Cryptanalysis of Symmetric Encryption Algorithms in a Polynomial Residue Number System

By:
Venkata Sai Sidhartha Ratnakaram (21CSB0F01)
Rahul Patel (21CSB0F13)
Anshu Kumar (21CSB0A05)

National Institute of Technology Warangal

Guide: Prof. R Padmavathy

Contents

Problem Statement

Goal: Analyze and assess the security of the symmetric encryption algorithm based on the Polynomial Residue Number System (PRNS) by I. Yakymenko, M. Karpinski, R. Shevchuk, M. Kasianchuk published on Hindawi, Journal of Applied Mathematics.

- **Identify Weaknesses:** Examine key selection, polynomial reconstruction (*Chinese Remainder Theorem - CRT*), and potential structural vulnerabilities.
- **Explore Cryptanalysis Techniques:** Test feasibility of *brute-force attacks, algebraic attacks*, and *side-channel vulnerabilities*.
- **Develop Cryptanalytic Methods:** Construct potential *attack strategies* to test encryption resilience.
- **Summarize Findings:** Provide *conclusions and recommendations* for improving security or modifying the encryption scheme.

Theoretical Foundations of Polynomial RNS.

- In PRNS, we represent polynomials using modular arithmetic over a set of moduli polynomials (or base polynomials)
- An arbitrary polynomial $N(x)$ in the RNS is represented as the residual $b_i(x)$ from dividing $N(x)$ by each of the pairwise mutually coprime modulo polynomials $p_i(x)$:

$$b_i(x) = N(x) \mod p_i(x) \tag{1}$$

- The recovery of the polynomial $N(x)$ is usually based on the Chinese Remainder Theorem in the ring of polynomials $\mathbb{Z}[x]$:

$$N(x) = \left(\sum_{i=1}^{s} b_i(x) M_i(x) m_i(x) \right) \mod P(x) \tag{2}$$

Theoretical Foundations of Polynomial RNS.

- Where $P(x) = \prod_{i=1}^{s} p_i(x)$, $M_i(x) = \frac{P(x)}{p_i(x)}$, and $m_i(x) = M_i^{-1}(x) \mod p_i(x)$
- s is the number of modules
- For polynomial powers, the inequality $\deg(N_x) < \deg(P_x)$ must be satisfied.

Development of Polynomial Symmetric Encryption Methods in the RNS.

- When recovering a polynomial from its residuals in the sum (2), the multiplication is not by the parameters $m_i(x) = M_i^{-1} \mod p_i(x)$, but by arbitrarily chosen polynomials $k_i(x)$.
- Where $k_i(x)$ is mutually prime with $p_i(x)$.
- Both subscribers choose module systems $p_i(x)$ and $k_i(x)$ only known to them for which the following conditions are met:

$$\deg(N_x) < \deg(P_x) \quad \text{and} \quad \gcd(k_i(x), p_i(x)) = 1$$

- Accordingly both sender and receiver know know the parameters $M_i(x)$ and $m_i(x)$.

Development of Polynomial Symmetric Encryption Methods in the RNS.

- Encryption occurs when the number is restored to the positional number system according to the following expression:

$$N'(x) = \left(\sum_{i=1}^{s} b_i(x) M_i(x) k_i(x) \right) \mod P(x) \tag{3}$$

The polynomial is ciphertext which is transferred.

- When decrypting, following values are calculated first:

$$q_i(x) = (m_i(x) \cdot (k_i^{-1}(x) \mod p_i(x))) \mod p_i(x) \tag{4}$$

$$b'_i(x) = N'(x) \mod p_i(x)$$

- To obtain the true residuals $b_i(x)$, we need to perform following:

$$b_i(x) = (b'_i(x) q_i(x)) \mod p_i(x)$$

$$= (b'_i(x) m_i(x) \cdot (k_i^{-1}(x) \mod p_i(x))) \mod p_i(x)$$

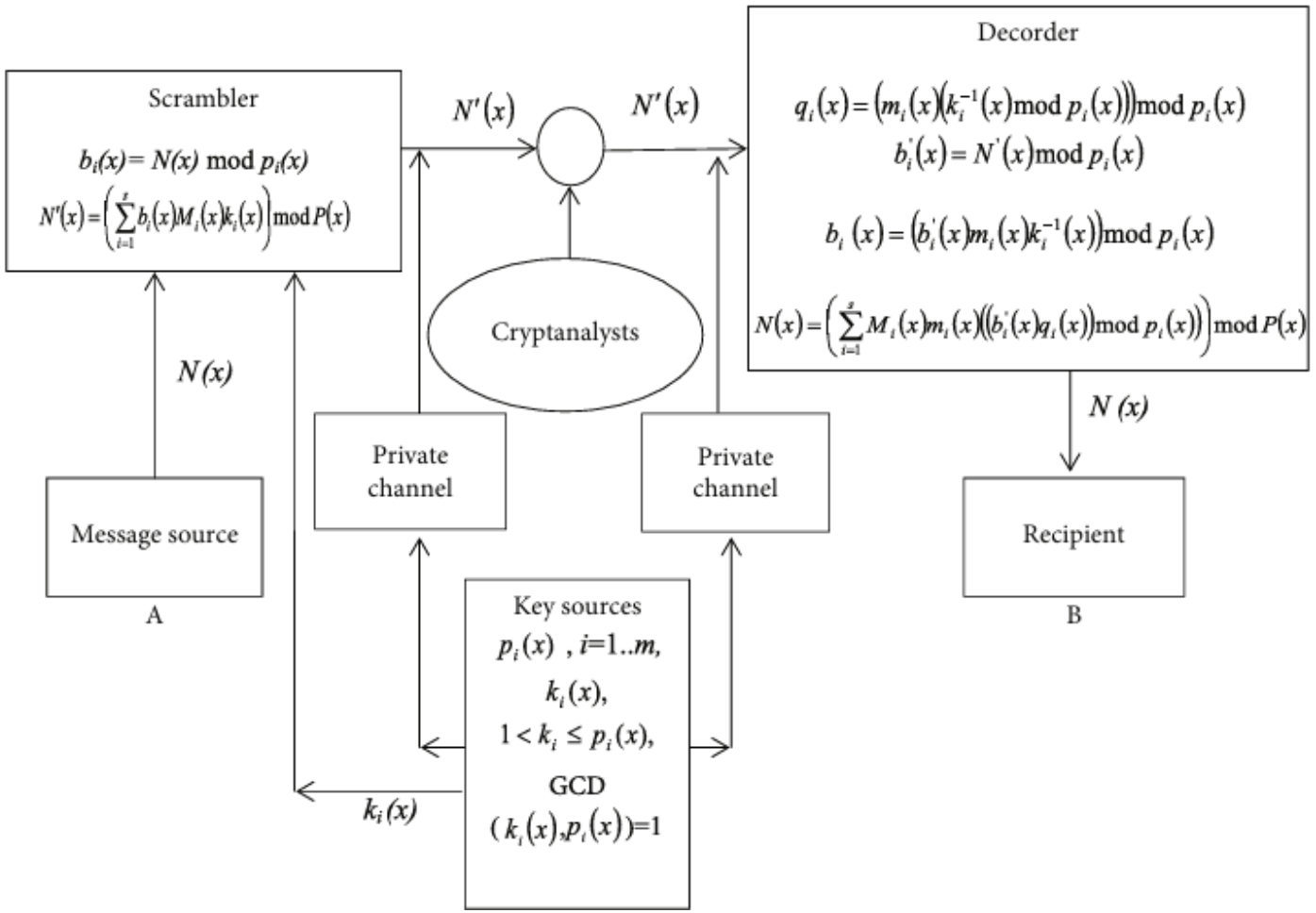
Development of Polynomial Symmetric Encryption Methods in the RNS.

- The recovery of the plain text polynomial N(x) is carried out according to Formula(2) or the expression that follows from it can be used:

$$N(x) = \left(\sum_{i=1}^{s} M_i(x) m_i(x) (b'_i(x) m_i(x) \cdot k_i^{-1}(x) \mod p_i(x)) \right) \cdot \mod p_i(x) \mod P(x)$$

$$N(x) = \left(\sum_{i=1}^{s} M_i(x) m_i(x) \left(b'_i(x) q_i(x) \right) \mod p_i(x) \right) \mod P(x) \tag{5}$$

Development of Polynomial Symmetric Encryption Methods in the RNS.



[1]Scheme of the proposed polynomial symmetric encryption in RNS

Cryptanalysis of Case-1: Overview

- **Adversary Goal**: Recover the secret irreducible polynomial $p(x)$ of degree n over $\mathbb{GF}(p)$.
- **Adversary Knowledge**:
  - Plaintext polynomial $N(x)$.
  - Residuals $b(x) = N(x) \mod p(x)$.
  - Computes $d(x) = N(x) - b(x)$, where $p(x) \mid d(x)$.
- **Key Insight**: Factorization of $d(x)$ leaks candidates for $p(x)$.
- **Security Assumptions**:
  - Irreducibility of $p(x)$ over $\mathbb{GF}(p)$.

Cryptanalysis of Case-1: Attack Strategy

- **Theoretical Cryptanalysis**:
  1. Compute $d(x) = N(x) - b(x)$ (complexity $O(n)$).
  2. Factorize $d(x)$ using Cantor-Zassenhaus algorithm (complexity $O(n^3)$).
  3. Filter irreducible factors of degree n.
- **Success Probability**:

$$P_{\text{success}} = \frac{1}{k} \quad \text{if } d(x) \text{ has } k \text{ irreducible factors of degree } n.$$

- **Search Space**:
  - Bounded by the number of irreducible polynomials:

$$I(p, n) = \frac{1}{n} \sum_{d|n} \mu(d)\, p^{n/d}.$$

  - Even for For large p or n, the search space for the irreducible factors is slighty reduced to $\frac{p^n}{n}$
  - **Risk**: Small p or n reduces adversarial effort.

Cantor-Zassenhaus algorithm

Input: A polynomial $f(x) \in F[x]$, where F is a finite field (like $\mathbb{F}_p$ where p is a prime number).
Output: Irreducible factors of $f(x)$.

1. **Compute gcd**:
   - Choose a random polynomial $g(x) \in \mathbb{F}[x]$ of the same degree as $f(x)$.
   - Compute the GCD of $f(x)$ and $g(x)$, denoted $\gcd(f(x), g(x))$.
     - If the GCD is a non-trivial factor (i.e., a proper divisor of $f(x)$), then this is a potential irreducible factor of $f(x)$.
     - If the GCD is 1, repeat the process with a different randomly chosen polynomial $g(x)$.
   - For each non-trivial factor found:
     - Factor it out from $f(x)$
     - Continue the factorization process on the remaining polynomial
   - Repeat until $f(x)$ is completely reduced to irreducible factors.

Berlekamp Algorithm

Berlekamp Algorithm for Polynomial Factorization over GF(p)
Input: A square-free polynomial $f(x) \in \mathrm{GF}(p)[x]$.
Output: Irreducible factors of $f(x)$.

1. **Ensure Square-Free**: Compute $\gcd(f(x), f'(x))$. If the gcd is non-trivial, divide $f(x)$ by the gcd to remove repeated factors.
2. **Build the Berlekamp Matrix** Q: Construct a $n \times n$ matrix Q (where $n = \deg f(x)$) such that:

$$Q_{i,j} = \text{coefficient of } x^j \text{ in } x^{ip} \mod f(x), \quad 0 \leq i, j < n$$

Here, p is the field characteristic.

Berlekamp Algorithm

1. **Find Null Space Vectors**: Solve the linear system $(Q - I)\mathbf{v} = \mathbf{0}$, where I is the identity matrix. Each non-trivial solution $\mathbf{v}$ corresponds to a polynomial $g(x)$.
2. **Factor Using Solutions**: For each solution $g(x)$, compute:

$$\gcd(f(x), g(x) - c) \quad \forall c \in \mathrm{GF}(p)$$

   This splits $f(x)$ into non-trivial factors.
3. **Recurse**: Apply the algorithm recursively to each factor until all factors are irreducible.

Case 2: Cryptanalysis Overview

Goal: Factor $P(x)$, recover secret parameters $p_i(x)$ and $k_i(x)$

- **Factor** $P(x)$:
  - Use polynomial factorization over $\mathbb{GF}(p)$:

$$P(x) = \prod_{i=1}^{s} p_i(x)$$

  - Algorithms: Berlekamp or Cantor-Zassenhaus (complexity $O(n^3)$).
- **Compute** $M_i(x)$:

$$M_i(x) = \frac{P(x)}{p_i(x)} \quad \text{(for each recovered } p_i(x)\text{)}$$

- **Recover** $m_i(x)$:
  - Calculate modular inverse via Extended Euclidean Algorithm:

$$m_i(x) \equiv M_i^{-1}(x) \mod p_i(x)$$

Case 2: Recovering $k_i(x)$

- **Solve for** $k_i(x)$ using known plaintext-ciphertext pairs $(N_j(x), N'_j(x))$:
  1. Compute residuals:
  $$b_{i,j}(x) = N_j(x) \mod p_i(x), \quad b'_{i,j}(x) = N'_j(x) \mod p_i(x)$$
  2. Derive $k_i(x)$:
  $$k_i(x) \equiv \left(b'_{i,j}(x) \cdot m_i(x) \cdot b^{-1}_{i,j}(x)\right) \mod p_i(x)$$
- **Success Conditions**:
  - Requires at least one known plaintext-ciphertext pair.
  - Unique solution if $b_{i,j}(x)$ is invertible modulo $p_i(x)$.
- **Complexity**:
  - Modular inverses: $O(n^2)$ per $p_i(x)$.
  - Solving for $k_i(x)$: $O(s \cdot n^2)$ for s factors.

Case-3: Coppersmith attack overview

Goal:Some information about the plaintext polynomial $N(x)$ is leaked. The goal is to recover $N(x)$ efficiently.

- **Attacker's knowledge:** Moduli $p_i(x)$, Part of the plaintext and the residuals $b_i(x)$.
- **Equation formation:**
  - We assume that $N(x)$ is the sum of a known part $g(x)$ and an unknown small part $r(x)$:

$$N(x) = g(x) + r(x)$$

  - The cryptosystem provides modular reductions of $N(x)$ with respect to known polynomials $p_1(x)$ and $p_2(x)$:

$$N(x) \equiv b_1(x) \mod p_1(x)$$

$$N(x) \equiv b_2(x) \mod p_2(x)$$

Case-3: Coppersmith attack strategy

- Since we know $N(x) = g(x) + r(x)$, we can substitute this into the above equations:

$$g(x) + r(x) \equiv b_1(x) \mod p_1(x)$$

$$g(x) + r(x) \equiv b_2(x) \mod p_2(x)$$

- Rearranging for $r(x)$:

$$r(x) \equiv b_1(x) - g(x) \mod p_1(x)$$

$$r(x) \equiv b_2(x) - g(x) \mod p_2(x)$$

- Since r(x) is small, we can find it using lattice basis reduction techniques like the LLL algorithm.

Case 4: Recovering Missing Plaintext Coefficients

Attacker's Goal:

- Recover the missing coefficients of the plaintext polynomial $N(x)$.

Attacker's Knowledge:

- **The Encrypted Polynomial** $N'(x)$:

$$N'(x) = -64x^6 - 250x^5 - 360x^4 - 545x^3 - 492x^2 - 403x - 172$$

- **The Modulus Polynomials** $p_i(x)$:

$$p_1(x) = x^2 + x + 1, \quad p_2(x) = x^3 + x + 1, \quad p_3(x) = x^2 + 1$$

- **Partial Information on Plaintext** $N(x)$: Some coefficients a_0, a_1, a_2, a_3 are known.

Case 4: Recovering Missing Plaintext Coefficients(Continued)

Attack Strategy:

1. Compute residuals using polynomial division:

$$b'_{i,j}(x) = N'_j(x) \mod p_i(x)$$

2. Construct a lattice representation using the modulus polynomials.
3. Apply the LLL algorithm to find a short vector, approximating $N(x)$.

How the Attack Works:

- Polynomial division isolates useful parts of the ciphertext.
- Lattice reduction (LLL algorithm) finds the hidden structure of $N(x)$.
- The recovered polynomial provides an approximation of the missing coefficients.

Complexity:

- Polynomial division: $O(n^2)$.
- Lattice reduction (LLL): $O(n^3)$.

Lenstra–Lenstra–Lovász lattice basis reduction algorithm

- Given a basis $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_n$.
- Compute the Gram-Schmidt basis $\mathbf{b}_0^*, \mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$.
- Set index $k = 1$.
- For j from $k-1$ down to 0:
- Compute Gram-Schmidt coefficient:

$$\mu_{k,j} = \frac{\mathbf{b}_k \cdot \mathbf{b}_j^*}{\mathbf{b}_j^* \cdot \mathbf{b}_j^*} \tag{1}$$

- If $|\mu_{k,j}| > 0.5$, update $\mathbf{b}_k$:

$$\mathbf{b}_k = \mathbf{b}_k - \lfloor \mu_{k,j} \rceil \mathbf{b}_j \tag{2}$$

Lenstra–Lenstra–Lovász lattice basis reduction algorithm

- Continue checking size reduction conditions.
- Check if the Lovász condition holds:

$$\delta \cdot \|\mathbf{b}_{k-1}^*\|^2 \leq \|\mathbf{b}_k^* + \mu_{k,k-1}\mathbf{b}_{k-1}^*\|^2 \tag{3}$$

- If true, increment k.
- Otherwise:
  - Swap $\mathbf{b}_k$ and $\mathbf{b}_{k-1}$.
  - Update Gram-Schmidt coefficients.
  - Set $k = \max(k-1, 1)$ and repeat.
- The algorithm stops when $k > n$.
- The output is a reduced lattice basis.
- This basis is nearly orthogonal and more useful for applications.

Recovering Plaintext Polynomial Coefficients Using Gröbner Bases

- The goal is to recover the unknown coefficients a_6, a_5, a_4 of the plaintext polynomial $N(x)$.
- Known values:
  - Known partial plaintext coefficients
  - Known ciphertext polynomial $N'(x)$.
  - Modulus polynomials $p_1(x), p_2(x), p_3(x)$
- The ciphertext is given by:

$$N'(x) = N(x) \cdot k_i(x) + c_i(x) \cdot p_i(x)$$

where $k_i(x)$ are the secret key polynomials, and $p_i(x)$ are modulus polynomials and the $c_i(x)$ is the quotient polynomial.

Recovering Plaintext Polynomial Coefficients Using Gröbner Bases

- Steps:
  1. Define the system of equations using the relation

$$N'(x) = N(x) \cdot k_i(x) + c_i(x) \cdot p_i(x)$$

  2. Substitute known values for plaintext polynomial into the equations.
  3. Use **Gröbner basis** to solve for the unknown remaining coefficients
- The Gröbner basis is computed using Buchberger's algorithm, which reduces the system of equations to a simpler form.
- The unknowns are then solved from the reduced system, yielding the recovered coefficients for the plaintext polynomial $N(x)$.

Buchberger's Algorithm for Computing Gröbner Bases

Overview of Buchberger's Algorithm:

- Buchberger's Algorithm computes the Gröbner basis for a given ideal in a polynomial ring.
- The algorithm refines a set of polynomials iteratively until it forms a Gröbner basis.
- A Gröbner basis has special properties that make solving systems of polynomial equations easier.

Key Concepts:

- **Ideal:** A set of polynomials closed under addition and multiplication by any polynomial in the ring.
- **Gröbner Basis:** A special basis for the ideal, where the polynomials have standard properties (e.g., leading terms).

Buchberger's Algorithm for Computing Gröbner Bases

Steps in Buchberger's Algorithm:

- **Start with a set of polynomials:**
  - Let $S = \{f_1, f_2, \ldots, f_m\}$.
- **Compute S-polynomials:**

$$S(f_i, f_j) = \frac{\text{lcm}(lt(f_i), lt(f_j))}{lt(f_i)} f_i - \frac{\text{lcm}(lt(f_i), lt(f_j))}{lt(f_j)} f_j$$

- lcm denotes the least common multiple of the leading terms $lt(f_i)$ and $lt(f_j)$.
- **Reduce S-polynomials:**
  - Reduce $S(f_i, f_j)$ by subtracting multiples of polynomials in S until it has no leading term in common.

Buchberger's Algorithm for Computing Gröbner Bases

- **Add the remainder to the set:**
  - If the remainder is non-zero, add it to S.
- **Repeat the process:** Continue until no new polynomials can be added.
- **Stop when the set stabilizes:** The set S is now a Gröbner basis.

Key Properties of Gröbner Bases:

- **Uniqueness:** The Gröbner basis is unique for a given monomial order.
- **Solvability:** Gröbner bases allow efficient solving of polynomial systems.
- **Normalization:** Every ideal has a Gröbner basis, which simplifies manipulation and solving.

Case 6: Recovering plaintext polynomial using Lagrange Interpolation

Attack Overview:

- The goal is to recover the original plaintext polynomial $N(x)$ from an encrypted polynomial.
- The encryption masks the coefficients using a secret key polynomial.
- The attack exploits Lagrange Interpolation to reconstruct $N(x)$.

Attack Knows:

- The ciphertext or the encrypted polynomial representing ciphertext
- The correction polynomial $q_i(x)$
- The moduli polynomial $p_i(x)$

Case 6: Recovering plaintext polynomial using Lagrange Interpolation

Steps of the Attack:

- **Compute the encrypted residues**:

$$b_i'(x) = \text{Ciphertext Polynomial} \mod p_i(x)$$

- **Undo the masking to recover true residues**:
  - Compute true residues:

$$b_i(x) = b_i'(x) \cdot q_i(x) \mod p_i(x)$$

- **Construct the Lagrange Basis Polynomials:**
  -

$$L_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{3} \frac{x - p_j(x)}{p_i(x) - p_j(x)} \tag{4}$$

  - The value is equal to 1 at $p_i(x)$ and 0 for other which is similar to $M_i(x) \cdot m_i(x)$
  - $L_i(x)$ evaluates to 0 at all other $p_j(x)$, eliminating unwanted terms in interpolation.

Case 6: Recovering plaintext polynomial using Lagrange Interpolation

- **Reconstruct $N(x)$ using Lagrange Interpolation**:

$$N(x) = \sum_{i=1}^{3} b_i(x) L_i(x) \tag{5}$$

Results

- The algorithm was implemented in Python 3.12 for its efficiency in mathematical computations, using the SymPy library for symbolic mathematics.
- SymPy allows exact algebraic manipulation, crucial for cryptographic attacks involving polynomials and modular arithmetic.
- It ensures precision without rounding errors, unlike floating-point libraries like NumPy, making it suitable for cryptographic applications.
- The symbolic output is human-readable, aiding in debugging and understanding intermediate steps.
- The implementation was done on a system with an 11th Gen Intel Core i5-1135G7 processor and 8 GB of RAM.

Comparison

Performance comparison of cryptanalytic cases (time in microseconds)

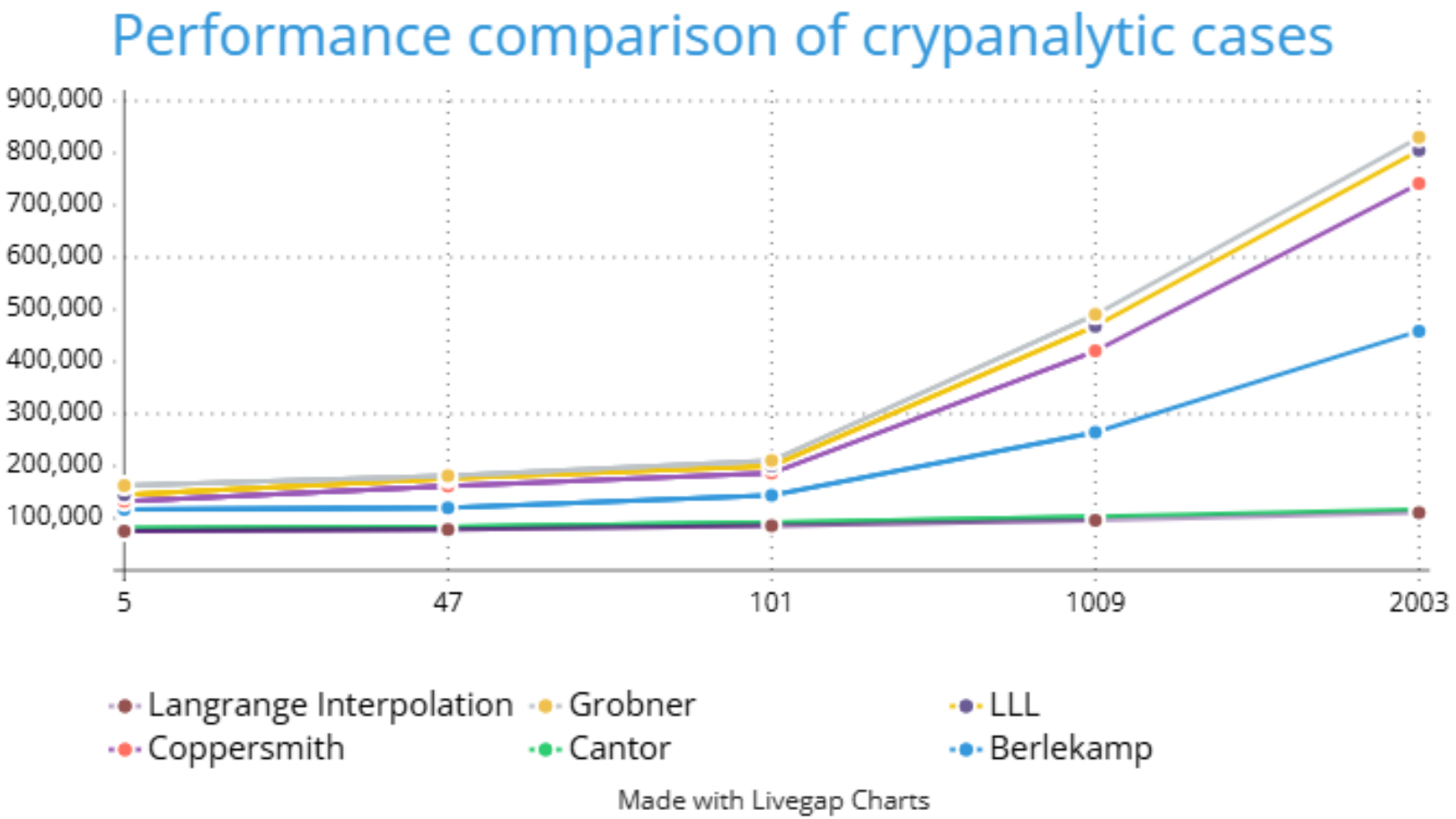
Method	GF(5)	GF(47)	GF(101)	GF(1009)	GF(2003)
Berlekamp (P)	117,185.35	119,898.09	144,148.12	264,570.00	458,820.00
Cantor-Zassenhaus	80,101.00	83,231.00	91,910.00	103,020.00	116,700.00
Coppersmith	132,021.00	161,047.00	186,090.00	420,327.43	741,361.00
LLL (Lattice)	145,301.00	1,706,003.32	200,000.00	468,200.50	805,006.70
Gröbner Basis	162,037.23	181,010.00	210,067.00	490,209.00	830,050.62
Lagrange Interpolation	75,210.00	78,109.00	85,257.00	96,031.00	110,240.11

- The table presents execution times (in microseconds) for six cryptanalytic methods across various Galois fields F_p with prime sizes $p = 5, 47, 101, 1009, 2003$.
- The plaintext polynomial was fixed at degree 6, and encryption was performed using three known modulus polynomials, each of degree at most 3.
- Cantor–Zassenhaus outperforms most methods, especially in lower fields, due to its efficient randomized factorization mechanism.

Comparison

- Lagrange Interpolation shows strong performance across all fields, as it only depends on solving for coefficients given a set of values, making it computationally light.
- Berlekamp's method shows a gradual increase in runtime but remains competitive, especially effective in small to mid-sized fields due to its deterministic nature.
- Algebraic recovery methods—Coppersmith, LLL, and Gröbner Basis—are slower overall but are influenced by the number of known plaintext coefficients.
- The performance of algebraic methods would improve significantly if 3 or more coefficients were known, reducing the solution space and enabling faster recovery.
- While interpolation and factorization methods dominate in speed, algebraic techniques offer deeper recovery potential, particularly when more side information, like known coefficients, is available.

Comparison



Comparision of cryptanalysis cases

Comparison

- A comparison of Gröbner basis, Coppersmith's method, and LLL lattice reduction is provided, evaluating their performance across different Galois fields and varying levels of known coefficients.

Execution times (in μs) vs. number of known coefficients for $\mathbb{F}_{101}$ (1/3)

Method	**2 Coeffs.**	**3 Coeffs.**	**4 Coeffs.**
Coppersmith	151101	92500	62037
LLL	171038	101200	66503
Gröbner Basis	188529	119328	73206

Execution times (in μs) vs. number of known coefficients for $\mathbb{F}_{1009}$ (2/3)

Method	**2 Coeffs.**	**3 Coeffs.**	**4 Coeffs.**
Coppersmith	265034	162109	103249
LLL	285063	174012	108023
Gröbner Basis	309045	188539	120010

Execution times (in μs) vs. number of known coefficients for $\mathbb{F}_{2003}$ (3/3)

Method	**2 Coeffs.**	**3 Coeffs.**	**4 Coeffs.**
Coppersmith	150035	92519	60205
LLL	171038	101204	66510
Gröbner Basis	188526	119338	73280

Solution

- **Encoding:** We can use random polynomial masking which involves adding a carefully constructed random polynomial $R(x)$ to $N(x)$ in a way that:

$$N_{\text{enc}}(x) \rightarrow N(x) + R(x) \cdot P(x)$$

where $R(x)$ is chosen randomly and$P(x)$ is product of moduli.

- **Agreement on $R(x)$:**
  - $$\deg(R) \geq \deg(N) - \deg(P)$$

    to ensure sufficient masking.
    $R(x)$ must be large to prevent brute-force attacks.
  - Before communication, both parties agree on a secret key K.
    Use a Pseudo-Random Function (PRF) to generate $R(x)$:

$$R(x) = \text{PRF}(K \parallel \text{nonce})$$

where the nonce ensures that $R(x)$ changes per encryption.

Solution

- **Decoding:** The receiver, who knows $R(x)$, can recover $N(x)$ as:

$$N(x) = N_{\text{enc}}(x) - R(x) \cdot P(x)$$

- **Attack Resistant:**
  - The term $R(x) \cdot P(x)$ introduces large, random coefficients, making it impossible to distinguish $r(x)$ from noise.
  - Even if $r(x)$ is small, $N_{\text{enc}}(x)$ appears random.
  - Without $R(x)$, an attacker cannot separate $N(x)$ from $R(x) \cdot P(x)$.
- Helps in prevention of coppersmith attack as small root attack is not feasible.

Solution

This encryption scheme uses the Chinese Remainter Theorem (CRT) for polynomials to break a plaintext into smaller blocks, encrypt them, and later reconstruct them. Here's a concise breakdown:

- **Encryption process:**
  - **Plaintext splitting:**
    1. The plaintext (e.g., PSMFSRND = 151812051817 1303) is divided into smaller blocks (e.g., 1518, 1205, 1817, 1303).
    2. Each block is converted into a polynomial $N_i(x)$ of degree less than the chosen modulus polynomials $p_i(x)$.
  - **Modular reduction:** Each $N_i(x)$ is treated as a residue $b_i(x)$ modulo $p_i(x)$:

$$b_i(x) \equiv N_i(x) \mod p_i(x)$$

The ciphertext $N'(x)$ is computed using:

$$N'(x) = \sum_{i=1}^{s} b_i(x) \cdot M_i(x) \cdot k_i(x) \mod P(x)$$

Solution

- **Decryption process:**
  - **Recover Residues:** Compute $b'_i(x)$ as:

$$b'_i(x) \equiv N'(x) \mod p_i(x)$$

  - **Remove Masking:** Use precomputed inverses $q_i(x)$ (derived from $k_i(x)$) to recover the original residues:

$$b_i(x) \equiv b'_i(x) \cdot q_i(x) \mod p_i(x)$$

  - **Reconstruct plaintext:** Concatenate the coefficients of $b_i(x)$ to recover the original message.

Conclusion

- A symmetric cryptographic algorithm based on the Polynomial Residue Number System (RNS) was analyzed.
- Various potential attacks were studied and evaluated for their effectiveness.
- Cryptanalysis of the algorithm involves combinatorial complexity, leading to an NP-complete problem.
- The implementation of the encryption algorithm was successfully carried out.
- Possible future research directions were identified, including algebraic attacks and algorithm modifications to enhance security.
- The feasibility of Berlekamp's algorithm for factorization-based attacks was examined in different cases.

References

- [1] Iakymenko, I., Karpinski, M., Shevchuk, R., Kasianchuk, M. (2024). Symmetric Encryption Algorithms in a Polynomial Residue Number System. *Journal of Applied Mathematics*, 2024, 4894415:1–12. https://doi.org/10.1155/2024/4894415
- [2] Kasianchuk, M. M., Yakymenko, I. Z., Nykolaychuk, Ya. M. (2021). Symmetric Cryptoalgorithms in the Residue Number System. *Cybernetics and Systems Analysis*, 57(2), 329–336. https://doi.org/10.1007/s10559-021-00358-6
- [3] Luo, Q. (2023). Research and application of Chinese remainder theorem. *Theoretical and Natural Science*, 9, 45–53. https://doi.org/10.54254/2753-8818/9/20240711
- [4] Hoffstein, J., Pipher, J., Silverman, J. H. (1998). NTRU: A ring-based public key cryptosystem. In J. P. Buhler (Ed.), *Algorithmic Number Theory* (pp. 267–288). Springer, Berlin, Heidelberg.

[5] Tiwari, N. K., Chaurasia, B. (2015). Various Approaches towards Cryptanalysis. *International Journal of Computer Applications*, 127(14), 1–5. https://doi.org/10.5120/ijca2015906518

[6] Jakhar, A., Kotyada, S. (2020). On the irreducible factors of a polynomial II. *Journal of Algebra*, 556, 1–20. https://doi.org/10.1016/j.jalgebra.2020.02.045

[7] Berlekamp, E. R. (1967). Factoring polynomials over finite fields. *The Bell System Technical Journal*, 46(8), 1853–1859. https://doi.org/10.1002/j.1538-7305.1967.tb03174.x

[8] Cantor, D. G., Zassenhaus, H. (1981). A New Algorithm for Factoring Polynomials Over Finite Fields. *Mathematics of Computation*, 36(154), 587–592.

[9] Miller, S. D., Narayanan, B., Venkatesan, R. (2021). Coppersmith's lattices and "focus groups": An attack on small-exponent RSA. *Journal of Number Theory*, 222, 376–392. https://doi.org/10.1016/j.jnt.2021.01.002

[10] Herrmann, M., May, A. (2009). Solving Linear Equations Modulo Divisors: On Factoring Given Any Bits. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (pp. 406–424). Springer.

[11] Buchberger, B. (1965). An Algorithm for Finding the Basis Elements of the Residue Class Ring of a Zero Dimensional Polynomial Ideal. *Journal of Symbolic Computation*, 41(3-4), 475–511. https://doi.org/10.1016/j.jsc.2005.09.007

[12] Jakobsen, T., Knudsen, L. R. (1997). The Interpolation Attack on Block Ciphers. In *Fast Software Encryption*, Lecture Notes in Computer Science, 1267, 28–40. Springer. https://doi.org/10.1007/BFb0052332

[13] Hossain, M. K., Sorwar, G., Ahmad, A., Saman, W. (2024). Lattice-Based Cryptanalysis of RSA-Type Cryptosystems: A Bibliometric Analysis. *Cybersecurity*, 7(1), 1–26. https://doi.org/10.1186/s42400-024-00289-7

Thank You!