Security Analysis of Zero-Knowledge Proof Protocol for Passwordless Authentication Using Modular Number Theory

Revandra Zacky Maharta - 13525031
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: mahartarevandra@gmail.com, 13525031@std.stei.itb.ac.id

***Abstract*— Authentication systems that rely on passwords are increasingly vulnerable to brute-force attacks and large-scale data breaches. Zero-Knowledge Proof (ZKP) offers an alternative authentication method where a prover convinces a verifier that they possess certain information without revealing any parts of that information itself. This paper analyzes the security of the Schnorr identification protocol, a ZKP-based authentication scheme that relies on modular number theory. This paper further examine the Fiat-Shamir transformation, which converts the interactive Schnorr identification protocol into a noninteractive zero-knowledge proof (NIZK), removing the need for real-time interaction between prover and verifier. This paper's analysis indicates that ZKP-based authentication is a mathematically sound and computationally possible alternative to conventional password-based systems, provided that implementation parameters such as randomness and prime size are chosen correctly.**

***Keywords*— ZKP, Schnorr Identification Protocol, Fiat-Shamir Transformation, NIZK, Number Theory, Modular**

I. Introduction

Password-based authentication is subject to a range of well-documented weaknesses. According to Verizon's 2025 Data Breach Investigations Report, compromised credentials served as the initial access vector in 22% of analyzed breaches [1], further reminding how easily passwords can be stolen, guessed, or reused across services. These findings motivate interest in authentication schemes that do not require a verifier to store or transmit a reusable secret in the first place. Zero-Knowledge Proof (ZKP) addresses this vulnerability by allowing a prover to convince a verifier that they possess certain knowledge without disclosing any parts of that knowledge itself. This paper analyzes the security of ZKP and NIZK-based authentication, with particular attention to the Schnorr identification protocol and its noninteractive Fiat-Shamir variant.

II. Theoretical Foundations

A. Modular Arithmetic

Modulo (shorten as mod) is a mathematical operation that finds the remainder after dividing one integer with another. Congruence describes an equality between two integers that yields the same remainder when divided by the same integer. If integer a and b yields the same remainder when divided by m, a is said to be congruent $b \, mod \, m$, denoted with a triple-bar symbol:

$$a \equiv b \, (mod \, m) \tag{1}$$

For example, $26 \equiv 5 \, (mod \, 7)$ because when dividing 26 with 7 it yields 3 with a remainder of 5, same with 5, when divided by 7 it yields 0 with a remainder of 5.

The congruence relation in modular arithmetic has several algebraic properties that preserve its equality across modular operations. For any given integer c, if $a \equiv b \, (mod \, m)$, then:

1. $(a + c) \equiv (b + c) \, (mod \, m)$
2. $ac \equiv bc \, (mod \, m)$
3. $a^c \equiv b^c \, (mod \, m)$

The multiplicative group of integers modulo p, denoted as $\mathbb{Z}_p^*$, is a group consisting of the sequential integers $\{1, 2, \dots, p-1\}$. Because p is a prime number, every element within this set is coprime to p, which guarantees that each integer in the group has a unique multiplicative inverse modulo p.

B. Prime Numbers & Fermat's Little Theorem

Prime numbers are defined as an integer greater than 1 whose only positive divisors are 1 and itself, a prime number guarantees that all preceding positive integers are inherently coprime to it.

Fermat's Little Theorem states that if p is a prime number and a is an integer such that the greatest common divisor of a and p is 1, then $a^{p-1} \equiv 1 \, (mod \, p)$. For example, given $a = 5$ and $p = 11$, then we have $5^{10} \equiv 1 \, (mod \, 11)$.

C. Cyclic Groups & Generators

A mathematical group $(G,\cdot)$ is a set of elements combined with a binary operation $(\cdot)$ that satisfies four fundamental axioms:

1. Closure
2. Associativity
3. Identity element
4. Inverse elements

A group is classified as cyclic if there exists at least one element g in G, known as a generator, such that every element in the group can be expressed as a power of g. In cryptographic applications such as the Schnorr identification protocol, computations are carried within a cyclic subgroup of prime order q nested within $\mathbb{Z}_p^*$, rather than within the full group. The decision to make the subgroup order a prime number q is done for security reasons. If q were not prime, the DLP could be split into several smaller, easier problems, one for each prime factor of q, making the protocol much weaker. We can verify this cyclic structure with the following example: given $g = 2$, $q = 11$ and $p = 23$, the property $\mathrm{g^q} \equiv 1 \pmod{\mathrm{p}}$ holds, since $2^{11}\ (mod\ 23) = 2048\ (mod\ 23) = 1$. This confirms that the element g generates a subgroup of order q within the group of $\mathbb{Z}_{23}^*$.

D. Discrete Logarithm Problem (DLP)

Given a generator g, a prime p and a public key y, the Discrete Logarithm Problem requires an attacker to determine x such that $g^x \equiv y\ (mod\ p)$. The security of this system rests on a computational asymmetry: evaluating $g^x\ (mod\ p)$ in the forward direction is computationally easy, whereas recovering x from y is computationally difficult for well-chosen parameters.

This asymmetry is very crucial for zero-knowledge proof-based authentication to work as intended. When prime number p is very large, there are too many possibilities for an algorithm to guess x in a reasonable time. Because the sequence of values produced by modular exponentiation appears pseudo-randomly, an attacker is forced to guess without any indicator. By basing the protocol's security to this problem, we can reasonably ensure that a prover can proof they know x without ever revealing it. Any attempt to calculate x from y will almost always fail because calculating it is mathematically improbable.

III. Zero-Knowledge Proof

An interactive proof system is a protocol in which a prover attempts to convince a verifier that a statement is true through a two-way exchange of messages. In order to determine if an interactive proof is a valid zero-knowledge proof, it must satisfy the following properties:

1. Completeness: Assuming the statement is true and both parties are honest (follows protocol), the verifier accepts a valid proof. This ensures that any legitimate prover who actually holds the correct statement will always be verified by the verifier.
2. Soundness: Assuming the statement is false, no prover can convince the verifier to accept an invalid proof within a reasonable timeframe. This secures the system against identity theft, a dishonest prover who does not know the correct statement has an improbable chance of guessing correctly.
3. Zero-Knowledge: The verifier gains no new information beyond what the prover provides. This guarantees that even if a verifier tries to cheat, they cannot reverse-engineer the correct statement.

IV. Schnorr Identification Protocol

The Schnorr identification protocol is an interactive protocol that allows a prover (A) to prove their identity to a verifier (B) without revealing their private key. The protocol is structured into a couple of steps:

A. Setup

Before the protocol begins, a set of global cryptographic parameters must be established:

1) A prime number p is chosen such that another prime number q divides $p - 1$.

2) A generator (g) of a cyclic subgroup of order q mod p.

3) A random integer chosen by A, such that $0 < x < q$.

4) The public identity of A, computed as $y = g^x\ (mod\ p)$.

B. Steps

The interactive proof runs in three distinct phases: commitment, challenge, and response.

1) A picks a random integer r, such that $0 < r < q$ and computes the value $R = g^r (mod\ p)$, which is sent to B.

2) B generates a random integer challenge c and sends it to A.

3) A calculates the value $s = r + c \cdot x\ (mod\ q)$ and sends it to B.

C. Verification

When B receives the response s, they check if the equation $g^s \equiv R \cdot y^c\ (mod\ p)$ holds true. If the equation is proven to be true, B knows with certainty that A possess the correct private key (x). For better clarity, the communication flow and execution steps of the Schnorr identification protocol are visualized in the following diagram:

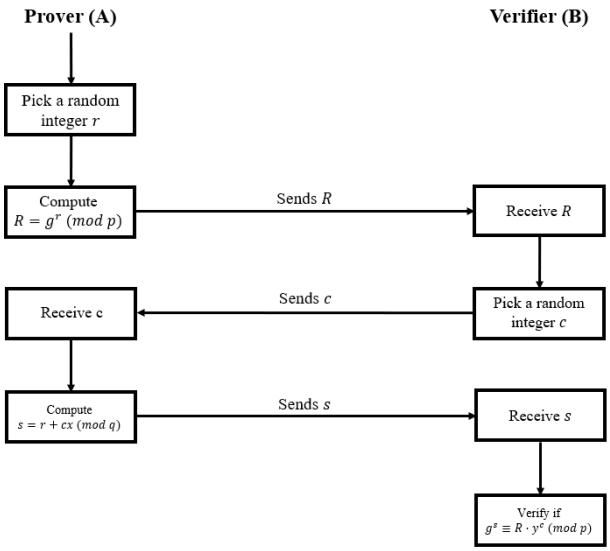

Fig. 4.1 Interactive Schnorr Identification Protocol sequence flow.

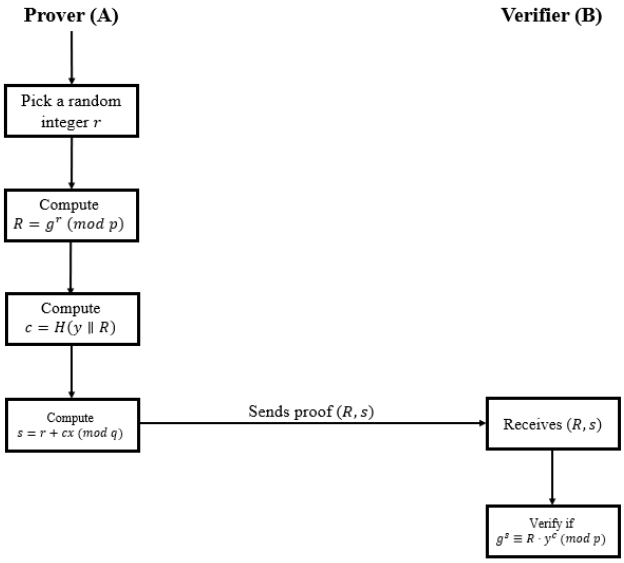

Fig. 4.2 Noninteractive Schnorr Identification Protocol using Fiat-Shamir Transformation sequence flow.

D. Example

To verify the protocol with a small test case, let's use the parameter: $p = 23$, $q = 11$, $g = 2$, with private key $x = 3$, random integer $r = 7$, and challenge integer $c = 4$.

1) We get that the public key $y = 2^3 \ (mod\ 23) = 8$.

2) The commitment $R = 2^7 \ (mod\ 23) = 128 \ (mod\ 23) = 13$.

3) A computes the response: $s = 7 + 4 \cdot 3 \ (mod\ 11) = 19 \ (mod\ 11) = 8$.

4) B verifies both sides of the equation modulo 23.

a) Left side: $2^8 \ (mod\ 23) = 256 \ (mod\ 23) = 3$.

b) Right side: $13 \cdot 8^4 \ (mod\ 23) = 53{,}248 \ (mod\ 23) = 3$.

Since both sides are equal to 3, the proof is verified.

E. Fiat-Shamir Transformation

The Schnorr identification protocol requires interactive back-and-forth communication. To convert this into a Noninteractive Zero-Knowledge (NIZK) proof, we apply the Fiat-Shamir Transformation.

Instead of waiting for B to pick a random challenge, A generates the challenge independently using a secure hash function (H). The challenge is calculated as:

$$c = H(y \parallel R) \tag{2}$$

By replacing the human verifier's randomness with the output of a hash function, A can compute the entire proof transcript (R, s) independently. Anyone can later verify this proof, eliminating the need for real-time interactions. To illustrate how Fiat-Shamir Transformation eliminates real-time interaction, the following modified diagram is shown below.

V. Experiment and Implementation

A. Implementation Setup

To validate the theoretical properties of the Schnorr identification protocol, three experiments were implemented in Python. The first experiment simulates an honest prover scenario using small, manually verifiable parameters to confirm that a legitimate prover always passes verification. The second experiment simulates a dishonest prover across 1,000 attempts to measure the soundness bound. The third experiment benchmarks verification time across three parameter sets of increasing prime size to proof the importance of large primes. Additionally, a fourth demonstration implements the Fiat-Shamir Transformation.

B. Honest and Dishonest Prover

```python
def schnorr_honest(p, q, g, x, r, c):
    y = pow(g, x, p)
    R = pow(g, r, p)
    s = (r + c * x) % q

    lhs = pow(g, s, p)
    rhs = (R * pow(y, c, p)) % p

    return lhs == rhs
```

Fig. 5.1 Implementation of an honest prover in Python.

The honest prover simulation uses the parameters $p = 23$, $q = 11$, $g = 2$, with private key $x = 3$, random integer $r = 7$, and verifier challenge $c = 4$ as implemented in the code snippet shown in Fig. 5.2. Following the protocol, the prover computes the public key $y = g^x \ (mod\ p) = 8$, commitment $R = g^r \ (mod\ p) = 13$, and response $s = (r + cx) \ (mod\ q) = 8$. The verifier then evaluates both sides of the verification equation: the left-hand side yields $g^s \ (mod\ p) =$

$2^8 \ (mod\ 23) = 3$, and the right-hand side yields $R \cdot y^c \ (mod\ p) = 13 \cdot 8^4 \ (mod\ 23) = 3$. Since both sides are equal, the verification passes, confirming the completeness property of the protocol.

```
# Honest prover test case
p, q, g = 23, 11, 2
x, r, c = 3, 7, 4
status = schnorr_honest(p, q, g, x, r, c)
```

Fig. 5.2 Parameters for the honest prover simulation.

```
.../Makalah > python3 simulation.py
Honest status: True
```

Fig. 5.3 Terminal execution and output for the honest prover test case.

```
def schnorr_dishonest(p, q, g, y, c):
    s_fake = random.randrange(1, q)

    y_inv_c = pow(pow(y, c, p), p - 2, p)
    R_fake = (pow(g, s_fake, p) * y_inv_c) % p

    c_verifier = random.randrange(1, q)
    lhs = pow(g, s_fake, p)
    rhs = (R_fake * pow(y, c_verifier, p)) % p
    return lhs == rhs
```

Fig. 5.4 Implementation of a dishonest prover in Python.

For the cheating prover simulation, the attacker does not have access to the private key x. The only viable cheating strategy is to pre-select a fake response s and back-compute a commitment $R = g^s \cdot y^{-c} \ (mod\ p)$ that satisfies the equation for a chosen c. However, the verifier subsequently sends a fresh, independently random challenge c, which will almost certainly differ from the c used to construct the fake commitment.

```
# Dishonest prover test case
p, q, g = 23, 11, 2
c = 4
y = pow(g, x, p)
correct = sum(schnorr_dishonest(p, q, g, y, c) for _ in range(1000))
print(f"Dishonest success rate: {correct / 1000:.4f} (theoretical 1/q = {1 / q:.4f})")
```

Fig. 5.5 Parameters for the dishonest prover simulation.

```
.../Makalah > python3 simulation.py
Dishonest success rate: 0.0920 (theoretical 1/q ≈ 0.0909)
```

Fig. 5.6 Terminal execution and output for the dishonest prover test case.

TABLE I.
Honest vs. Dishonest Prover Results

Scenario	Parameters	Result	Observed Success Rate
Honest	$p = 23, q = 11, g = 2, x = 3, r = 7, c = 4$	PASS	100%
Dishonest	$p = 23$, $q = 11$, 1,000 trials	Mostly FAIL	9.2%

C. *Performance Benchmark Across Prime Sizes*

```
PARAMETERS = [
    ("Small", 23, 11, 2),
    ("Medium", 1009, 7, 105),
    ("Large", 7919, 107, 6451)
    ]

print("Prime Size, Avg Time (µs)\n")

for label, p, q, g in PARAMETERS:
    x = random.randrange(1, q)

    times = []
    for _ in range(1000):
        r = random.randrange(1, q)
        c = random.randrange(1, q)
        t0 = time.perf_counter()
        schnorr_honest(p, q, g, x, r, c)
        times.append((time.perf_counter() - t0) * 1_000_000)

    avg = sum(times) / len(times)
    print(f"{label:<20} {avg:.3f}")
```

Fig. 5.7 Implementation of prime sizes benchmark in Python.

To assess the computational practicality of the Schnorr identification protocol, verification time is measured across three parameter sets of increasing prime size: a small set ($p = 23$, $q = 11$), a medium set ($p = 1{,}009$, $q = 7$), and a large set ($p = 7{,}919$, $q = 107$). For each parameter set, 1,000 trials were executed with independently randomized values of x, r and c, and the average verification time per trial was recorded. As shown in Table II, verification time increases with prime size due to the greater computational cost of modular exponentiation over larger integers. However, even at the largest tested prime, the average verification time remains within the microsecond range, demonstrating that the Schnorr identification protocol is computationally practical and well-suited for deployment in real authentication systems.

TABLE II.
Verification Time vs. Prime Size

Parameter Set	p	q	Avg. Verification Time (µs)
Small	23	11	1.065
Medium	1,009	7	1.634
Large	7,919	107	2.267

D. *Noninteractive Proof using Fiat-Shamir Transformation*

```
def nizk_prove(p, q, g, x):
    r = random.randrange(1, q)
    y = pow(g, x, p)
    R = pow(g, r, p)

    c = int(hashlib.sha256(f"{y}{R}".encode()).hexdigest(), 16) % q
    s = (r + c * x) % q
    return y, R, c, s

def nizk_verify(p, q, g, y, R, c, s):
    lhs = pow(g, s, p)
    rhs = (R * pow(y, c, p)) % p
    return lhs == rhs
```

Fig. 5.8 Implementation of Noninteractive Zero Knowledge Schnorr Identification Protocol in Python.

To demonstrate the NIZK variant of the Schnorr identification protocol, the Fiat-Shamir Transformation is implemented using Python's hashlib.sha256 as the hash function H. Rather than waiting for a verifier to issue a random challenge, the prover independently computes $c = H(y \parallel R) \ mod\ q$, where $\parallel$ denotes concatenation of the string

representations of y and R. The prover then computes the response $s = (r + cx) \bmod q$ and publishes the proof transcript (R, c, s). Verification proceeds identically to the interactive case: the verifier checks whether $g^s \equiv R \cdot y^c \ (mod\ p)$ holds. As shown in Table III, the proof was successfully verified without any real-time interaction between the prover and the verifier. This result confirms that the Fiat-Shamir Transformation effectively eliminates the need for a live verifier while preserving all three zero-knowledge properties.

TABLE III.
NIZK Fiat-Shamir Proof Transcript

Field	Value	Description
p, q, g	23, 11, 2	Public parameters
x	2	Private key
y	4	Public key
R	18	Commitment
c	3	Hash-derived challenge
s	1	Response
Verified	True	LHS = RHS status

VI. Security Analysis

To evaluate the security level of the Schnorr identification protocol, we must analyze it against the three foundational properties of a zero-knowledge proof, as well as its secureness.

A. Completeness

Completeness is a fundamental property ensuring that an honest prover who actually holds the correct private key x will always successfully pass the verifier's check. This guaranteed success is rooted in the mathematical consistency of the protocol. When the verifier receives the prover's response, they evaluate the verification equation:

$$g^s \equiv R \cdot y^c \ (mod\ p) \tag{3}$$

To see why an honest prover always passes, we can break down the equation by replacing the response s with the prover's actual calculation, which is $s = r + cx$. Substituting this value into the left side of the equation yields:

$$g^s \equiv g^{r+cx} \ (mod\ p) \tag{4}$$

Using standard exponent laws, we can split the addition in the exponent into multiplication:

$$g^s \equiv g^r \cdot g^{cx} \equiv g^r \cdot (g^x)^c \ (mod\ p) \tag{5}$$

Since we know from the setup phase that $y = g^x$ and from the previous steps that $R = g^r$, we substitute these values into the equation:

$$g^s \equiv R \cdot y^c \ (mod\ p) \tag{6}$$

Because both sides of the equation are mathematically identical, the Verifier's check will yield a positive result every single time for an honest prover.

B. Soundness

Soundness ensures that a dishonest prover without the knowledge of x cannot fake a valid proof. To cheat successfully, an attacker would have to guess the verifier's challenge c before committing to R. Because c is randomly selected from a pool of size q, the probability of successfully guessing the challenge in a single round is bounded by $\left(\frac{1}{q}\right)$. If the protocol is repeated for k rounds, the probability of a cheating prover succeeding in all rounds drops to $\left(\frac{1}{q}\right)^k$. Given a large enough q, this probability becomes a challenge for the lying prover.

C. Zero-Knowledge

The protocol satisfies the zero-knowledge property assuming the honesty of verifier. The message exchange between the prover and an honest verifier leaks no information about the private key x beyond the fact that the prover knows it. This is can be proven using a simulator argument. Without knowing x, a simulator can produce a transcript (R, c, s) that is indistinguishable from a genuine protocol run by reversing the usual order of operations: it first chooses s and c at random, then computes $R = g^s \cdot y^{-c} \ (mod\ p)$. Because the simulated transcript follows the same pattern as a real one, an honest verifier cannot distinguish a simulated proof from a genuine interaction, showing that no information about x is leaked when the challenge is generated independently of R.

D. DLP Hardness

The foundational security of the Schnorr identification protocol relies directly on the Discrete Logarithm Problem (DLP). Even though the public key y and the commitment R are transmitted in plain text, extracting the private key x or the random integer r requires solving $y = g^x \ (mod\ p)$ or $R = g^r \ (mod\ p)$. As long as p and q are large enough, calculating the discrete logarithm is computationally heavy, thus preserving the protocol's integrity.

E. Vulnerabilities

While mathematically sound, the scheme is fragile if implemented incorrectly. The most critical vulnerability appears if a prover reuses the same random integer r across two different authentication sessions. If the same r is used for two different challenges (c_1 and c_2), it produces two responses:

$$s_1 = r + c_1 \cdot x \ (mod\ q) \tag{7}$$

$$s_2 = r + c_2 \cdot x \ (mod\ q) \tag{8}$$

A hacker can easily subtract the two equations to eliminate r:

$$s_1 - s_2 = x(c_1 - c_2) \ (mod\ q) \tag{9}$$

The attacker can then extract private key x with a simple division:

$$x = \frac{s_1 - s_2}{c_1 - c_2} \ (mod\ q) \quad (10)$$

This catastrophic failure is not merely theoretical. A real-world example is the 2010 Sony PlayStation 3 ECDSA signing-key recovery: Sony's implementation reused the same nonce for every signature instead of generating a fresh random value each time, which allowed researchers to recover the console's private signing key using the same algebraic technique described above [6].

Additionally, picking prime p that is too small exposes the system to various attacks. Using small prime numbers compromises the Schnorr Identification Scheme by making the underlying DLP computationally trivial to solve. In the setup phase, the prover's public identity is revealed as $y = g^x \ (mod\ p)$. If p is small, a hacker can easily bypass the protocol's mathematical barrier and extract private key x directly using brute-force method.

Furthermore, a small prime order q drastically reduces the randomness space, allowing a hacker to effortlessly brute-force the verification equation or perform an exhaustive guess of the random integer r.

VII. Conclusion

This paper analyzed the security level of the Schnorr identification protocol and its noninteractive Fiat-Shamir Transformation variant as zero-knowledge-proof-based alternatives to password authentication. This paper showed analytically that the protocol satisfies completeness, soundness and zero-knowledge. Security reduces to the hardness of the DLP within a prime-order subgroup. Our experiments confirmed the following properties: honest provers were verified with 100% success, the observed success rate of dishonest provers (9.2%) closely matched the theoretical soundness bound of 1/q (≈ 9.09%), and verification remained practical, completing within microseconds even at the largest tested parameter sizes. The Fiat-Shamir transformation was further shown to preserve these guarantees while eliminating the need for real-time prover and verifier interaction. At the same time, our analysis of the reuse of random integer and small chosen prime attacks shows that these guarantees hold only under correct implementation. Reusing a random value across sessions or choosing an insufficiently large prime p or q, can compromise the private key, as shown by real-world failures such as the Sony PlayStation 3 incident.

Attachments

Source code used for simulation in part V. Experiment and Implementation: https://github.com/revandrazm/if1220-makalah

Video explaining and demonstrating the program used for simulation: https://youtu.be/0eUzRLYhmNk

Acknowledgment

I would like to thank Allah Subhanahu wa Ta'ala for making any of this possible and both of my parents for giving me the opportunity to live this life I'm grateful for. I would also like to thank the lecturer of IF1220 Discrete Mathematics course, Prof. Dr. Ir. Rinaldi, M.T., for his guidance throughout this course.

References

[1] Verizon, "2025 Data Breach Investigations Report". 2025. https://www.verizon.com/business/resources/reports/2025-dbir-data-breach-investigations-report.pdf. Diakses pada 11 Juni 2026.

[2] R. Lavin et al., "A Survey on The Applications of Zero-Knowledge Proofs". 2024. https://arxiv.org/pdf/2408.00243. Diakses pada 11 Juni 2026.

[3] R. Munir, "Teori Bilangan (Bagian 1)". 2026. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/15-Teori-Bilangan-Bagian1-2026.pdf. Diakses pada 16 Juni 2026.

[4] DI Management Services, "The multiplicative group modulo p". https://di-mgt.com.au/multiplicative-group-mod-p.html. Diakses pada 17 Juni 2026.

[5] K. S. McCurley, "The Discrete Logarithm Problem". https://www.mccurley.org/papers/dlog.pdf. Diakses pada 17 Juni 2026.

[6] M. Schmid, "ECDSA - Application and Implementation Failures". 2015. http://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Schmid.pdf. Diakses pada 18 Juni 2026.

[7] F. Hamila et al., "Enhancing security in Fiat–Shamir transformation-based non-interactive zero-knowledge protocols for IoT authentication". 2023. https://link.springer.com/article/10.1007/s10207-023-00779-8. Diakses pada 19 Juni 2026.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026

Revandra Zacky Maharta, 13525031