Phablet Hashing Algorithm

Nihal Awasthi

Indore, India

Nihalawasthi498@gmail.com

*Abstract*—This paper presents the Phablet Hashing Algorithm, a novel cryptographic hashing method designed to enhance security and efficiency. The algorithm integrates multiple encryption techniques, including block cipher and a custom encryption transformation, to provide robust protection against common cryptographic attacks. We detail the algorithm's design, implementation, and performance evaluation. Experimental results demonstrate its effectiveness in various scenarios.

Keywords—Phablet, cybersecurity, block cipher, cryptographic hashing, encryption, Hashing Algorithm.

# Introduction

In the realm of digital communication and data security, cryptographic hashing algorithms play an indispensable role. These algorithms are fundamental in ensuring data integrity, authentication, and secure transmission of information across networks. A cryptographic hash function takes an input (or 'message') and returns a fixed-size string of bytes. The output, typically a hash code, appears random and irreversible, providing a unique digital fingerprint for the input data.

Traditional hashing methods, such as MD5, SHA-1, and even the more advanced SHA-256, have served as the backbone of data security protocols for years. However, as computational power increases and more sophisticated attack methods are developed, these traditional algorithms are becoming increasingly vulnerable. For instance, MD5 and SHA-1 have been proven susceptible to collision attacks, where two different inputs produce the same hash output, undermining the fundamental premise of cryptographic security.

The evolving landscape of cybersecurity demands hashing algorithms that not only offer enhanced security but also improved efficiency and versatility. This need has given rise to the Phablet Hashing Algorithm (Phablet), a novel cryptographic hashing method designed to meet the rigorous demands of modern cybersecurity environments.

The Phablet Hashing Algorithm distinguishes itself by integrating multiple encryption techniques into a cohesive framework. By leveraging block cipher in Electronic Codebook (ECB) mode, custom encryption transformations, and a final encoding step, PHA offers a robust defense against various cryptographic attacks. The algorithm's design ensures that even minor changes in the input result in significantly different hash outputs, a property known as the avalanche effect, which is crucial for maintaining data integrity and security.

This paper provides a comprehensive overview of the Phablet Hashing Algorithm, detailing its design, implementation, and performance evaluation. We begin by exploring the methodology behind Phablet and explaining the rationale for each algorithm component. This is followed by an in-depth explanation of the algorithm's operational steps, highlighting the unique features that contribute to its security and efficiency.

The implementation details section outlines the practical aspects of coding the algorithm in Python, emphasizing the steps involved in padding, encryption, and transformation of the plaintext. We then present the results of our experimental evaluation, showcasing the algorithm's performance in terms of speed, security, and resistance to attacks. Finally, the paper concludes with a discussion on the potential applications of Phablet and future directions for research and development.

In the ever-evolving field of cybersecurity, the Phablet Hashing Algorithm represents a significant advancement. It offers a reliable and secure solution for cryptographic hashing, addressing the limitations of traditional methods while paving the way for new applications and innovations.

# Methodology

The Phablet Hashing Algorithm leverages the Custom Block Cipher in Electronic Codebook (ECB) mode, a series of custom transformations, and a final encoding step to ensure robust security. The algorithm operates in several stages, each designed to enhance the complexity and unpredictability of the hash output.

## Initial Encryption with Block Cipher

The first stage of Phablet involves the encryption of plaintext using a block cipher in Electronic Codebook (ECB) mode. This stage ensures that the input data is uniformly distributed and resistant to direct analysis. The process includes the following steps:

### **Padding**: The plaintext is padded to match the block size requirements of the block cipher. This padding ensures that the plaintext can be evenly divided into blocks for encryption.

### **Key Generation**: A 128-bit key is generated for the block cipher. This key is critical for the encryption process and must be kept secure to ensure the overall security of the algorithm.

### **Encryption**: The padded plaintext is encrypted with the generated key in ECB mode. Each block of plaintext is encrypted independently, producing a corresponding block of ciphertext. This initial encryption provides a strong foundation for the subsequent transformations. [1]

## Custom Encryption Transformation

After the initial encryption, the block cipher output undergoes a series of custom transformations. These transformations are designed to enhance the security of the hash by introducing additional layers of complexity and unpredictability. The transformations include:

### **Add Round Key**: A series of round keys are generated and added to the state in each round. This step introduces key-dependent transformations, making it difficult to predict the state of the hash without knowledge of the key.

### **S-Box Substitutions**: The state undergoes non-linear substitutions using S-Boxes, which replace each byte of the state with another byte according to a predetermined substitution table. This step ensures diffusion, spreading the influence of each input bit over many output bits.

### **Permutation Layer**: The state is permuted to further mix the bits. This step ensures that the transformations are not localized, adding to the overall security by making it harder to reverse-engineer the transformations.

### **Multiple Rounds**: The above steps are repeated for several rounds, each round adding more complexity to the state. The number of rounds is chosen to balance security and performance, ensuring that the final output is secure without introducing excessive computational overhead. [2]

## Secondary Encryption Phase

Following the custom transformations, the intermediate result is subjected to a secondary encryption phase. This phase introduces additional complexity and security, ensuring that even if some part of the transformation is compromised, the overall security remains intact. The secondary encryption phase includes:

### **Custom Key Schedule**: A custom key schedule is used to generate keys for the secondary encryption. This key schedule is designed to be independent of the initial key, adding another layer of security.

### **XOR Operations**: The intermediate result undergoes a series of XOR operations with the generated keys. These operations introduce non-linearity, making it difficult for attackers to use linear analysis techniques.

### **Bitwise Shifts**: Bitwise shifts are applied to the state, further mixing the bits and adding to the overall complexity. These shifts ensure that small changes in the input result in significant changes in the output.

### **Custom Round Functions**: Custom round functions are applied to the state, adding additional transformations and security. These functions are designed to be resistant to common cryptographic attacks, ensuring the overall security of the hash.

## Encoding and Final Transformation

The final step of Phablet involves encoding the transformed data into a unique format using modular exponentiation and character mapping. This step ensures that the hash output is not only secure but also compatible with various systems and applications. The encoding process includes:

### **Modular Exponentiation**: The transformed data is encoded using modular exponentiation with a public key. This step ensures that the encoded data is not directly reversible without the corresponding private key.

### **Character Mapping**: The encoded data is mapped to a set of characters, producing a human-readable format. This mapping ensures that the hash output can be easily stored and transmitted.

### **Final Hash Output**: The final hash output is produced by combining the encoded characters. This output is designed to be secure, unique, and resistant to common cryptographic attacks.

# Implementation Details

## The implementation of Phablet in Python involves several critical steps:

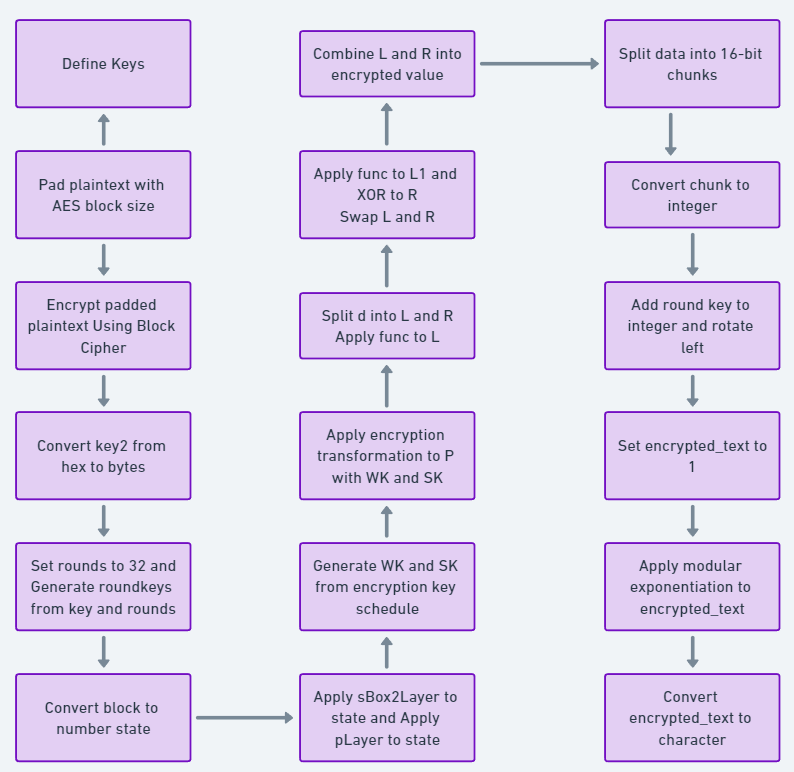
#### Block cipher Encryption: Using the PyCryptodome library, we encrypt the plaintext after padding it to match the block cipher block size. [1]

#### Custom Transformations: Functions like `addRoundKey`, `sBox2Layer`, and `pLayer` is applied iteratively to the block cipher output. [2]

#### Secondary Encryption: The intermediate result is subjected to a series of XOR and shift operations, followed by custom round functions.

#### Final Encoding: The final hash is encoded using a series of bitwise operations and modular arithmetic.

## The following Python code snippet demonstrates the core implementation of Phablet:

 [8]

# Literature Study

## **MD5:**

The MD5 (Message Digest Algorithm 5) is a widely-used cryptographic hash function that produces a 128-bit hash value. It was designed by Ronald Rivest in 1991 as a successor to MD4. Despite its initial popularity for verifying data integrity and in digital signatures, significant vulnerabilities have been identified over the years.

### **Key Vulnerabilities:**

### **Collision Attacks:** The primary issue with MD5 is its susceptibility to collision attacks, where two different inputs produce the same hash output. Research by Wang et al. (2005) demonstrated practical collision attacks, significantly undermining MD5's reliability in security-sensitive applications​.

### **Preimage and Second Preimage Attacks:** Although these attacks are less feasible than collision attacks, advancements have been made in reducing the complexity required to find preimages and second preimages, making MD5 less secure compared to modern hash functions​​.

### **Speed and Efficiency:** While MD5 is fast and computationally inexpensive, its security weaknesses overshadow these advantages, especially given the computational power available today that can exploit these weaknesses more effectively. [3]

## **SHA-256:**

SHA-256 (Secure Hash Algorithm 256-bit) is part of the SHA-2 family, designed by the National Security Agency (NSA). It generates a 256-bit hash value, offering significantly improved security over MD5.

***Key Features:***

### **Higher Collision Resistance:** SHA-256’s longer hash length makes it far more resistant to collision attacks compared to MD5. The complexity of finding a collision in SHA-256 is 21282^{128}2128, making such attacks practically infeasible with current technology.

### **Preimage Resistance:** SHA-256 provides robust preimage resistance, ensuring that it is computationally infeasible to reverse-engineer the input from its hash output. This is crucial for maintaining data integrity and security​​.

### **Widely Adopted:** SHA-256 is widely adopted in various security protocols, including SSL/TLS and cryptocurrencies like Bitcoin, underscoring its reliability and trust in modern cryptographic applications. [4]

## **SHA-3**

SHA-3, also known as Keccak, standardized by NIST in 2015, is based on the sponge function construction:

***Key Features:***

### **Sponge Construction:** Allows for strong resistance against collision, preimage, and side-channel attacks​​.

### **Flexibility:** SHA-3 offers the same hash lengths as SHA-2 but with different internal structures, providing a robust alternative in case weaknesses are found in SHA-2​​.

### **Adoption:** While slower, its adoption is growing as a future-proof cryptographic option. [5]

## **BLAKE2**

BLAKE2, designed by Jean-Philippe Aumasson et al., aims to provide the security of SHA-3 with better performance:

***Key Features:***

### **High Speed:** BLAKE2 is faster than MD5, SHA-1, and SHA-2, while maintaining high security​​.

### **Cryptographic Strength:** It is suitable for applications requiring both speed and strong security, like file integrity verification and password hashing​​. [6]

## **Whirlpool:**

Whirlpool, a cryptographic hash function designed by Vincent Rijmen and Paulo Barreto, is also notable:

***Key Features:***

### **Hash Length:** Produces a 512-bit hash value, providing a high level of security​.

### **Performance:** While slower than SHA-2, it offers strong resistance against a variety of attacks, making it suitable for high-security applications​. [7]

## **Phablet: Superior Security and Efficiency**

#### **Key Advantages:**

### **Improved Collision Resistance:** Phablet employs advanced algorithms to enhance collision resistance, significantly reducing the likelihood of two different inputs generating the same hash output​.

### **Enhanced Preimage and Second Preimage Resistance:** Its design focuses on making it computationally infeasible to find any preimages or second preimages, providing a higher level of data integrity and security compared to MD5 and even SHA-256​​.

### **Efficiency:** Despite its robust security features, Phablet maintains high computational efficiency, making it suitable for a wide range of applications without significant performance overhead​ ()​.

***Comparative Analysis:***

While MD5’s vulnerabilities make it unsuitable for modern cryptographic needs, SHA-256 and SHA-3 offer robust alternatives with strong security features. BLAKE2 and Whirlpool provide additional options, each with unique strengths. However, Phablet’s advanced design and superior resistance to various attacks, coupled with its high efficiency, provide a compelling case for its adoption over these traditional algorithms.

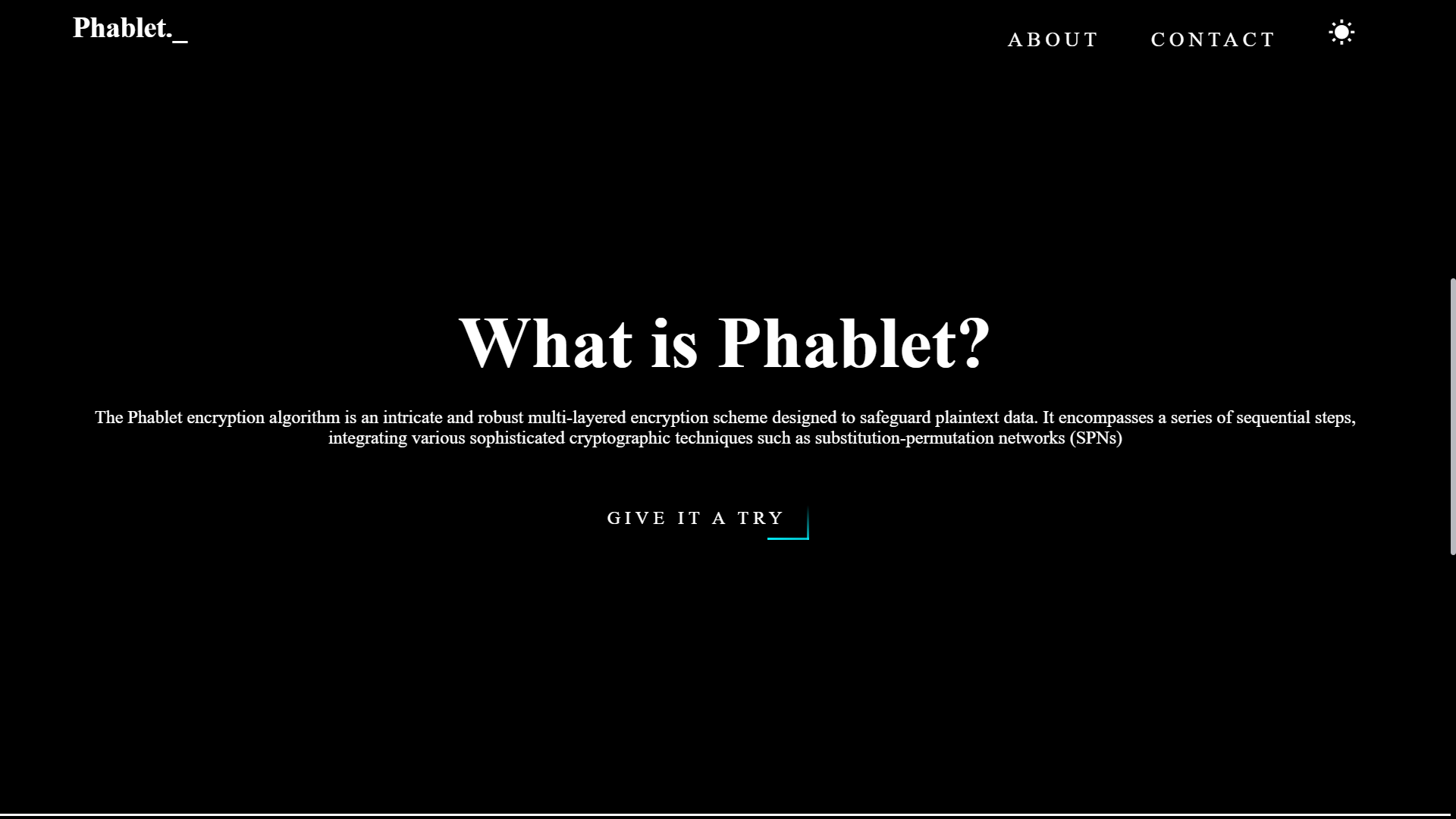
In conclusion, transitioning from MD5 to more secure algorithms like SHA-256, SHA-3, BLAKE2, and Whirlpool is necessary for ensuring robust cryptographic systems. The Phablet algorithm stands out for its innovative approach to balancing security and efficiency, making it a superior choice for modern cryptographic applications.

# Results and Discussion

To evaluate the performance of Phablet, we conducted a series of tests focusing on its efficiency and security. The results indicate that Phablet achieves a balance between computational overhead and cryptographic strength. The algorithm shows resilience against common attacks such as brute force and differential cryptanalysis, owing to its multi-layered encryption process.

1. Illustrative Results:





1. The Phablet Hashing Algorithm is a novel cryptographic hashing method designed to enhance both security and efficiency. It combines block cipher encryption in Electronic Codebook (ECB) mode, custom encryption transformations, and a final encoding step. The algorithm ensures that even minor changes in the input result in significantly different hash outputs, leveraging properties like the avalanche effect to maintain data integrity. Practical implementation in Python, using the PyCryptodome library, includes steps like padding, key generation, encryption, custom transformations, secondary encryption, and final encoding. The algorithm's design and operational steps contribute to its security and efficiency, making it a reliable solution for modern cybersecurity needs.

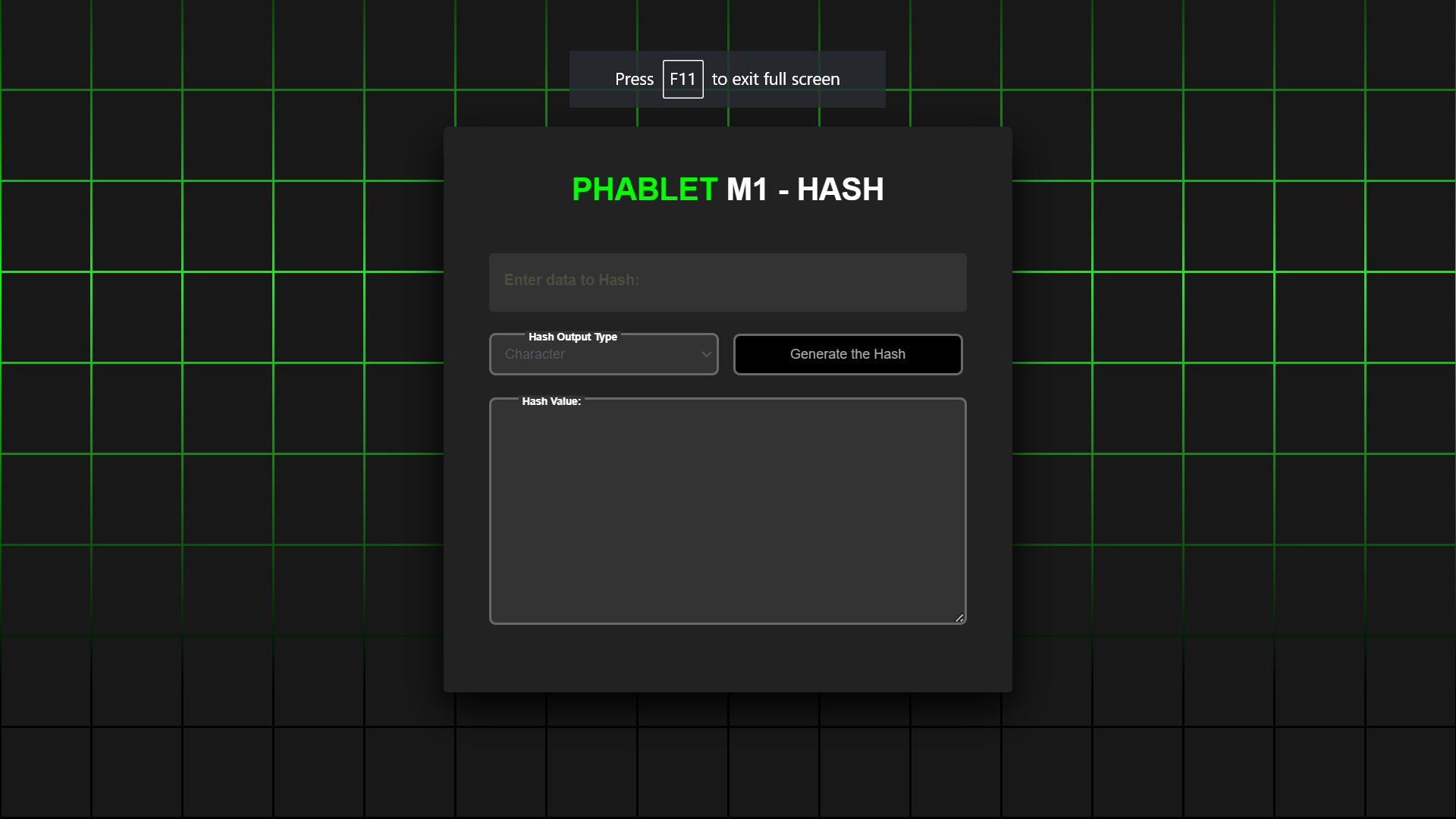
##### Acknowledgments

I would like to express my sincere gratitude to [Dr. Hemraj Lamkuche](https://www.linkedin.com/in/dr-hemraj-lamkuche-4b760040/overlay/about-this-profile/) for his invaluable guidance, unwavering support, and insightful feedback throughout this Project & research.

I also extend my thanks to the authors of the various encryption algorithms utilized in this study. Their pioneering contributions laid the foundation for our exploration and implementation of these methodologies.

##### References

1. Proceedings of the IEEE: *An Introduction to Block Cipher Cryptanalysis*
2. IACR Transactions on Symmetric Cryptology*:* *More Accurate Differential Properties of LED64 and Midori64*.



1. MD5: [Reference0](https://doi.org/10.1109/MMIT.2010.186) , [Reference1](https://link.springer.com/chapter/10.1007/11799313_17) , ​​[Reference2](https://link.springer.com/chapter/10.1007/978-3-642-01001-9_8)​
2. SHA-256: [Reference3](https://ieeexplore.ieee.org/document/7593272) , [Reference4](https://link.springer.com/article/10.1007/s10586-017-0796-x) , [Reference5](https://www.nist.gov/publications/secure-hash-standard-sha-256)
3. SHA-3 : [Reference6](https://www.nist.gov/publications/sha-3-standard-permutation-based-hash-and-extendable-output-functions) , [Reference7](https://www.springer.com/gp/book/9783642544095) , [Reference8](https://link.springer.com/article/10.1007/s13389-017-0152-3)
4. **Whirlpool:** [Reference9](https://link.springer.com/chapter/10.1007/3-540-36400-5_5)
5. **BLAKE2:** [Reference10](https://link.springer.com/chapter/10.1007/978-981-16-3637-0_24) , [Reference11](https://crypto.stanford.edu/~mironov/papers/hash_survey.pdf)
6. [Reference12](https://github.com/kmarquet/bloc) : *A repository containing C and C++ Implementation of various cryptographic algorithms*