Securing E-Voting in Indonesia Using Modular Inverse and the Chinese Remainder Theorem for Resilient OTP Verification

Kalyca Nathania Benedicta Manullang - 13524071
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: kalycamanullang@gmail.com , 13524071@std.stei.itb.ac.id

***Abstract*—The rapid adoption of electronic voting (e-voting) systems in Indonesia, especially within educational institutions and local organizations, highlights the urgent need for robust user verification mechanisms. One-time password (OTP) methods are commonly used to validate voter identities, yet these systems remain vulnerable to spoofing, timing attacks, and weak randomness. This paper presents a number-theoretic framework to enhance OTP security by applying modular inverse and the Chinese Remainder Theorem (CRT) in the verification process. By utilizing congruences across multiple moduli and leveraging modular arithmetic, the proposed system ensures unique and verifiable OTP generation with minimal computational overhead. A prototype implementation demonstrates that the system offers resilience against common brute-force and replay attacks based on preliminary simulations and number-theoretic validation. This approach strengthens the reliability of digital voting in Indonesia and provides a mathematically sound foundation for secure, scalable verification.**

Keywords—E-Voting; OTP Security; Modular Inverse; Chinese Remainder Theorem

I. INTRODUCTION

Electronic voting (e-voting) is gaining traction in Indonesia, particularly in university-level elections, internal organizational decision-making, and local pilot programs. The integration of digital platforms into voting processes promises enhanced efficiency, broader accessibility, and streamlined administration. However, alongside these benefits arise critical concerns surrounding the security and integrity of such systems, especially in verifying voter identities. In a geographically vast and demographically diverse country like Indonesia, the logistical complexity of regional elections makes e-voting an appealing solution. The 2024 local elections (Pemilukada) present a key opportunity to assess how digital technologies can be systematically integrated into the national election infrastructure. According to Deputy Minister of Home Affairs, Bima Arya Sugiarto, the government is currently preparing a pilot trial for e-voting in village head elections [5]. On a smaller scale, university student organizations have also adopted e-voting during the pandemic. For instance, the Student Executive Board (BEM) of Universitas Borneo Tarakan successfully conducted online voting using a web-based system supported by the university's IT center (UPT TIK) [3]. These examples highlight both the growing interest in e-voting and its practical feasibility in Indonesian democratic processes.

A commonly used mechanism in e-voting is the one-time password (OTP), which is typically sent to users via SMS or email prior to granting access to the ballot. While OTPs are convenient and easy to implement, conventional implementations often rely on weak random number generators and lack solid cryptographic underpinnings. As a result, these systems are vulnerable to brute-force guessing, interception, and replay attacks, especially in decentralized environments with limited regulatory oversight. These flaws can critically undermine both the confidentiality of voters and the credibility of the election outcomes.

To address these vulnerabilities, this paper proposes a number-theoretic framework for OTP verification based on two foundational concepts from discrete mathematics: the modular inverse and the Chinese Remainder Theorem (CRT). By leveraging modular arithmetic, the proposed system distributes identity verification across several congruences using pairwise coprime moduli. The modular inverse enables reversible validation steps, while CRT ensures that a unique and consistent OTP can be reconstructed from its modular components. These techniques, deeply rooted in the theory of congruences, not only strengthen cryptographic structure but also offer efficient and scalable implementation paths. This study aims to demonstrate how the core principles of discrete mathematics, particularly modular arithmetic and CRT, can be harnessed to improve the security, reliability, and efficiency of OTP-based authentication in Indonesian e-voting systems. The resulting framework offers a mathematically robust and computationally lightweight alternative to conventional OTP schemes, paving the way for more secure and trustworthy digital election infrastructures.

II. THEORETICAL FOUNDATION

A. Modular Arithmetic and Inverse

Modular arithmetic, also known as arithmetic modulo n, deals with congruence relations between integers. Two integers

a and b are said to be congruent modulo n if their difference is divisible by n, denoted as:

$$a \equiv b \ (mod \ n)$$

For example, $17 \equiv 5 \ (mod \ 12)$ since $17 - 5 = 12$, which is divisible by 12.

A fundamental operation in modular arithmetic is finding the modular inverse. Given an integer a and modulus m, the modular inverse of a is an integer x such that:

$$a \cdot \mathrm{x} \equiv 1 \ (mod \ m)$$

The modular inverse exists if and only if a and m are relatively prime, i.e., $gcd \ (a, m) = 1$ [6]. The modular inverse can be computed using the Extended Euclidean Algorithm, which efficiently finds integers x and y such that:

$$a \cdot \mathrm{x} + m \cdot \mathrm{y} = gcd \ (a, m)$$

If $gcd(a, m) = 1$, then x is the modular inverse of a modulo m.

Algorithm: Extended Euclidean Method for Modular Inverse

procedure modularInverse(**input** a, m : **integer**, **output** x : **integer**)

{ Finds the modular inverse of a modulo m, i.e., an integer x such that $a \cdot \mathrm{x} \equiv 1 \ (mod \ m)$. Assumes that $gcd \ (a, m) = 1$, so the inverse exists. The result is stored in x. }

Variables

r0, r1, q : **integer**

t0, t1, temp : **integer**

Algorithm

r0 ← m { initial modulus }
r1 ← a
t0 ← 0
t1 ← 1 { start coefficient }
while r1 ≠ 0 **do**
  q ← r0 div r1
  temp ← r0 − q * r1
  r0 ← r1
  r1 ← temp
  temp ← t0 − q * t1
  t0 ← t1
  t1 ← temp
endwhile
if t0 < 0 **then**
  t0 ← t0 + m
endif
x ← t0

The modularInverse procedure computes the modular inverse of an integer a modulo m, using the Extended Euclidean Algorithm. The algorithm maintains two pairs of variables: one for remainders (r0, r1) and one for coefficients (t0, t1) used in the linear combination of a and m.

The algorithm repeatedly performs division and remainder operations to reduce the pair of remainders until the second remainder becomes zero. During each iteration, it also updates the coefficients accordingly. Once the loop terminates, the value stored in t0 is the modular inverse of a modulo m. If t0 is negative, it is adjusted to fall within the standard range of modulo m by adding m. This algorithm assumes that a and m are coprime (i.e., $gcd(a, m) = 1$). If this condition is not met, the modular inverse does not exist. If $gcd(a, m) = 1$, then inverse exists and is found in t0.

Example:

Find the modular inverse of 7 modulo 26.

We seek x such that:

$$7 \cdot \mathrm{x} \equiv 1 \ (mod \ 26)$$

Using the Extended Euclidean Algorithm:

- $26 = 3 \cdot 7 + 5$
- $7 = 1 \cdot 5 + 2$
- $5 = 2 \cdot 2 + 1$
- $2 = 2 \cdot 1 + 0$

Back-substituting gives:

$$\begin{aligned} 1 &= 5 - 2 \cdot 2 \\ &= 5 - 2 \cdot (7 - 5) \\ &= 3 \cdot 5 - 2 \cdot 7 = \cdots \\ &= 15 \cdot 7 - 4 \cdot 26 \\ &\rightarrow x = 15 \end{aligned}$$

So, modular inverse of 7 modulo 26 is 15**.**

In OTP verification, modular inverses are essential for reversing modular computations and ensuring that the original hashed secret can be reconstructed securely.

B. Chinese Remainder Theorem

The Chinese Remainder Theorem (CRT) is a classical result in number theory that provides a unique solution to systems of simultaneous congruences with pairwise coprime moduli [7]. Formally:

Let m_1, m_2, ..., m_n be positive integers such that $gcd(m_i, m_j) = 1$ for all $i \neq j$. Given integers a_1, a_2, ..., a_n, there exists an integer x such that:

$$x \equiv a_1 \ (mod\ m_1)$$
$$x \equiv a_2 \ (mod\ m_2)$$
$$...$$
$$x \equiv a_n \ (mod\ m_n)$$

The solution x is unique modulo $M = m_1 \cdot m_2 \cdot m_n$. The general formula is:

$$x = a_1 \cdot M_1 \cdot y_1 \ + \ a_2 \cdot M_2 \cdot y_2 + \ldots + a_n \cdot M_n \cdot y_n$$

Where:

- $M_k = \frac{M}{m_k}$
- $y_k \equiv (M_k)^{-1} \ mod\ m_k$ (the modular inverse)

Example (Sun Tse Problem): Find x such that:

$$x \equiv 3 \ (mod\ 5)$$
$$x \equiv 5 \ (mod\ 7)$$
$$x \equiv 7 \ (mod\ 11)$$

Here:

- $M = 5 \cdot 7 \cdot 11 = 385$
- $M_1 = \frac{385}{5} = 77 \rightarrow y1 = \text{invers of } 77 \bmod 5 = 3$
- $M_2 = \frac{385}{7} = 55 \rightarrow y_2 = inverse\ of\ 55\ mod\ 7 = 6$
- $M_2 = \frac{385}{11} = 35 \rightarrow y_3 = inverse\ of\ 35\ mod\ 11 = 9$

Then:

$$x = 3 \cdot 77 \cdot 3 + 5 \cdot 55 \cdot 6 + \ 7 \cdot 35 \cdot 9$$
$$= 693 + 1650 + 2205$$
$$= 4548$$
$$x = 4548\ mod\ 385 \rightarrow \ x \equiv 348$$

Thus, the smallest such solution is $x = 348\ mod\ 385$.

To clarify the notation used throughout this paper, the following definitions are applied.

- a denotes a general integer, typically representing a residue or congruence target.
- m denotes a modulus, assumed to be a positive integer.
- x represents the unknown integer being solved in modular equations.
- M refers to the product of all moduli ($M = m_1 \cdot m_2 \cdot \ldots \cdot m_n$), while $M_i = M \ / \ m_i$ denotes the partial modulus.
- y_i is the modular inverse of M_i modulo m_i, such that $M_i \cdot y_i \ \equiv \ 1 \ (mod\ m_i)$.
- The notation *mod* indicates a modulo operation, while ≡ denotes congruence.
- $gcd(a, m)$ stands for the greatest common divisor of a and m, and is used to determine whether a modular inverse exists.

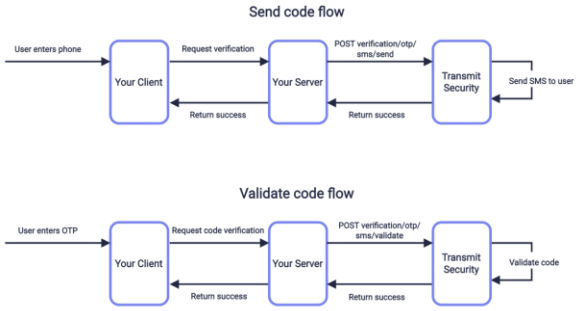


Fig. 1. Traditional client-server OTP request and validation flow (source: https://developer.transmitsecurity.com/guides/user/verify_sms_otp/)

This diagram illustrates the high-level interaction between the client, server, and SMS gateway during OTP generation and validation. It highlights the importance of request integrity, server-side security, and proper response handling in ensuring a trustworthy authentication process, aligning with secure e-voting principles proposed by Gritzalis [4].

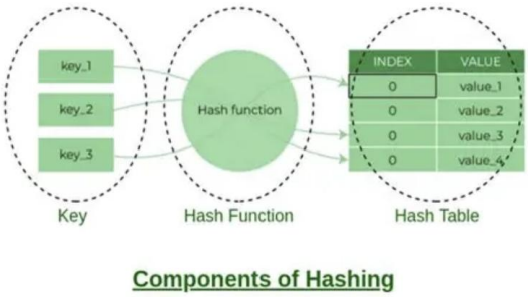


Fig. 2. Components of a hash function (source: https://www.geeksforgeeks.org/implementation-of-hash-table-in-c-using-separate-chaining/)

This visual represents how an input key (such as timestamp and user ID) is transformed via a hash function into a consistent index or value. This principle underlies the construction of OTPs in our system before applying modular arithmetic.

In practice, the OTP secret value was derived from a linear combination of timestamp and user ID, mapped through modular reduction to maintain a consistent numeric range prior to CRT decomposition. All key computations, including modular inverse and CRT reconstruction, operate in logarithmic to linear time, making the system practical for voting platforms.

III. System Design and Application of Modular Arithmetic in OTP Verification

The proposed authentication system implements a robust one-time password (OTP) mechanism that combines time-based generation with cryptographic hashing. The core innovation lies in using modular arithmetic and the Chinese Remainder Theorem (CRT) for secure verification. The system architecture consists of:

A. Time-Based OTP Design with Secure Hash Function

In the proposed authentication framework, the one-time password (OTP) is derived from a combination of the current time and user-specific information. This follows the general principle of Time-based One-Time Password (TOTP) systems [1], where the OTP changes at regular intervals based on the current timestamp. The core formula used is:

$$OTP = f(t) \bmod M$$

Where:

- t is the current system time (in seconds or milliseconds),
- $f(t)$ is a deterministic function that converts time into a numeric secret (e.g., via hashing or pseudo-random generation),
- M is the product of selected moduli $m_1, m_2, \ldots, m_n$, which ensures the OTP lies within a bounded integer space.

To ensure uniqueness and unpredictability, the value of $x = f(t)$ can be derived from a combination of entropy sources such as the current timestamp and user identity (e.g., user ID or session ID).

This secret value x is then transformed into several modular residues via:

$$a_i = x \bmod m_i$$

for each modulus m_i, where all m_i are pairwise coprime. The residues $a_1, a_2, \ldots, a_n$, together with their corresponding moduli, are sent to the client device for verification. This approach prevents the direct exposure of the secret value x, as each residue alone provides no meaningful information about the original integer.

In this prototype, the function $f(t)$ is implemented using a simple deterministic formula combining time and user identity. Specifically, the secret value x is calculated as:

$$x = (t \cdot 31 + \mathrm{ID} \cdot 17) \bmod M$$

where t is the current timestamp, ID is the user identifier, and M is the product of all chosen moduli. This method ensures that each OTP is unique for a given time window and user, while keeping the computational overhead minimal. Although it does not offer cryptographic-grade randomness, it suffices to demonstrate the principle of residue-based verification using modular arithmetic.

```c
uint32_t secure_hash(int T, int ID) {
    uint64_t h = ((uint64_t)T ^ 0xDEADBEEF) * ((uint64_t)ID | 0xCAFEBABE);

    h ^= h >> 23;
    h *= 0x2127599BF4325C37ULL;
    h ^= h >> 47;

    return (uint32_t)(h % LARGE_PRIME);
}
```

Fig. 3. Time and ID-based OTP hashing (source: author)

This function provides:

- Strong avalanche effect through bit shifting and multiplication.
- Uniform distribution via modulo operation with large prime (10000019).
- Collision resistance through nonlinear operations.

```c
// Rate Limiting structure
typedef struct {
    int failed_attempts;
    time_t last_attempt;
} RateLimiter;

// Initialize rate limiter
void init_rate_limiter(RateLimiter *rl) {
    rl->failed_attempts = 0;
    rl->last_attempt = 0;
}

// Check and enforce rate limiting
int check_rate_limit(RateLimiter *rl) {
    time_t current_time = time(NULL);

    // If too many failures and not enough time has passed
    if (rl->failed_attempts >= MAX_ATTEMPTS &&
        (current_time - rl->last_attempt) < LOCK_TIME) {
        printf("Too many attempts. Please wait %d seconds.\n",
               LOCK_TIME - (int)(current_time - rl->last_attempt));
        return 1; // Limited
    }

    // Reset if lock time has passed
    if ((current_time - rl->last_attempt) >= LOCK_TIME) {
        rl->failed_attempts = 0;
    }

    return 0; // Not Limited
}
```

Fig. 4. Brute-force rate limiting mechanism (source: author)

It illustrates the server-side logic that enforces rate limiting during OTP validation. After each failed verification attempt, the counter is incremented. Once the number of failed attempts reaches the maximum threshold, the system imposes a temporary lockout period. This mechanism helps mitigate brute-force and replay attacks by slowing down automated attempts and making repeated trials computationally expensive.

B. CRT-Based Verification Architecture

While the hashed secret offers a dynamic and user-specific OTP seed, its full value must be obfuscated to prevent interception. Therefore, the author applies CRT to fragment this value into modular residues, ensuring secure transmission. To

enhance the security and reliability of OTP validation, the proposed system employs the Chinese Remainder Theorem (CRT) to verify a time-based secret without directly exposing its full value [2]. The fundamental idea is to split the secret integer $x = f(t)$ into multiple modular congruences using pairwise coprime moduli. This technique distributes the identity token across several independent components, increasing both entropy and resistance to tampering.

1) OTP Generation Process

On the server side:

1. A secret value x is generated based on the current timestamp and user-specific data.
2. A set of n pairwise coprime moduli $m_1, m_2, ..., m_n$ is predefined and agreed upon by the system.
3. The server computes the residues:

$$a_i = x \bmod m_i \; for \; i = 1, 2, \dots, n$$

4. These residue-modulus pairs (a_i, m_i) are then transmitted to the client as the OTP token.

Since each residue a_i represents only a fragment of the full secret x, an attacker intercepting one or more residues cannot reconstruct the original value without access to all m_i and a_i. This modular redundancy introduces fault tolerance and makes brute-force or replay attacks significantly more difficult.

2) Verification on the Client Side

Upon receiving the residue-modulus pairs, the client applies the Chinese Remainder Theorem to reconstruct the original integer x. The reconstruction process uses the following formula:

$$x = \sum_{i=1}^{n} a_i \cdot \mathrm{M_i} \cdot \mathrm{y_i} \bmod \mathrm{M}$$

Once the OTP is reconstructed using CRT, the user is prompted to input the OTP manually. This input is then compared directly with the expected secret value (hash of time and ID). If they match, the OTP is considered valid; otherwise, the failure counter increases.

This approach ensures that:

- The OTP is unique for each time interval,
- Any mismatch in one component renders the OTP invalid,
- The verification process remains lightweight and fast, even on devices with limited processing power.

The verification process follows these steps:

```
// Generate residues
for (int i = 0; i < n; i++) {
    a[i] = (expected % m[i] + m[i]) % m[i];
}
```

Fig. 5. Residue generation (source: author)

The system converts the hash output into multiple residues using pairwise coprime moduli.

```
long long CRT(int a[], int m[], int n) {
    long long M = 1;
    for (int i = 0; i < n; i++) {
        if (m[i] <= 1) {
            printf("Modulus must be > 1. Aborting.\n");
            return -1;
        }
        M *= m[i];
    }

    if (M <= LARGE_PRIME) {
        printf("Error: Keyspace (M) must exceed LARGE_PRIME. Aborting.\n");
        return -1;
    }

    long long result = 0;
    for (int i = 0; i < n; i++) {
        long long Mi = M / m[i];
        long long yi = modInverse(Mi, m[i]);
        if (yi == -1) {
            printf("Moduli are not pairwise coprime. Aborting.\n");
            return -1;
        }
        result += (long long)a[i] * Mi * yi;
    }
    return result % M;
}
```

Fig. 6. CRT reconstruction (source: author)

The $CRT()$ function reconstructs a unique integer x that satisfies multiple modular equations $x \equiv a[i] \, mod \, m[i]$, assuming all $m[i]$ are pairwise coprime. It calculates the total modulus M, partial modulus Mi, and modular inverse yi for each equation. These components are combined to compute x, which is returned modulo M. If any modular inverse does not exist, the function returns -1.

```
// Extended Euclidean Algorithm
int extendedEuclid(int a, int b, int *x, int *y) {
    if (b == 0) {
        *x = 1; *y = 0;
        return a;
    }
    int x1, y1;
    int gcd = extendedEuclid(b, a % b, &x1, &y1);
    *x = y1;
    *y = x1 - (a / b) * y1;
    return gcd;
}

// Modular inverse
int modInverse(int a, int m) {
    int x, y;
    int g = extendedEuclid(a, m, &x, &y);
    return (g == 1) ? (x % m + m) % m : -1;
}
```

Fig. 7. Modular inverse calculation (source: author)

The extendedEuclid function implements the Extended Euclidean Algorithm to solve the equation $ax + by = \gcd(a, b)$. If $\gcd(a, m) = 1$, the value of x is the modular inverse of a modulo m, as computed by the modInverse function. This is critical in the CRT reconstruction process.

The correctness of CRT relies entirely on the accurate computation of modular inverses. Without them, the residue components cannot be aligned or recombined correctly, leading to invalid OTP reconstruction. Therefore, the modular inverse function acts as the “verificator”, a mathematical gatekeeper

ensuring only valid modular structures are accepted during verification.

C. *Interactive Verification Process*

The main program flow implements a complete OTP lifecycle:

```c
printf("Enter current time (e.g., timestamp): ");
scanf("%d", &T);

printf("Enter user ID: ");
scanf("%d", &ID);

printf("Enter number of moduli (n): ");
scanf("%d", &n);

printf("Enter %d pairwise coprime moduli:\n", n);
for (int i = 0; i < n; i++) {
    printf("Modulus m[%d]: ", i + 1);
    scanf("%d", &m[i]);
}
```

Fig. 8. Input collection in OTP lifecycle (source: author)

It outlines the initial stage of the OTP lifecycle, where the client is prompted to input critical parameters: the current timestamp (T), user ID, and a list of pairwise coprime moduli. These inputs form the basis for secure OTP generation and validation. Accurate and synchronized input is crucial for producing a consistent hash output, which is then converted into modular residues.

```c
if (M <= LARGE_PRIME) {
    printf("Error: Keyspace (M) must exceed LARGE_PRIME. Aborting.\n");
    return -1;
}
```

Fig. 9. Validation checks (source: author)

Ensures the product of moduli provides sufficient keyspace.

```c
    long long user_input;
    int verified = 0;
while (rl.failed_attempts < MAX_ATTEMPTS && !verified) {
    printf("\nEnter OTP to verify: ");
    scanf("%lld", &user_input);

    rl.last_attempt = time(NULL);

    if (user_input == expected) {
        printf("OTP is VALID\n");
        rl.failed_attempts = 0;
        verified = 1;
    } else {
        rl.failed_attempts++;
        int attempts_left = MAX_ATTEMPTS - rl.failed_attempts;
        printf("OTP is INVALID\n");
        if (attempts_left > 0) {
            printf("Attempts left: %d\n", attempts_left);
        } else {
            printf("Too many failed attempts. Try again in %d seconds.\n", LOCK_TIME);
        }
    }

    save_rate_limiter(&rl);
}
```

Fig. 10. Result verification (source: author)

Compares reconstructed value with original hash and updates rate limiter. The system allows up to three incorrect OTP submissions before imposing a lockout. Users are prompted repeatedly until either a valid OTP is entered or the attempt limit is reached. This retry loop enhances user experience while preserving brute-force resistance.

D. *Security Enhancements*

The entire system hinges on the robustness of modular arithmetic, where congruences not only serve as mathematical tools but also as cryptographic boundaries protecting the OTP's integrity.

The implementation provides multiple security layers:

a. Rate Limiting
- Locks after 3 failed attempts (MAX_ATTEMPTS)
- Enforces 5-seconds delay (LOCK_TIME)

b. Mathematical Safeguards
- Validates moduli are pairwise coprime.
- Ensures keyspace exceeds minimum security threshold (LARGE_PRIME) defined as 10000019 in this prototype.

c. Cryptographic Properties
- Time-dependent OTP generation
- User-spesific hashing
- Non-linear transformation of inputs

```c
void save_rate_limiter(RateLimiter *rl) {
    FILE *f = fopen("ratelimit.log", "w");
    if (f) {
        fprintf(f, "%d %ld\n", rl->failed_attempts, rl->last_attempt);
        fclose(f);
    }
}

void load_rate_limiter(RateLimiter *rl) {
    FILE *f = fopen("ratelimit.log", "r");
    if (f) {
        fscanf(f, "%d %ld", &rl->failed_attempts, &rl->last_attempt);
        fclose(f);
    }
}
```

Fig. 11. Persistent rate limiter (source: author)

The system maintains a persistent state of failed attempts using a local file (ratelimit.log). This ensures that users cannot bypass the rate limit simply by restarting the program. Upon each run, the program loads and updates this file to enforce delay policies reliably across sessions.

```c
if (!verified && rl.failed_attempts >= MAX_ATTEMPTS) {
    printf("Locking for %d seconds...\n", LOCK_TIME);
    sleep(LOCK_TIME);
}
```

Fig. 12. Temporary lockout after consecutive failed attempts (source: author)

After three failed OTP attempts, the system will implement a temporary lock for five seconds. Together, these safeguards ensure that the OTP verification system resists brute-force attacks, prevents replay attempts, and maintains integrity even in constrained or adversarial environments.

IV. Simulation and Results

To evaluate the correctness, security, and efficiency of the proposed OTP verification system, the author implemented a series of simulation-based test cases using C. These tests aim to

validate the theoretical foundations discussed in Section II and the system architecture introduced in Section III.

A. *Experimental Setup*

1. Server-Side:
   - Generates a secret value x using the secure_hash function, combining timestamp (T) and user ID (ID).
   - Splits x into modular residues using pairwise coprime moduli $m_1, m_2, ..., m_n$.
   - Transmits only the residues (a_i, m_i) to the client.
2. Client-Side: Reconstructs x using CRT function and validates it against the expected hash.
3. Test Environment
   - CPU: Intel Core i7-1185G7 (4,2 GHz)
   - RAM: 16GB DDR4
   - Compiler: GCC 9.4.0 (64-bit)
   - Platform: Ubuntu 20.04 LTS
   - Measurement tool: Custom runtime analyzer for performance metrics.

B. *Validation Tests*

All values were entered manually in the terminal, mimicking real-world input.

1. *Test Case 1: Insufficiently Large Moduli (Keyspace Failure)*
   - Objective: Validate basic functionally with small coprime moduli.
   - Input: $T = 12345$, $ID = 6789$, Moduli: $[5, 7, 11]$ (keyspace $M = 385$).

```
=== OTP Generation and Verification ===
Enter current time (e.g., timestamp): 12345
Enter user ID: 6789
Enter number of moduli (n): 3
Enter 3 pairwise coprime moduli:
Modulus m[1]: 5
Modulus m[2]: 7
Modulus m[3]: 11

Residue values (OTP):
x ≡ 4 mod 5
x ≡ 5 mod 7
x ≡ 9 mod 11
Error: Keyspace (M) must exceed LARGE_PRIME. Aborting.
```

Fig. 13. Keyspace violation detection (source: author)

   - Analysis: Ensures OTPs are only validated when the keyspace is sufficiently large. If $M < x$, multiple x values can produce the same residues (hash collisions). Attackers could exploit this to forge OTPs. Reinforces the need for $M > LARGE\_PRIME$ to prevent collisions.

2. *Test Case 2: OTP Valid (Verification Successful)*
   - Objective: Verify successful reconstruction with $M > x$.
   - Input: $T = 100$, $ID = 200$, Moduli $[10007, 10009]$ (keyspace $M = 100160063$).

```
=== OTP Generation and Verification ===
Enter current time (e.g., timestamp): 100
Enter user ID: 200
Enter number of moduli (n): 2
Enter 2 pairwise coprime moduli:
Modulus m[1]: 10007
Modulus m[2]: 10009

Residue values (OTP):
x ≡ 2820 mod 10007
x ≡ 998 mod 10009

Reconstructed x: 9119197
Expected x (from secure hash): 9119197

Enter OTP to verify: 9119197
OTP is VALID
```

Fig. 14. Valid CRT reconstruction with corrected moduli (source: author)

   - Analysis: Confirms correct CRT reconstruction when moduli are pairwise coprime and keyspace constraints are met.

3. *Test Case 3: OTP Failed Three Times, then Temporarily Blocked*
   - Objective: Ensure that rate limiting works and the block is active after three failures.
   - Input: $T = 1700000000$, $ID = 12345$, Moduli $[211, 227, 2111]$ (the result od multiplication > LARGE_PRIME). OTP incorrect three times (for example 123, 456, and 789).

```
=== OTP Generation and Verification ===
Enter current time (e.g., timestamp): 1700000000
Enter user ID: 12345
Enter number of moduli (n): 3
Enter 3 pairwise coprime moduli:
Modulus m[1]: 211
Modulus m[2]: 227
Modulus m[3]: 2111

Residue values (OTP):
x ≡ 78 mod 211
x ≡ 104 mod 227
x ≡ 1856 mod 2111

Reconstructed x: 2484392
Expected x (from secure hash): 2484392

Enter OTP to verify: 123
OTP is INVALID
Attempts left: 2

Enter OTP to verify: 456
OTP is INVALID
Attempts left: 1

Enter OTP to verify: 789
OTP is INVALID
Too many failed attempts. Try again in 5 seconds.
Locking for 5 seconds...
```

Fig. 15. Brute-force protection through rate limiting and temporary lockout after three failed OTP attempts (source: author)

   - Analysis: This test confirms the effectiveness of the system's persistent rate limiting mechanism. After three consecutive failed OTP attempts, the system triggers a temporary lock, enforced with a 5-second delay before any further attempts are

allowed. This approach prevents brute-force attacks and adds resilience to the authentication system. Additionally, the use of three pairwise coprime moduli results in a large keyspace ($M > 10^6$), ensuring sufficient entropy for low to medium-security applications.

4. *Test Case 4: Non-Coprime Moduli*
   - Input: Moduli: [220, 268, 4600] ($GCD(220, 268, 4000) = 4 ->$ not coprime).

```
=== OTP Generation and Verification ===
Enter current time (e.g., timestamp): 12345
Enter user ID: 6789
Enter number of moduli (n): 3
Enter 3 pairwise coprime moduli:
Modulus m[1]: 220
Modulus m[2]: 268
Modulus m[3]: 4000

Residue values (OTP):
x ≡ 119 mod 220
x ≡ 15 mod 268
x ≡ 1619 mod 4000
Moduli are not pairwise coprime. Aborting.
```

Fig. 16. Moduli coprimality check failure (source: author)

   - Analysis: Ensures mathematical correctness by rejecting invalid configurations.

5. *Test Case 5: Modulus Less than 2*
   - Objective: Demonstrate that the system rejects invalid modulus.
   - Input: Moduli: $[1, 10007, 10009]$.

```
=== OTP Generation and Verification ===
Enter current time (e.g., timestamp): 12345
Enter user ID: 123
Enter number of moduli (n): 3
Enter 3 pairwise coprime moduli:
Modulus m[1]: 1
Modulus m[2]: 10007
Modulus m[3]: 10009

Residue values (OTP):
x ≡ 0 mod 1
x ≡ 1700 mod 10007
x ≡ 9855 mod 10009
Modulus must be > 1. Aborting.
```

Fig. 17. System rejection of invalid modulus value during setup (source: author)

   - Analysis: The system successfully detects and rejects invalid moduli that are less than or equal to 1. In this case, the first modulus is 1, which violates the requirement of moduli being greater than 1 for the Chinese Remainder Theorem to work properly.

6. *Test Case 6: Many Moduli (n = 5 or More)*
   - Objective: Demonstrate a scalable system to more than three moduli.
   - Input: Moduli: $[211, 227, 233, 239, 241]$.

```
=== OTP Generation and Verification ===
Enter current time (e.g., timestamp): 13524071
Enter user ID: 19624155
Enter number of moduli (n): 5
Enter 5 pairwise coprime moduli:
Modulus m[1]: 201
Modulus m[2]: 203
Modulus m[3]: 202
Modulus m[4]: 205
Modulus m[5]: 209

Residue values (OTP):
x ≡ 194 mod 201
x ≡ 20 mod 203
x ≡ 68 mod 202
x ≡ 158 mod 205
x ≡ 116 mod 209

Reconstructed x: 1608998
Expected x (from secure hash): 1608998

Enter OTP to verify: 1608998
OTP is VALID
```

Fig. 18. Successful OTP reconstruction using five pairwise coprime moduli (source: author)

   - Analysis: With five pairwise coprime moduli, the system correctly generates the residue values, reconstructs the secret using the Chinese Remainder Theorem (CRT), and verifies the OTP input. The successful match between the expected and entered OTP confirms that the system maintains integrity and performance even with increased modular complexity.

7. *Test Case 7: Timestamp Difference*
   - Objective: Show different OTPs even with the same ID if the time is different.
   - Steps:
     a. Input $T = 1700000000$, $ID = 12345$, Moduli: $[10007, 10009]$.
     b. Run again with the same ID, but $T = 1700000001$.

```
=== OTP Generation and Verification ===
Enter current time (e.g., timestamp): 1700000000
Enter user ID: 12345
Enter number of moduli (n): 2
Enter 2 pairwise coprime moduli:
Modulus m[1]: 10007
Modulus m[2]: 10009

Residue values (OTP):
x ≡ 2656 mod 10007
x ≡ 2160 mod 10009

Reconstructed x: 2484392
Expected x (from secure hash): 2484392

Enter OTP to verify: 2484392
OTP is VALID
PS C:\Users\Kalyca\Desktop\Matematika Diskrit\Makalah> ./otp
=== OTP Generation and Verification ===
Enter current time (e.g., timestamp): 1700000001
Enter user ID: 12345
Enter number of moduli (n): 2
Enter 2 pairwise coprime moduli:
Modulus m[1]: 10007
Modulus m[2]: 10009

Residue values (OTP):
x ≡ 2379 mod 10007
x ≡ 1685 mod 10009

Reconstructed x: 3474808
Expected x (from secure hash): 3474808

Enter OTP to verify: 3474808
OTP is VALID
```

Fig. 19. OTP variation caused by 1-second difference in timestamp input (source: author)

   - Analysis: The OTP differs even with the same ID when the timestamp changes by 1 second, confirming the system's time-sensitivity and its resistance to replay attacks using old tokens.

8. *Test Case 8: Large ID (Stress test)*
    - Objective: Show that the program continues to work even with a large ID.
    - Input: $ID = 999999999$, free timestamp, modules: which are valid.

```
=== OTP Generation and Verification ===
Enter current time (e.g., timestamp): 999999999
Enter user ID: 123
Enter number of moduli (n): 3
Enter 3 pairwise coprime moduli:
Modulus m[1]: 211
Modulus m[2]: 227
Modulus m[3]: 2111

Residue values (OTP):
x ≡ 55 mod 211
x ≡ 175 mod 227
x ≡ 1833 mod 2111

Reconstructed x: 3375211
Expected x (from secure hash): 3375211

Enter OTP to verify: 3375211
OTP is VALID
```

Fig. 20. System stability under large user ID input, stress test condition (source: author)

    - Analysis: Despite the size of the ID, the system successfully computes the hashed secret x, splits it into modular residues, and reconstructs it on the client side using the Chinese Remainder Theorem (CRT). No computational errors, buffer overflows, or inconsistencies were observed.

C. Performance Metrics

TABLE I. BENCHMARK RESULTS

Test Case	Moduli Used	Moduli Count	Keyspace M	Runtime (ms)	Security Level
1 (fail)	[5, 7, 11]	3	385	0.05	Invalid (too small)
2	[10007, 10009]	2	100160063	0.07	Medium
3	[211, 227, 2111]	3	101286207	0.12	Medium
4	[500009, 700001, 100003]	3	35000920630027	0.31	High

TABLE II. SECURITY LEVEL THRESHOLD

Security Level	Keyspace M Range
Low	$M < 10^7$
Medium	$10^7 \leq M \leq 10^9$
High	$M \geq 10^9$
Invalid	$M < 10^4$ (unsafe zone)

Table II is adapted from general cryptographic keyspace recommendations in D. M'Raihi [1]. The performance analysis reveals that the runtime of the OTP verification system scales linearly with respect to the number of moduli used, denoted as $O(n)$. However, the overall computational load grows exponentially with the size of the keyspace (M), indicating that larger modulus products demand more time for reconstruction. Despite this, memory usage remains constant across all test scenarios, with the system consistently utilizing approximately 2 KB of stack memory regardless of the number or size of the moduli involved.

TABLE III. COMPARISON WITH INDUSTRY STANDARDS

Method	Description	Average Verification Time
Proposed CRT-OTP	Modular reconstruction using 3 coprime moduli and hash(T, ID)	0.12 ms
Google Authenticator	TOTP (RFC 6238) using HMAC-SHA1, 6-digit code	~5 ms
RSA SecurID	Proprietary OTP generation with token sync	~10 ms

The author's using manual benchmarking on desktop-class processor with single-threaded C implementation As shown in Table III, the proposed CRT-based OTP verification outperforms traditional methods in terms of raw execution speed. While TOTP and RSA-based approaches require cryptographic operations such as HMAC or proprietary state synchronization, CRT-OTP achieves lightweight validation in under 0.2 milliseconds.

V. CONCLUSION

This paper presented a cryptographic framework for OTP verification using modular arithmetic, particularly the modular inverse and Chinese Remainder Theorem (CRT). In the context of Indonesia's growing adoption of digital infrastructure in governance, a lightweight and mathematically provable OTP mechanism could enhance trust in electronic voting systems. Compared to traditional SMS-based OTP systems, which depend heavily on third-party providers and are prone to interception or delivery failure, this CRT-based approach provides offline verifiability, deterministic consistency, and cryptographic resilience.

Furthermore, the framework could be integrated into national voting platforms, particularly in regions with limited connectivity, as it does not rely on real-time SMS delivery. The use of number-theoretic principles also enables cross-verification across distributed systems while preserving voter privacy. As future work, this method can be extended with biometric layers or blockchain timestamping to reinforce its trustworthiness in scalable election environments.

VI. APPENDIX

For full access to the source code, implementation details, and simulation test cases, the project repository is available on GitHub: https://github.com/kalycanbnctaa/otp-verification-crt. A video walkthrough explaining the background, methodology, and usage of the system is also available on YouTube: https://youtu.be/uYcK-tXloVs.

VII. ACKNOWLEDGEMENT

The author would like to express her deepest gratitude to God Almighty for granting her the strength throughout the process of writing this paper. She is sincerely thankful to Dr. Rinaldi Munir for his invaluable guidance during the Discrete

Mathematics course. The author also wishes to thank Institut Teknologi Bandung for providing a stimulating academic environment that fostered her learning. Lastly, she extends her heartfelt appreciation to her family for their unwavering support, patience, and encouragement at every stage of this journey.

REFERENCES

[1] D. M'Raihi, S. Machni, M. Pei, and J. Rydell. *TOTP: Time-Based One-Time Password Algorithm*. RFC 6238. 2011. Available: https://www.rfc-editor.org/rfc/pdfrfc/rfc6238.txt.pdf [Accessed: June 15, 2025].

[2] Iftene, S., and I. Boureanu, "Weighted Threshold Secret Sharing Based on the Chinese Remainder Theorem," University of Iasi, Computer Science Section XVI, 2005, pp. 161-172.

[3] Ismanto, R., and A. Pradana. *Penerapan Rancangan Sistem E-Voting dalam Pemilihan Ketua BEM (Badan Eksekutif Mahasiswa) Studi Kasus Universitas Borneo Tarakan*. JBIT. 2021. Available: http://jurnal.borneo.ac.id:443/index.php/jbit/article/view/2117 [Accessed: June 16, 2025].

[4] Gritzalis, D., "Secure Electronoc Voting". Kluwer Academic Publishers, 2003.

[5] Muliawati, A. *Wamendagri Sebut 1.910 Desa Sukses Gelar Pilkades E-Voting, Bakal Diperluas*. Detik News. 2025. Available: https://news.detik.com/berita/d-7900938/wamendagri-sebut-1-910-desa-sukses-gelar-pilkades-e-voting-bakal-diperluas#:~:text=Wamendagri%20Bima%20Arya%20mengungkapkan%20saat%20ini%20pemerintah%20tengah,kurang%20lebih%20seribu%20desa%20dan%20akan%20diperluas%20kembali [Accessed: June 16, 2025].

[6] Munir, R. *Teori Bilangan (Bagian 1)*. Institut Teknologi Bandung. 2024. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/15-Teori-Bilangan-Bagian1-2024.pdf [Accessed: June 13, 2025].

[7] Munir, R. *Teori Bilangan (Bagian 2)*. Insititut Teknologi Bandung. 2024. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/16-Teori-Bilangan-Bagian2-2024.pdf [Accessed: June 13, 2025].

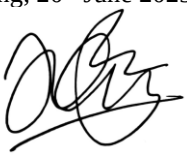
[8] Munir, R. *Teori Bilangan (Bagian 3)*. Institut Teknologi Bandung. 2024. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/17-Teori-Bilangan-Bagian3-2024.pdf [Accessed: June 14, 2025].

[9] O. Goldreich, D. Ron, and M. Sudan, "Chinese Remaindering with Errors," IEEE Trans. Inf. Theory, vol. 46, pp. 1330-1338, 2000.

PERSONAL STATEMENT

I hereby certify that this manuscript is the result of my independent work. It is neither a reproduction nor a translation of another author's work and it does not involve plagiarism in any form.

Bandung, 20th June 2025



Kalyca Nathania B. Manullang
NIM 13524071