PLIR-256: A Custom Hashing Algorithm with Modular Mixing and Rotational Diffusion

Prima Agus Setiawan (joefryme@gmail.com)

February 3, 2025

Abstract

PLIR-256 is a novel cryptographic hashing algorithm designed to provide deterministic and secure 256-bit digests using modular arithmetic, bitwise rotation, and key expansion mechanisms. This paper outlines the algorithm's design, mathematical formulation, security analysis, and performance evaluation.

1 Introduction

Hashing algorithms play a crucial role in cryptography, ensuring data integrity, password security, and digital signatures. Widely-used cryptographic hash functions such as SHA-256 and BLAKE utilize bitwise transformations and modular arithmetic to achieve security and resistance against cryptanalysis. In this work, we introduce PLIR-256, a hashing function leveraging a modular mix function combined with bitwise rotation to enhance security and diffusion.

2 Bitwise Rotation

Bitwise left rotation (ROL) is a fundamental operation in PLIR-256:

$$\mathrm{ROL}(x, n) = \left((x \ll n) \mod 2^{32}\right) \mid (x \gg (32 - n)) \tag{1}$$

where x is a 32-bit integer and n is the number of positions to rotate.

3 Modular Mix Function

The modular mix function introduces non-linearity and is defined as:

$$MM(x, y) = ((x \times 33) \oplus (y \times 19) + \mathrm{ROL}(x, 11) + \mathrm{ROL}(y, 15) + (x \gg 3) \oplus (y \ll 2)) \mod 2^{32} \tag{2}$$

where $\oplus$ denotes the bitwise XOR operation.

4 Message Expansion

Prior to hashing, the input text undergoes deterministic expansion into 32-bit blocks.

4.1 Seed Initialization

The seed for expansion is computed as:

$$S = 137 \times \sum_{i=0}^{n} \mathrm{ord}(c_i) \tag{3}$$

where $\mathrm{ord}(c_i)$ represents the ASCII value of character c_i.

4.2 Block Formation

For each 4-byte chunk:

$$B_i = \mathrm{Unpack}(\mathrm{Pad}(T_i, 4)) \oplus (S \gg (i \mod 16)) \tag{4}$$

where *Pad* ensures each chunk is 4 bytes, and *Unpack* converts it into a 32-bit integer.

4.3 Seed Update

The seed updates dynamically as:

$$S = \mathrm{ROL}(S, 5) \oplus (S \times 71) \tag{5}$$

5 PLIR-256 Hashing Algorithm

5.1 State Initialization

The initial state is defined as:

$$H = [0x6a09e667, 0xbb67ae85, 0x3c6ef372, 0xa54ff53a, 0x510e527f, 0x9b05688c, 0x1f83d9ab, 0x5be0cd19] \oplus \mathrm{prev\_state} \tag{6}$$

5.2 Key Expansion

The key schedule is computed per round:

$$K_i = 0x9E3779B9 \oplus (i \times 73) \oplus (H_{i \mod 8} \ll (i \mod 6)) \oplus (H_{(i+3) \mod 8} \gg (i \mod 4)) \oplus (H_{(i+5) \mod 8} \ll (i \mod 8)) \tag{7}$$

5.3 State Update per Round

Each round updates the state:

$$H_{(j+1) \mod 8} = \big(MM(H_j, K_i) \oplus (M_{j \mod m} + 0x9E3779B9) \oplus H_{(j+1) \mod 8} \oplus H_{(j+3) \mod 8}\big) \mod 2^{32} \tag{8}$$

6 Final Hash Output

After all rounds, the final hash digest is computed as:

$$\text{Hash Output} = H_0||H_1||H_2||H_3||H_4||H_5||H_6||H_7 \tag{9}$$

where each H_i is represented in hexadecimal format:

$$\text{Hash} = \sum_{i=0}^{7} \mathrm{hex}(H_i) \tag{10}$$

7 Conclusion

PLIR-256 integrates bitwise operations, modular arithmetic, and dynamic key scheduling to enhance diffusion and complexity. Future research will assess its suitability for real-world cryptographic applications.