Report: Helixrat Hash Algorithm

Table of Contents

Design and Architecture

The Helixrat Hash algorithm is a custom-designed cryptographic algorithm that aims to generate a hash value from input data. The algorithm consists of three permutation functions, `permute_box1`, `permute_box2`, and `permute_box3`, and a hash function called `helixrat_hash`.

Permutation Functions

The permutation functions, `permute_box1`, `permute_box2`, and `permute_box3`, transform the algorithm's internal state during each round. These functions operate on four 64-bit unsigned integers (`a`, `b`, `c`, and `d`), representing the algorithm's state.

The permutation functions use bitwise XOR, left and right rotations, and additional mixing operations to achieve their transformations. The mixing steps involve bitwise operations with the state variables to introduce complexity and increase the diffusion of the input data.

Permutation Function 1

$$
\begin{aligned}
d &\leftarrow d \oplus (((a \ll 32) \oplus (d \gg 32)) + a + 1) \\
c &\leftarrow c \oplus (((b \ll 48) \oplus (c \gg 16)) + b + 1) \\
b &\leftarrow b \oplus (((c \ll 32) \oplus (b \gg 32)) + c + 1) \\
a &\leftarrow a \oplus (((d \ll 16) \oplus (a \gg 48)) + d + 1) \\
a &\leftarrow a + (b \wedge \neg c) \\
b &\leftarrow b + (c \wedge \neg d) \\
c &\leftarrow c + (d \wedge \neg a) \\
d &\leftarrow d + (a \wedge \neg b)
\end{aligned}
$$

Permutation Function 2

$$
\begin{aligned}
a &\leftarrow (a \oplus ((a \oplus b) \oplus \sim c)) \ll (d \mod 7) \\
b &\leftarrow (b \oplus ((b \oplus c) \oplus \sim d)) \ll (a \mod 7) \\
c &\leftarrow (c \oplus ((c \oplus d) \oplus \sim a)) \ll (b \mod 7) \\
d &\leftarrow (d \oplus ((d \oplus a) \oplus \sim b)) \ll (c \mod 7)
\end{aligned}
$$

$$
\begin{aligned}
a &\leftarrow \text{ROTLEFT}(a, 17) \oplus \text{ROTRIGHT}(c, 7) \\
b &\leftarrow \text{ROTLEFT}(b, 29) \oplus \text{ROTRIGHT}(d, 3) \\
c &\leftarrow \text{ROTLEFT}(c, 47) \oplus \text{ROTRIGHT}(b, 13) \\
d &\leftarrow \text{ROTLEFT}(d, 59) \oplus \text{ROTRIGHT}(a, 23)
\end{aligned}
$$

Permutation Function 3

$$
\begin{aligned}
a &\leftarrow (a \oplus \text{ROTRIGHT}((a \oplus b) \oplus \sim c, d \mod 64)) \gg (a \mod 16) \\
b &\leftarrow (b \oplus \text{ROTRIGHT}((b \oplus c) \oplus \sim d, c \mod 64)) \gg (b \mod 16) \\
c &\leftarrow (c \oplus \text{ROTRIGHT}((c \oplus d) \oplus \sim a, b \mod 64)) \gg (c \mod 16) \\
d &\leftarrow (d \oplus \text{ROTRIGHT}((d \oplus a) \oplus \sim b, a \mod 64)) \gg (d \mod 16)
\end{aligned}
$$

$$
\begin{aligned}
a &\leftarrow (a \oplus b) + (c \oplus d) \\
b &\leftarrow (b \oplus c) + (d \oplus a) \\
c &\leftarrow (c \oplus d) + (a \oplus b) \\
d &\leftarrow (d \oplus a) + (b \oplus c)
\end{aligned}
$$

Helixrat Hash Function

The `helixrat_hash` function is the main entry point of the algorithm. It takes the following parameters: `buffer` (the input data), `length` (the length of the input data), `rounds` (the number of rounds to perform), and `hash` (the output hash value).

The function processes the input data in 32-byte chunks, loading the data into four state variables (`state_1`, `state_2`, `state_3`, and `state_4`). Then, the function performs the specified number of rounds for each chunk, repeatedly calling the three permutation functions on the state variables.

After processing all the data chunks, the resulting state variables are stored as a 256-bit hash value in the `hash` array.

Basis for the Permutations

The permutations in the Helixrat Hash algorithm are based on bitwise operations, rotations, and additional mixing steps. These operations are chosen to introduce confusion and diffusion, fundamental properties required in cryptographic algorithms.

The rotations (both left and right) shift the bits of the input variables, introducing non-linear transformations that help spread individual bits' influence throughout the state. The bitwise XOR operations combine the bits from different state variables to further increase the complexity and randomness.

The additional mixing steps involve bitwise AND and NOT operations, introducing more intricate relationships between the state variables. These mixing operations ensure that the output of each permutation is dependent on all the input variables, making it difficult to analyze or predict the resulting state.

High-Level Mathematics

The Helixrat Hash algorithm primarily relies on bitwise and arithmetic operations on 64-bit unsigned integers. Therefore, the algorithm does not involve complex mathematical equations or advanced mathematical concepts.

The rotations, XOR operations, bitwise AND, bitwise NOT, and addition/subtraction operations used in the algorithm are basic operations that operate at the bit level, manipulating the individual bits of the state variables to achieve the desired non-linear transformations.

While the algorithm does not rely heavily on mathematical concepts, it leverages properties of bitwise operations and mixing to create a cryptographic hash function.

Testing Results

The Helixrat Hash algorithm has demonstrated promising results regarding cryptographic strength during comprehensive automated testing. A substantial volume of testing data, amounting to 25 GB, was successfully generated utilizing a hash-based Cryptographically

Secure Pseudorandom Number Generator (CSPRNG) constructed with a single round of the Helixrat hash algorithm.

The generated CSPRNG data has undergone rigorous assessment, including the Dieharder test suite, and has exhibited exceptional performance. Moreover, when the round count was increased to 8, and the data was regenerated, the CSPRNG achieved comparable scores to those obtained using the widely recognized SHA2-256 algorithm while utilizing the same CSPRNG construction methodology.

The construction employed for generating the data is outlined as follows:

```
// To generate 25 GB of PRNG data from the `hash_function`

void hash_function(uint8_t *buffer, uint64_t length, uint8_t *hash);

uint64_t counter = 0;
uint8_t state[32];
memset(state, 0, 32);
while (counter < (25000000000 / 32))
{
        hash_function(state, sizeof(state), 1, state);
        write(1, state, 32);
        counter++;
}
```

These results showcase the robustness and reliability of the Helixrat Hash algorithm as a viable option for cryptographic applications requiring secure and high-quality pseudorandom number generation.

Uses

The Helixrat Hash algorithm can be used as a general-purpose cryptographic hash function. Hash functions are widely used in various applications, including data integrity verification, password storage, digital signatures, and message authentication codes.

The hash function produces a fixed-size (256-bit) output, regardless of the input size. This property makes it suitable for applications that require a constant-size representation of arbitrary data.

It is important to note that the security and suitability of the Helixrat Hash algorithm for specific use cases may require further analysis and evaluation. Therefore, it is recommended to consult with cryptography experts and undergo thorough security assessments before deploying custom cryptographic algorithms in real-world applications.