International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 11-No.4 (2025) pp. 8218-8225
http://www.ijcesen.com

ISSN: 2149-9144

Research Article

The Influence of Concepts from Number Theory on the Behavior and Security of Hash Functions

Salah Adoui[1], Ayache Benhadid[2]*

[1]Batna 2 University, Faculty of Mathematics and Computer Science, Department of Mathematics, Batna, Algeria.
Email: s.adoui@univ-batna2.dz- **ORCID:** 0000-0001-7567-2100

[2] Batna 2 University, Faculty of Mathematics and Computer Science, Department of Mathematics, *Batna, Algeria.*
* **Corresponding Author Email:** a.benhadid@univ-batna2.dz - **ORCID:** 0000-0001-7567-2192

Article Info:

DOI: 10.22399/ijcesen.4043
Received : 05 August 2025
Accepted : 29 September 2025

Keywords

Number theory
Hash functions
Systems Security
Data integrity

Abstract:

This paper explores the influence of number theory concepts on the behavior and security of cryptographic hash functions. Hash functions play a critical role in modern cryptography, ensuring data integrity, authentication, and digital signatures. While they are primarily designed using principles from algebra and complexity theory, number theory significantly contributes to their construction and security analysis. Key number-theoretic conceptssuch as modular arithmetic, prime number distributions, and discrete logarithmsunderpin many hash function designs, especially in schemes that rely on structured algebraic inputs or are constructed from hard mathematical problems. We analyze how these mathematical foundations affect essential properties like collision resistance, pre-image resistance, and avalanche behavior. Additionally, we examine how number-theoretic attacks (e.g., those exploiting modular congruences or integer factorization) pose potential threats to certain classes of hash functions. The paper concludes by highlighting current research trends leveraging advanced number-theoretic techniques to enhance hash function robustness, emphasizing the ongoing interplay between pure mathematics and practical cryptographic design.

1. Introduction

Hash functions play a fundamental role in modern cryptography, ensuring data integrity, authentication, and the secure functioning of digital systems. These mathematical constructs compress input data of arbitrary length into fixed-size outputs, or "hashes", in such a way that even small changes to the input produce significantly different outputs. For a hash function to be cryptographically secure, it must satisfy several stringent properties: pre-image resistance, second pre-image
resistance, and collision resistance. Achieving these properties often relies on deep mathematical foundationsamong which 'number theory' plays a particularly crucial role. Number theory, traditionally studied for its intrinsic mathematical elegance, has found extensive applications in cryptography, particularly in the design and analysis of cryptographic algorithms. Concepts such as modular arithmetic, prime number theory, the distribution of primes, discrete logarithms, and properties of finite fields are all instrumental in shaping the theoretical underpinnings and practical construction of secure hash functions. For instance, the hardness assumptions derived from number-theoretic problems often serve as the basis for assessing the computational infeasibility of reversing or forging hash outputs. This paper explores the interplay between number-theoretic concepts and hash function design, highlighting how these mathematical principles influence the 'behavior', 'performance', and 'security' guarantees of modern hash functions. By examining both historical and contemporary approaches, we aim to illustrate how number theory not only strengthens the robustness of cryptographic hashing but also

exposes potential vulnerabilities when misapplied or inadequately understood.
Concepts from 'number theory', particularly 'prime modulus selection' and 'quadratic residues', play subtle but important roles in the design and security of 'cryptographic hash functions', especially when these functions are used within cryptographic protocols.

2. Hash functions in mathematics.

In mathematics and computer science, 'hash functions' are functions that take an input (or "message") and return a fixed-size string of bytes, typically used to index data or verify integrity. They play a fundamental role in many areas, including
cryptography, data structures (like hash tables), and error detection.

Definition: A Hash function *h* is a function from X to Y where:
***** *X* is the set of possible inputs (which may be infinite or very large),
***** *Y* is the set of possible outputs (often finite and of fixed size).

2.1 Mathematical Properties of Hash Functions.

1. Deterministic: For any input *x*, the output $h(x)$ is always the same.
2. Efficient : The function should be fast and easy to compute for any input *x*.
3. Uniformity: Outputs should be uniformly distributed across the output space *Y* to
avoid clustering.
4. Compression: Hash functions typically map large inputs (e.g., strings of arbitrary length) to small fixed-size outputs (e.g., 32 bits, 256 bits, etc.).

2.2 Cryptographic Hash Functions (Special Type)

Cryptographic hash functions are designed with added properties for security:
1. Pre-image Resistance: Given a hash value $h(x)$, it should be * computationally infeasible * to find *x*.
2. Second Pre-image Resistance: Given input *x*, it should be hard to find a different $x' \neq x$ such that $h(x) = h(x')$
3. Collision Resistance: It should be hard to find any two distinct inputs $x \neq x'$ such that $h(x) = h(x')$.

2.3 Examples of Hash Functions

a/- **Non-Cryptographic:**
* The simple Modulo Function:
$h(x) = x \bmod n$ Used in basic hash tables.

* The multiplicative Hashing:

$$h(x) = \lfloor m(xA \mod 1) \rfloor$$

Where *A* is a constant irrational number and *m* is the table size

b/ ***Cryptographic:***

** **SHA-256:** Produces a 256-bit hash. Common in Bitcoin and secure applications.*

** **MD5** (now considered insecure).*

** **SHA-1** (also broken for security purposes).*

2.4 Some applications of Hash Functions:

** Data Structures: Hash tables, maps, sets.*

** Cryptography: Digital signatures, password storage, blockchain.*

** Checksums: File integrity, error detection.*

** Randomization: Hashing to uniformly distribute values.*

3. How to construct a simple hash function

Let's walk through the concept of a 'hash function', build a simple one, and explore collisions with clear examples and visualizations. Hash function takes an input (key) and returns a fixed-size output (a number or string), often called a 'hash' or 'hash code'. In programming, it's used in 'hash tables', 'data indexing', and 'cryptography'.

Properties of a Good Hash Function

** Deterministic: Same input always gives same output.*

** Fast: Should compute quickly.*

** Uniform: Outputs should be evenly distributed.*

** Minimizes Collisions: Different inputs should (ideally) give different outputs.*

Let's build a very basic hash function that:

** Takes a 'string' as input.*

** Converts each character to its 'ASCII' value.*

** Sums the ASCII values.*

** Uses 'modulo division' to restrict output range (simulate a hash table of fixed size).*

```
def simple_hash(key, table_size):
    total = 0
    for char in key:
        total += ord(char)  # ord() gets ASCII value of the character
    return total % table_size
```

Figure 1.*Python code using modulo division.*

Let take Hash table of size 10:

```
print(simple_hash("cat", 10))  # 99 + 97 + 116 = 312 → 312 % 10 = 2
print(simple_hash("dog", 10))  # 100 + 111 + 103 = 314 → 314 % 10 = 4
print(simple_hash("act", 10))  # 97 + 99 + 116 = 312 → 312 % 10 = 2
```

Figure 2.*Python code Hash table of size 10.*

Visualizing Collisions

A 'collision' happens when 'two different inputs' produce the 'same hash value'.

From the example above:

→ '"act"'→ *hash* = 2

→ '"cat"' →*hash* = 2

This is a 'collision'.

Table 1.*Visual Table (Hash Values)*

Input	ASCII Sum	Hash (mod 10)
cat	312	2
act	312	2 (collision)
dog	314	4
god	314	4 (collision)
tac	312	2 (collision)

Improving the Hash Function:

Lets reduce collisions using 'character position weighting':

```
def better_hash(key, table_size):
    total = 0
    for i, char in enumerate(key):
        total += (i + 1) * ord(char)  # Weight based on position
    return total % table_size
```

Figure 3.*Python code using 'character position weighting'.*

Let's try again:

```
print(better_hash("cat", 10))  # (1*99) + (2*97) + (3*116) = 842 → 842 % 10 = 2
print(better_hash("act", 10))  # (1*97) + (2*99) + (3*116) = 843 → 843 % 10 = 3
```

Figure 4.Python code improved, using 'character position weighting'.

→ Now '"cat"' and '"act"' no longer collide.

Visualization: Hash Table With Collisions

Lets visualize a hash table of size 10.

Table 2. *Hash table of size 10*

Index	Keys
0	
1	
2	cat, tac, act← collisions
3	
4	dog, god← collisions
5	
6	
7	
8	
9	

Using the improved function, the keys would spread more evenly:

Table 3. S*pread keys more evenly*

Index	Keys
0	
1	
2	cat
3	act
4	dog
5	god

Summary

* A hash function maps data (like strings) to numbers.

* Simple functions can create 'many collisions'.

* 'Better hashing strategies' reduce collisions by accounting for character position, prime numbers, etc.

* Visualization helps understand and debug hash function behavior.

4. Prime Modulus Selection

Prime modulus arithmetic is foundational in many cryptographic systems. While hash functions themselves are often bitwise operations and not directly numbertheoretic, when 'hashes are used within number-theoretic constructions', primes matter a lot.

• **Modular Arithmetic in Hash-Based Cryptography**

Some 'hash functions or hash-based constructions' (e.g., Merkle-Damgrd with modular reductions, or keyed-hash constructions in protocols) use

arithmetic modulo a prime p. The prime is typically:

* 'Large', to resist brute-force attacks

* 'Safe', e.g., a *safe prime* $p = 2q + 1$ with both p and q prime

• **Collision Resistance**

When hash outputs are used 'modulo a prime', the distribution of those outputs can be more uniform, which is essential for:

* 'Reducing collisions' (i.e., two inputs yielding the same hash)

* Preventing 'modulo bias' when mapping to finite fields or rings

• **Prime Selection for Hash-to-Field**

When hash functions are used to map arbitrary data into a finite field (e.g., in elliptic curve cryptography or zk-SNARKs), the field is defined by

a prime modulus p. The prime must be carefully selected to:

* Ensure uniform distribution

* Avoid easy-to-invert mappings

* Be compatible with the field structure (e.g., for pairing-friendly curves)

What is Hash-to-Field?

In cryptology, 'hash-to-field' maps arbitrary byte strings (like a message or hash) into an element of a 'finite field'. This is a key step in:

* Hashing to elliptic curves (e.g., in BLS signatures, zk-SNARKs)

* Pairing-based cryptography

* Cryptographic commitments

The 'field' is often defined as F_p or F_p^k, where:

* p is a 'prime number' (the field characteristic)

* k is an integer (often 1 for simple fields)

Why Primes Are Essential in Hash-to-Field?

(1) Finite Fields Need Prime Characteristics:

* A finite field F_p only exists if p is a 'prime number'.

* The arithmetic rules (like existence of multiplicative inverses) only hold if the fields characteristic is prime.

(2) Uniformity and Distribution:

* Hash-to-field needs **uniform distribution** over the field.

* For this, the field size p must be chosen to make modulo operations efficient and reduce 'bias'.

* Number theory ensures minimal **modulo bias** when mapping large hashes to elements mod p.

Number-Theoretic Concepts That Influence Prime Choice:

A. 'Modular Arithmetic'

* Hash outputs are converted to integers and reduced modulo p.

* You want p to be close to a power of 2 for efficient reduction (e.g., $p \approx 2^{255}$).

B. 'Prime Density and Distribution'

* Number theory tells us how primes are distributed (e.g., 'Prime Number Theorem').

* Designers use this to find **large safe primes** near powers of 2, like:

* $p = 2^{255} - 19$ (used in Curve25519)

* $p = 2^{381} - x$ (used in BLS12-381)

C. 'Quadratic Residues and Nonresidues'

* Needed when hashing to elliptic curves, especially with map-to-curve algorithms like SWU.

* You must compute square roots modulo p, which is only possible if certain conditions (like $p \equiv 3 \bmod 4$) are met.

* Number theory provides **TonelliShanks** algorithm or conditions for easy square root calculation.

D. 'Field Extensions and Subgroup Orders'

* In pairing-based cryptography, the field might be F_p^k.

* The subgroup order should also be prime (or have prime factors) for security.

* The 'order of the fields multiplicative group', p - 1, and its factors matter:

- Prevents small-subgroup attacks.

- Ensures the security of discrete log assumptions.

E. 'Cyclic Groups'

* Many cryptographic groups are cyclic.

* For the field F_p^*, it is cyclic of order p - 1.

* Having p-1 divisible by a large prime ensures secure discrete logarithm hardness.

F. 'Montgomery-Friendly Primes'

* Primes of the form 2^k - c enable faster modular arithmetic (Montgomery reduction).

* These forms are chosen using number-theoretic constraints for:

- Efficient modular reduction

- SIMD or constant-time implementation

Examples of Number Theory in Real Systems:

- in Curve/System the Prime Used is $p = 2^{255} - 19$ and NumberTheoretic Reasoning is: Close to power of 2, fast mod reduction

- in BLS12-381 the Prime Used is $p = 2^{381} - x$ and Number-Theoretic

Reasoning is: Supports pairing-friendly curves.

- in secp256k1 (Bitcoin) the Prime Used is $p = 2^{256} - 2^{32} - 977$ and

Number-Theoretic Reasoning is: Optimized for 256-bit fields.

So, number Theory Provides:

- Existence and structure of finite fields

- Efficient algorithms (e.g., square roots mod p)

- Security guarantees (via field and group order)

- Bias minimization in hash mapping

5. Quadratic Residues

Quadratic residues are elements that are squares modulo p, and they play roles in cryptographic constructions involving:

* 'Trapdoor permutations'

* 'Obfuscation'

* 'Verifiable random functions (VRFs)'

In hash functions, this matters particularly in 'hashing into elliptic curves' or 'designing hard-to-invert functions'.

• Obfuscation of Bit Patterns

If a hash construction involves selecting elements in a group where distinguishing 'quadratic residues' from non-residues is 'hard' (e.g., under the 'Quadratic Residuosity Assumption'), it can be used to:

* 'Hide information' in a hash output

* Ensure 'semantic security' in protocols using hashes

• Hashing to Elliptic Curves

When mapping hash values to points on an elliptic curve (like in BLS signatures or zk-SNARKs), the function must solve for $y^2 = f(x) \bmod p$. This requires checking whether $f(x)$ is a 'quadratic residue mod p', and only then can y be found. This process must:

* Be efficient

* Avoid leaking information

* Maintain uniformity

• Security via Hard Problems

Some hash-like functions or randomness extractors are built using problems like:

* 'Deciding quadratic residuosity modulo a composite'

* 'Finding square roots modulo a prime'

These are believed to be 'hard', and thus offer cryptographic strength.

6. Examples in Practice

* 'RSA-based hash constructions': Use modular arithmetic with composite moduli; security depends on number-theoretic assumptions.

* 'Hash-to-curve algorithms': Use finite field arithmetic, requiring understanding of residues for correct and uniform mappings.

* 'Discrete log-based hashes' (e.g., Pedersen commitments):

Use modular groups where quadratic residues help define group structure and hard problems.

Summary

◦ **In prime Modulus** the role in Hash Functions / Protocols is to ensures uniform distribution, avoids bias, supports strong mathematical structure (e.g., fields, groups).

◦ **In quadratic Residues** the role in Hash Functions / Protocols is: Used in secure mappings (e.g., hash to curve), obfuscation, and assumptions (e.g., Quadratic

Residuosity Problem).

◦ **In Hash Security** the role in Hash Functions / Protocols is: Often underpinned by hardness assumptions from number theory (e.g., discrete log, residuosity).

◦ **In efficiency** the role in Hash Functions / Protocols is: Prime-based fields allow fast Efficiency modular reduction and efficient arithmetic in hash constructions.

7. Results and Discussions

This study explored how number-theoretic principles influence the construction, performance, and security of modern cryptographic hash functions. Specifically, we examined how concepts such as modular arithmetic, prime number theory, multiplicative groups, and discrete logarithms are embedded in or affect the behavior of widely-used hash functions like SHA-2, SHA-3, and older constructs such as MD5 or RIPEMD.

• About the use of Modular Arithmetic

Hash functions often employ modular arithmetic to maintain fixed-size outputs and to manage overflow behavior. Our analysis confirmed that:

* In SHA-2, modulo 2^{32} or 2^{64} operations are used to constrain word sizes.

* Modular addition operations help create diffusion and avalanche effects,

which are desirable for resistance against differential attacks.

* Modular arithmetic ensures uniformity and unpredictability, enhancing hash function diffusion properties.

• Prime Numbers and Field Properties

While not all hash functions directly use large primes, their internal structure or supporting cryptographic frameworks (e.g., HMAC or MerkleDamgrd constructions) may depend on prime field behavior.
* In sponge constructions (e.g., Keccak in SHA-3), while primes are not explicitly required, number-theoretic randomness is desirable in round constants.
* Some theoretical hash designs (e.g., those based on ideal lattices or elliptic curves) depend heavily on prime fields or rings.
* Primes and related structures contribute to the unpredictability and mathematical hardness assumptions underlying theoretical hash constructs.

• Discrete Logarithm and Hard Problems

While hash functions typically do not rely directly on discrete logarithm problems (unlike key exchange protocols), some proposed constructions (e.g., ZK-friendly hash functions used in SNARKs and STARKs) do.
* Pedersen hash and MiMC hash use group operations based on discrete log hardness.
* These are particularly relevant in zero-knowledge proofs and blockchain applications.
* Discrete log-based hash functions offer algebraic structure suitable for proofs but require careful parameter selection to prevent leakage or collision vulnerabilities.

• Collision and Preimage Resistance via Number Theory

The difficulty of finding collisions or preimages in secure hash functions mirrors the complexity of number-theoretic problems. For instance:
* Hash functions using multiplicative group structures benefit from the intractability of solving inverse or discrete log problems.
* However, functions with insufficient randomness in number-theoretic components (e.g., poorly chosen primes or constants) can become vulnerable.
* Number-theoretic complexity can underpin collision resistance but must be well-implemented to avoid structural weaknesses.

• Discussion

- Theoretical vs Practical Relevance

In practice, standardized hash functions such as SHA-2 and SHA-3 do not heavily rely on advanced number-theoretic problems (like discrete logs or primes), favoring bitwise operations and permutation-based designs for efficiency. However, in theoretical cryptography and privacy-focused applications (e.g., zero-knowledge protocols),

number theory plays a central role. Functions designed for these contexts (e.g., Poseidon, MiMC) leverage field arithmetic and elliptic curve groups.

-Security Implications

Understanding the number-theoretic foundation can aid in:

* 'Evaluating resistance' to algebraic attacks.
* 'Designing hash functions' that align with specific mathematical assumptions (e.g., post-quantum resistance).
* 'Recognizing vulnerabilities', such as when poor prime selection leads to reduced entropy or faster collision finding

- Efficiency Trade-offs

Number-theoretic operations (e.g., modular exponentiation) are computationally more expensive than bitwise operations. Thus, hash functions based on them are slower and less suitable for general-purpose hashing but ideal for environments that require compatibility with other algebraic protocols. The study confirms that while general-purpose hash functions (SHA-2, SHA-3) do not directly rely on deep number-theoretic problems, their security and behavior are nonetheless influenced by basic number theoryparticularly through modular arithmetic and group theory. Advanced cryptographic applications increasingly use hash functions rooted in number theory to meet specialized needs, particularly in privacy-preserving technologies.

A proper understanding of these mathematical foundations is essential for evaluating hash function security and for designing new functions that are resistant to both classical and quantum attacks.

8. Conclusions

Number theory provides essential mathematical structuressuch as modular arithmetic, prime number theory, and discrete logarithmsthat underpin the design and analysis of many cryptographic hash functions. These structures ensure deterministic but complex behaviors necessary for cryptographic properties like 'collision resistance' and 'pre-image resistance'.

Concepts like 'modular operations', 'prime fields', and 'finite cyclic groups' introduce non-linearity and unpredictability into hash function design. These properties are crucial in making it computationally infeasible to reverse-engineer inputs or find two different inputs that produce the same hash (collisions). Number-theoretic hardness assumptionssuch as the 'difficulty of factoring large primes' or solving 'discrete logarithm problems'form the security backbone of not only public-key cryptography but also influence the *design philosophy* behind robust hash functions. For example, constructions that use modular exponentiation or elliptic curves add layers of resistance against attacks like brute-force, birthday attacks, and differential cryptanalysis. Hash functions built upon number-theoretic constructs (e.g., in the 'MerkleDamgard' paradigm or sponge functions) often rely on irreducible polynomials, prime moduli, and arithmetic over finite fields, which ensures good 'diffusion' and 'avalanche effects'. This mathematical rigor makes such functions well-suited for secure applications, from digital signatures to blockchain technologies. While number theory enhances security and provides a strong theoretical foundation, practical hash functions must balance complexity with performance. Some purely number-theoretic constructions can be secure but computationally expensive, thus limiting their adoption in resource-constrained environments. With the advent of quantum computing, some number-theoretic assumptions may become vulnerable (e.g., Shors algorithm impacting discrete log and factoringbased systems). However, number theory still contributes to 'post-quantum hashbased cryptographic schemes', such as those using lattice-based or multivariate polynomial systems.

In summary, 'number theory significantly influences both the theoretical and practical dimensions of hash function design'. It provides the mathematical foundation needed for strong cryptographic guarantees while also highlighting the importance of continuously evaluating the security of these functions in the face of advancing computational capabilities.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

References

[1] Khodakhast Bibak. (2022). Quantum Key Distribution Using Universal Hash Functions over Finite Fields. Quantum Information Processing,

[2] Kapron . (2015). On an almostuniversal hash function family with applications to authentication and secrecy codes Bibak,, Srinivasan, Tth (2015)

[3] D. R. Stinson. Some Observations on the Theory of Cryptographic Hash Functions. IACR Eprint Archive

[4] Yoneyama, Wang. Security of Practical Cryptosystems Using MerkleDamgard Hash Function in the Ideal Cipher Model Naito, Ohta (2009-2011)

[5] Barthe, Berg, Grgoire, Kunz, Skoruppa. (2012). Verified Security of MerkleDamgard Backes, ZanellaBguelin (CSF 2012).

[6] Christopher Battarbee, Ramn Flores, Thomas Koberda. (2022). Postquantum hash functions using *SLn*(F*p*)-2022 Corentin Le Coz,, Delaram Kahrobaei (ePrint / arXiv)

[7] Simran Tinani. (2023). Methods for Collisions in Some Algebraic Hash Functions-2023 (preprintarXiv)

[8] Jan Buzek, Stefano Tessaro. (2024) . Collision Resistance from MultiCollision Resistance for All Constant Parameters-(CRYPTO 2024)

[9] Tomer Ashur, Al Kindi, Mohammad Mahzoun. (2023). XHash8 and XHash12: Efficient STARKfriendly Hash Functions-2023 (ePrint)

[10] Qing Zhou, Xueming Tang, Songfeng Lu. (2023). Quantuminspired Hash Function Based on Paritydependent Quantum Walks with Memory-2023, Hao Yang (arXiv)

[11] A. Lubotzky. Discrete groups, expanding graphs and invariant measures, volume 125 of Progress in Mathematics. B With an appendix by J. D. Rogawski.