A Toy Gimli-Style Cipher

Gemini

November 9, 2025

1 Introduction

This document describes a toy implementation of a cipher based on the Gimli permutation. It is a 12-round permutation operating on a 3x4 matrix of bytes. The implementation is written in Python and is designed for educational purposes to demonstrate the core concepts of a permutation-based cipher.

1.1 Comparison with the real Gimli

The real Gimli is a high-speed cryptographic permutation designed for a wide range of platforms. This toy version simplifies many of its aspects to make it easier to understand. The key differences are:

- **State Size:** The toy cipher uses a 96-bit state (3x4 matrix of 8-bit words), whereas the real Gimli uses a 384-bit state (3x4 matrix of 32-bit words).
- **Rounds:** The toy cipher performs 12 rounds, while the real Gimli performs 24 rounds.
- **Non-linearity:** The toy cipher uses a simple 3-bit S-box, while the real Gimli uses a more complex non-linear T-function as part of an "SP-box".

2 Design and Parameters

The cipher operates on a state organized as a 3x4 matrix of 8-bit words (bytes). The three rows are referred to as lanes x, y, and z.

```
State = | x[0] x[1] x[2] x[3] |
        | y[0] y[1] y[2] y[3] |
        | z[0] z[1] z[2] z[3] |
```

2.1 S-box

The non-linear layer is a 3-bit S-box. It is applied to each of the 8 bit-columns of the state. For each column, the bits from (x, y, z) form a 3-bit input to the S-box, and the output is written back.

- **S-box**: `[7, 4, 6, 1, 0, 5, 2, 3]`
- **Inverse S-box**: `[4, 3, 6, 7, 1, 5, 2, 0]`

2.1.1 Comparison with the real Gimli

The real Gimli does not use an S-box. Instead, it has a more complex "SP-box" that includes rotations and a 3-input non-linear T-function. The toy cipher's S-box is a simplification to introduce a non-linear layer that is easy to understand and implement.

2.2 Round Constant

A round constant is used to break symmetry.

- **Round Constant**: `0x9e`

2.2.1 Comparison with the real Gimli

In the toy cipher, the round constant is a single byte. In the real Gimli, the round constant is a 32-bit word (`0x9e377900`) and it is XORed with the round number.

3 The Round Function

The cipher consists of 12 rounds. The round number `r` goes from 12 down to 1. Each round consists of the following steps:

1. **Rotation:** The lanes are rotated.
   - Lane x is rotated left by 6 bits: `rotl(x, 6)`
   - Lane y is rotated left by 2 bits: `rotl(y, 2)`
2. **S-box Layer:** The 3-bit S-box is applied to the state.
3. **Swaps:** Depending on the round number, columns of the state are swapped.
   - If `r % 4 == 0`: A "small swap" is performed (columns 0 and 1 are swapped, and columns 2 and 3 are swapped).
   - If `r % 4 == 2`: A "big swap" is performed (columns 0 and 2 are swapped, and columns 1 and 3 are swapped).
4. **Add Round Constant:** If `r % 4 == 0`, the round constant is XORed into the first word of the x lane, along with the round number itself.

   ```
   x[0] ^= (ROUND_CONSTANT ^ r)
   ```

3.1 Comparison with the real Gimli

The round function of the real Gimli is more complex. It consists of 24 rounds and includes:

- A non-linear layer (the SP-box).
- A linear mixing layer with Small-Swaps and Big-Swaps, similar to the toy cipher, but occurring at different round intervals.
- A constant addition, where a 32-bit round constant is XORed with the round number and added to the first state word every 4th round.

4 Encryption and Decryption

Encryption consists of applying the 12 rounds to the initial state.

Decryption is the inverse process. The rounds are applied in reverse order (from 1 to 12), and each operation is replaced by its inverse:

- Remove Round Constant (XOR is its own inverse)
- Inverse Swaps (swaps are their own inverse)
- Inverse S-box Layer
- Inverse Rotation (right rotations)

5 Example

Here is an example of encrypting a simple state where only the first bit is set.

5.1 Initial State

```
x: [0x01, 0x00, 0x00, 0x00]
y: [0x00, 0x00, 0x00, 0x00]
z:. [0x00, 0x00, 0x00, 0x00]
```

5.2 Encryption

After 12 rounds of encryption, the state becomes:

```
x: [0x48, 0x22, 0x00, 0x00]
y: [0x00, 0x00, 0x00, 0x00]
z: [0x00, 0x00, 0x00, 0x00]
```

5.3 Decryption

Applying the decryption function to the encrypted state recovers the original state, verifying the correctness of the implementation.