require: initialize $[(\boldsymbol{p_k}, \boldsymbol{q_k})] = \boldsymbol{Algorithm\ 3}$  // where  $[(\boldsymbol{p_k}, \boldsymbol{q_k})]$ is a list of k-LSDs of $\boldsymbol{p}$ and $\boldsymbol{q}$
1  For $\boldsymbol{r_p}, \boldsymbol{r_q}$ in $[(\boldsymbol{p_k}, \boldsymbol{q_k})]$ do  // $\boldsymbol{r_p} \leftarrow \boldsymbol{p_k}, \boldsymbol{r_q} \leftarrow \boldsymbol{q_k}$
2     $\boldsymbol{i} = \lceil (\boldsymbol{r_p r_q})^{1/2} \rceil$
3     $\boldsymbol{j} = \frac{\boldsymbol{r_q}}{\mathbf{2}} + \mathbf{2}^{\frac{\boldsymbol{m}}{\mathbf{2}} - \mathbf{1}} \boldsymbol{r_p} + \mathbf{1}$
4     For $\boldsymbol{i}$ to $\boldsymbol{j}$ do
```

DOI: https://doi.org/10.54216/JCIM.160104
Received: October 24, 2024 Revised: January 11, 2025 Accepted: February 09, 2025

```
5          $\delta = (\lceil\sqrt{N}\rceil - i)^2$
6          $z \equiv N - (r_p r_q) \ (mod\ \delta)$
7          $x_{1,2} \leftarrow root\ of\ x^2 - zx + \delta r_p r_q = 0$
8          If $\left(\frac{x_1}{r_q} + r_p \in \mathbb{Z}^+\right)$ or $\left(\frac{x_2}{r_p} + r_q \in \mathbb{Z}^+\right)$ then
9              Return $p = \frac{x_1}{r_q} + r_p,\ q = \frac{x_2}{r_p} + r_q$
10             Exit // Break inner loop
11         End if
12      End for
13    Exit // Break outer loop
14 End for
```

4. Results and Discussions

Simulations were carried out using Python 3.12.7 in the Jupyter Notebook environment on a local machine for the numerical simulation of the proposed heuristic algorithm based on CSP and an enhanced attack for assessing performance and robustness. The specifications of the machine utilized for the simulation included an AMD Athlon Silver 3050U processor with Radeon Graphics operating at 2.30 GHz and a 64-bit Windows 10 operating system with 20 GB of RAM. To facilitate rapid multiple-precision real and complex arithmetic, the gmpy2 module was employed.

Estimating the k-LSDs of p and q

The first test aimed to analyze the performance of Algorithm 3, which employs the GT and BTVO techniques for estimating k-LSD pairs of p and q. The test concentrated on the total number of feasible (p_k, q_k) LSD pairs, the number of steps needed to find the target pair and how long it would take to complete the search. This measure emphasizes the computational complexity and provides information about the algorithm's convergence to a solution. RSA-2048 bits modulus was utilized in this instance.

N=
303777936847867956132538165969531557029038039462550490562263778911266532621998830006022120
515195460081461072739766830427770638540776883097338470765695773446337634719363743523041287
195275107807588967629210296137809984650307049354235778708843244171705443329481896666555984
680875188962519514964943013668289321494604217634407215333494399925967281040098408258970787
402796210795620457031563314776297318457364727098150982319489607352639067598765581250334795
086167353619069957478276507597049982051339166520415877680068388933590536335414235769417888
81211767696803557896063624995312801941343569522495782450738789819077631126383

Table 1: Results of Algorithm 3 based on the first 5-LSDs of p and q

k	$Number\ of\ (p_k, q_k)\ LSDs$ pair n	Number of iterations to find the exact pair	$Total\ Time\ taken$
1	2	1	0 ns
2	16	11	0 ns
3	160	23	15.6 ms
4	1621	403	1.28 s
5	16206	6987	2 min 20s

DOI: https://doi.org/10.54216/JCIM.160104
Received: October 24, 2024 Revised: January 11, 2025 Accepted: February 09, 2025

As indicated in Table 1, conducting a linear search within a list of size (p_k, q_k) LSDs pair, where n denotes the size, results in a worst-case complexity of $O(n)$. Given that the optimal pair (p_k, q_k) can be positioned randomly, the anticipated number of iterations required to identify the optimal pair is $O(n/2)$ in the average scenario. However, with an increase in the value of k, both the time needed to generate and validate n and the time to find the target pair, rise linearly. The latest guidance from NIST [22] on key management specifies that a minimum-security level is 112 bits. This suggests that the maximum practical value of n that can be subjected to brute force by contemporary computing machines is 2^{112}. Consequently, any value of n below 2^{112} is negligible. Hence, it takes polynomial time to brute-force n to discover the optimal (p_k, q_k) LSD pair at any position of k, as illustrated in Table 1.

Known Bits Prime Factorization Attack Based on the 12-LSBs of p and q

This section replicates the known bits prime factorization attack outlined in Algorithm 4, which utilizes prior knowledge of the 4-LSDs of p and q. The experiments were conducted with balanced RSA primes, each having a bit length of 1024, thus forming an RSA-2048 modulus as follows;

N=
254432134848033306765466360605067672713192119562738803513743518254625615801325511773983650
045673026490293724691085285813831823660328796126064275138262348021411229982061934595317738
337964801727892542334700845922311179460436678038166743671495233267311270087333553618242507
436617332719512700416039949918552601931006443393514094460366015740466980367515605709366458
027738329608044170750026717443548158411552466678315129569489611803135375760808108789041284
576974946332649978083818108441170169597124938473832333003773478189908742844727615199026762
54694772586325941589525740707826852095908188649384624121217162949627607660163

We initiated the process by estimating the 4-LSDs (equivalent to the 12-LSBs) of p and q, utilizing N as required by Theorem 1 to facilitate the factorization of N through Algorithm 3. In our focus on the 4-LSDs of p and q, the RSA modulus N and the numeral four (4) are provided as inputs to Algorithm 1, ensuring that the generated constraints were up to four (4). The LSD of N, which is 3, was subsequently fed into Algorithm 2 to determine the last digit positions of of p (p_4), q (q_4) and the initial partial product carry, a_1. The results were $p_4 = [1,7]$, $q_4 = [3,9]$, and $a_1 = [0,6]$. The GT method was applied based on the values obtained for p_4 and q_4 (Algorithm 3, line 5), yielding the first set of feasible solutions. BHVO then undertook a local constraint for iterative refinement to filter these feasible solutions (Algorithm 3, lines 13-20). Ultimately, we identified 1623 optimal solutions, as shown in Table 2.

Table 2: Results of 4-LSDs (p, q) pairs of N

k	$Number\ of\ (p_k, q_k)\ LSDs$ pair	Total time taken (ms)
4	1623	953

Generating optimal combinations of 4-LSDs for p and q was finished in 953 milliseconds (refer to Table 2). This attainment is notable regarding the complexity of the task, which concerned listing and confirming all potential combinations of two major prime factors. This analysis is efficient enough to allow subsequent cryptographic analyses without significant delays in pre-processing for operations requiring real-time computations, such as predicting key structures or assessing factorization limitations. Early pruning of the algorithm, which uses a pre-computed list of 4-LSD primes, results in fewer potential pairs for analysis. Moreover, pairs that do not meet the necessary modular constraints for forming valid RSA moduli are eliminated at the outset.

We now turn to Algorithm 4, which simulates the factorization of N in polynomial time utilizing the 4-LSDs of p and q, as shown in Table 2, by Theorem 1. By pre-computing the 4-LSDs of p and q the factorization process (Algorithm 4) benefits from a refined search space, leading to decreased computational demands and minimizing dependence on assumptions that might not hold in actual attack scenarios. This pre-computation phase enhances overall efficiency by filtering out non-viable candidates, enabling the factorization algorithm to concentrate on pairs with a higher probability of success.

DOI: https://doi.org/10.54216/JCIM.160104
Received: October 24, 2024 Revised: January 11, 2025 Accepted: February 09, 2025

Each pair of 4-LSDs generated is systematically inputted into Algorithm 4, denoted as r_p and r_q. Algorithm 4, then iteratively assessed these pairs to pinpoint the precise combination that fulfils the condition outlined in line 8 of Algorithm 4. The optimal pairs (3017, 2539) yielded the prime factors of N, as shown in Table 3.

Table 3: Factorization of RSA-2048 known bits prime

Exact 4-LSDs of p and q	(3017, 2539)
Number of iterations	527
P value	207632566695348090325106198564354306872362493463538154841386 314580707220972445801440409737589803024013035554181699335224 061662229162879643933792870833231736875142501533422110427899 095351781206012327937258761409973123340262144886588093314114 536052456895922041585909651666335476791456709509341751911472 100003017
Q value	122539608741316849829261726098688957114502463272691906657106 165887494465656483627796660671278978213477051915433597161268 345944097932917669169852614268434890176706523882967335716979 529907163623313323845921267400475000574500531377847942396759 927437400904034577111052905698000623411296101838403579267392 100002539
Total Time taken (s)	2.61

The results were analysed and understood based on a practical application, as demonstrated in Table 3. Algorithm 4, which employs the GT and BHVO techniques, shows significant efficiency, especially when the parameter k (indicating the number of constraints) is minimal. This efficiency is attributed to its capacity to prioritize choices that eliminate infeasible solutions early in the computation process. These techniques ensure that the computational steps needed to meet constraints and perform factorization remain within polynomial limits (2.61 seconds). This guarantees feasibility and provides predictability in runtime, regardless of variations in input constraints. This method is a reliable process within polynomial time for a known bit's factorization process, which is a significant benefit.

Scope of the attack and countermeasures

The prime numbers examined in this paper focus on the RSA modulus, denoted as N, and its prime factors, p and q, which meet the specified condition:

$$\left\lceil \sqrt{N} - \left\lfloor \sqrt{p} \right\rfloor . \left\lfloor \sqrt{q} \right\rfloor \right\rceil < 2^{112}$$

Though this study's applied known bits prime factorization method focuses on a particular subset of prime numbers, there is no effective detection technique to mitigate the threat. Therefore, avoiding these primes in the RSA key generation process is crucial and requires a comprehensive awareness of the secret parameters p and q. If the RSA modulus N and its prime factors p and q fulfil the outlined condition, the key generator must identify new values for p and q; otherwise, N remains vulnerable to the discussed attack.

Future scope of the study

Algorithm 3, based on the GT and BHVO, exhibits effectiveness for smaller values of k (specifically, values below 6). Its reliance on sequential exploration may create difficulties as k grows considerably. Further studies should utilise performance enhancement techniques like advanced heuristics to improve the algorithm's resilience for larger k. Additional testing with higher-order LSDs is essential to validate the method's scalability, as the computational complexity may escalate exponentially with greater precision.

The formulated problem is grounded in CSP. As previously mentioned, a well-designed solver capable of evaluating variables in polynomial time (especially for larger k) can also retrieve the prime factors of the RSA

DOI: https://doi.org/10.54216/JCIM.160104
Received: October 24, 2024 Revised: January 11, 2025 Accepted: February 09, 2025

modulus N. Future research should concentrate on developing CSP solvers tailored explicitly to our CSP formulation. Given that deep learning has shown promising outcomes in CSP-related tasks [23], particularly with reinforcement learning [24] and transformer-based architectures [25], these models should inform the design of such solvers to optimize efficiency.

We assessed our proposed method using known 4-LSDs of known primes (12 bits of p and q). Further investigations should apply the proposed method to factorization attacks involving higher known LSBs of primes and attacks based on random knowledge of p and q.

5. Conclusion

This study introduced a new heuristic algorithm grounded in the CSP to estimate k-LSDs of the RSA prime factors, p and q, towards known bits factorization attacks. Based on the partial knowledge of the RSA prime factors, our improved attack does not require specialized hardware for side-channel attacks. The proposed algorithm was assessed through the known LSBs attack on specifically structured RSA primes outlined by Abd GHafar et al. [20], achieving successful factorization of a 2048-bit modulus N in polynomial time. Additionally, the study discussed countermeasures to protect RSA users from such attacks. The method presented lays a foundation for future investigations into prime factorization methods involving known bits and cryptanalysis. In future studies, integrating machine learning models such as reinforcement learning and transformer-based architectures may improve the selection and refinement of candidate pairs p and q. Given that the heuristic approach relies on the CSP, the development of intelligent solvers is of considerable significance in facilitating prime factorization.

Funding: "This research received no external funding"

Conflicts of Interest: "The authors declare no conflict of interest."

References

[1] R. Ranasinghe, M. Chathurangi, and P. Athukorala, "A novel improvement in RSA algorithm," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 27, no. 1, pp. 143–150, 2024, doi: 10.47974/JDMSC-1628.

[2] P. Cotan and G. Teşeleanu, "A security analysis of two classes of RSA-like cryptosystems," *Journal of Mathematical Cryptology*, vol. 18, no. 1, Jan. 2024, doi: 10.1515/jmc-2024-0013.

[3] D. Ramakrishna and M. A. Shaik, "A comprehensive analysis of cryptographic algorithms: Evaluating security, efficiency, and future challenges," *IEEE Access*, pp. 1–1, 2024, doi: 10.1109/ACCESS.2024.3518533.

[4] E. A. Adeniyi, P. B. Falola, M. S. Maashi, M. Aljebreen, and S. Bharany, "Secure sensitive data sharing using RSA and ElGamal cryptographic algorithms with hash functions," *Information (Switzerland)*, vol. 13, no. 10, Oct. 2022, doi: 10.3390/info13100442.

[5] A. Overmars and S. Venkatraman, "Continued fractions applied to the one line factoring algorithm for breaking RSA," *Journal of Cybersecurity and Privacy*, vol. 4, no. 1, pp. 41–54, Mar. 2024, doi: 10.3390/jcp4010003.

[6] S. Pochu, S. Rama, and K. Nersu, "Cybersecurity in the era of quantum computing: Challenges and solutions," 2022.

[7] A. Montina and S. Wolf, "An algebraic-geometry approach to prime factorization," Sep. 2022, [Online]. Available: http://arxiv.org/abs/2209.11650

[8] M. Zheng, Z. Chen, and Y. Wu, "Solving generalized bivariate integer equations and its application to factoring with known bits," *IEEE Access*, vol. 11, pp. 34674–34684, 2023, doi: 10.1109/ACCESS.2023.3264590.

[9] B. Harjito, H. N. Tyas, E. Suryani, and D. W. Wardani, "Comparative analysis of RSA and NTRU algorithms and implementation in the cloud," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 3, p. 2022, 2022, doi: 10.14569/IJACSA.2022.0130321.

DOI: https://doi.org/10.54216/JCIM.160104
Received: October 24, 2024 Revised: January 11, 2025 Accepted: February 09, 2025

[10] Y. Yarom, D. Genkin, and N. Heninger, "CacheBleed: A timing attack on OpenSSL constant-time RSA," *J. Cryptogr. Eng.*, vol. 7, no. 2, pp. 99–112, Jun. 2017, doi: 10.1007/s13389-017-0152-y.

[11] P. Singh, P. Pranav, and S. Dutta, "Optimizing cryptographic protocols against side-channel attacks using WGAN-GP and genetic algorithms," *Sci. Rep.*, vol. 15, no. 1, p. 2130, Jan. 2025, doi: 10.1038/s41598-025-86118-4.

[12] C. Luo, Y. Fei, and D. Kaeli, "Side-channel timing attack of RSA on a GPU," *ACM Trans. Archit. Code Optim.*, vol. 16, no. 3, Aug. 2019, doi: 10.1145/3341729.

[13] H. Ferradi, R. Géraud, S. Guilley, D. Naccache, and M. Tibouchi, "Recovering secrets from prefix-dependent leakage," *Journal of Mathematical Cryptology*, vol. 14, no. 1, pp. 15–24, Jan. 2020, doi: 10.1515/jmc-2015-0048.

[14] A. S. Chandran, "Review on cryptography and network security zero knowledge technique in blockchain technology," *International Journal of Information Security and Privacy*, vol. 16, no. 2, 2022, doi: 10.4018/IJISP.308306.

[15] R. L. Rivest and A. Shamir, "Efficient factoring based on partial information," in *Advances in Cryptology—EUROCRYPT' 85*, vol. 219, Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 31–34, doi: 10.1007/3-540-39805-8_3.

[16] D. Coppersmith, "Finding a small root of a bivariate integer equation; factoring with high bits known," vol. 1070, Springer, Berlin, Heidelberg, 1996, pp. 178–189, doi: 10.1007/3-540-68339-9_16.

[17] M. Herrmann and A. May, "Solving linear equations modulo divisors: On factoring given any bits," Springer, Berlin, Heidelberg, 2008, pp. 406–424, doi: 10.1007/978-3-540-89255-7_25.

[18] N. Heninger and H. Shacham, "Reconstructing RSA private keys from random key bits," Springer: Berlin/Heidelberg, Germany, 2009, pp. 1–17, doi: 10.1007/978-3-642-03356-8_1.

[19] S. Maitra, S. Sarkar, and S. Sen Gupta, "Factoring RSA modulus using prime reconstruction from random known bits," *Progress in Cryptology—AFRICACRYPT 2010*, pp. 82–99, 2010, doi: 10.1007/978-3-642-12678-9_6.

[20] A. H. Abd Ghafar, M. R. Kamel Ariffin, and M. A. Asbullah, "A new LSB attack on special-structured RSA primes," *Symmetry (Basel)*, vol. 12, no. 5, May 2020, doi: 10.3390/SYM12050838.

[21] A. H. Abd Ghafar, M. R. Kamel Ariffin, and M. A. Asbullah, "A new LSB attack on special-structured RSA primes," *Symmetry (Basel)*, vol. 12, no. 5, May 2020, doi: 10.3390/SYM12050838.

[22] E. Barker, "Recommendation for key management part 1: General," Gaithersburg, MD, Jan. 2016, doi: 10.6028/NIST.SP.800-57pt1r4.

[23] J. Zhang et al., "A survey for solving mixed integer programming via machine learning," *Neurocomputing*, vol. 519, pp. 205–217, Jan. 2023, doi: 10.1016/j.neucom.2022.11.024.

[24] Y. Xu, D. Stern, and H. Samulowitz, "Learning adaptation to solve constraint satisfaction problems," in *Proceedings of Learning and Intelligent Optimization (LION)*, p. 14, 2009.

[25] Z. Yang, A. Ishay, and J. Lee, "Learning to solve constraint satisfaction problems with recurrent transformer," Jul. 2023, [Online]. Available: http://arxiv.org/abs/2307.04895

DOI: https://doi.org/10.54216/JCIM.160104
Received: October 24, 2024 Revised: January 11, 2025 Accepted: February 09, 2025