NeoAlzette ARX-box Specification

Version 6 Algorithm Definition

No institute given.

Abstract. This document provides an unambiguous, implementation-aligned specification of the NeoAlzette 64-bit ARX-box. NeoAlzette operates on two 32-bit words and employs a structure consisting of two subrounds with ARX operations, cross-XOR-rotate mixing, and cross-branch injections. The specification includes the forward and inverse transformations, constant definitions, and the linear diffusion layers L_1 and L_2 derived from the SM4 and ZUC algorithms respectively. This corresponds to the current internal version V6 as implemented in the reference source code.

Keywords: ARX-box, symmetric cryptography, lightweight cipher, diffusion layer

Introduction

This document presents a complete specification of the NeoAlzette ARX-box, a 64-bit cryptographic primitive designed for use in symmetric cryptography constructions. NeoAlzette processes two 32-bit words (A, B) through a series of ARX (Addition-Rotation-XOR) operations, cross-branch mixing, and constant injections.

The specification defines:

- The forward transformation (encryption direction)
- The inverse transformation (decryption direction)
- All required constants and rotation distances
- The linear diffusion layers L_1 and L_2
- Dynamic diffusion mask generation functions
- Cross-branch injection procedures

Important Note: The linear functions L_1 and L_2 used in NeoAlzette are derived from established cryptographic standards:

- L_1 is the linear diffusion layer from the **SM4** block cipher algorithm
- L_2 is the linear diffusion layer from the **ZUC** stream cipher algorithm

These functions provide proven diffusion properties and have been extensively analyzed in their respective standards.

This document is a **specification only** and makes no security claims about the NeoAlzette primitive. The implementation corresponds to version V6 as referenced in the accompanying source code.

Overview

NeoAlzette is a 64-bit ARX-box operating on two 32-bit words (A, B). All arithmetic $+$ and $-$ is performed modulo 2^{32}. Bitwise operations ($\oplus$, $\wedge$, $\vee$, $\neg$) are on 32-bit words.

A single invocation of the box consists of: - two subrounds (each includes ARX mixing, cross-XOR-rotate mixing, and a cross-branch injection), - followed by a final XOR with constants.

This specification corresponds to the **current internal version V6** as implemented in the accompanying source.

Constants

Let RC[0..15] denote the array of 32-bit round constants:

```
RC = [
 0x16B2C40B, 0xC117176A, 0x0F9A2598, 0xA1563ACA,
 0x243F6A88, 0x85A308D3, 0x13198102, 0xE0370734,
 0x9E3779B9, 0x7F4A7C15, 0xF39CC060, 0x5CEDC834,
 0xB7E15162, 0x8AED2A6A, 0xBF715880, 0x9CF4F3C7
]
```

Only RC[0..11] are used by the current box core; the remaining constants are reserved.

Two fixed rotation distances are used for cross-XOR-rotate mixing:

$$R_0 = 23, \qquad R_1 = 16.$$

(These correspond to `CROSS_XOR_ROT_R0` and `CROSS_XOR_ROT_R1` in the code.)

Primitive operations

32-bit rotate

For a 32-bit word x and integer r (taken modulo 32), define:

$$\begin{aligned}\mathrm{rotl}(x, r) &= (x \ll r) \ \vee \ (x \gg (32 - r)), \\ \mathrm{rotr}(x, r) &= (x \gg r) \ \vee \ (x \ll (32 - r)).\end{aligned}$$

All shifts discard bits outside 32 bits; the OR merges the wrapped parts.

Linear diffusion layers L_1 and L_2

Define the forward linear layers:

$$\begin{aligned}L_1(x) &= x \oplus \mathrm{rotl}(x, 2) \oplus \mathrm{rotl}(x, 10) \oplus \mathrm{rotl}(x, 18) \oplus \mathrm{rotl}(x, 24), \\ L_2(x) &= x \oplus \mathrm{rotl}(x, 8) \oplus \mathrm{rotl}(x, 14) \oplus \mathrm{rotl}(x, 22) \oplus \mathrm{rotl}(x, 30).\end{aligned}$$

Note that L_1 is identical to the linear diffusion layer used in the SM4 block cipher, while L_2 corresponds to the linear function from the ZUC stream cipher.

Define the backward (inverse) linear layers exactly as in the implementation:

L_1^{-1} (code: `l1_backward`).

$$\begin{aligned}L_1^{-1}(x) = x &\oplus \mathrm{rotr}(x, 2) \oplus \mathrm{rotr}(x, 8) \oplus \mathrm{rotr}(x, 10) \oplus \mathrm{rotr}(x, 14) \\ &\oplus \mathrm{rotr}(x, 16) \oplus \mathrm{rotr}(x, 18) \oplus \mathrm{rotr}(x, 20) \oplus \mathrm{rotr}(x, 24) \oplus \mathrm{rotr}(x, 28) \oplus \mathrm{rotr}(x, 30).\end{aligned}$$

L_2^{-1} (code: `l2_backward`).

$$L_2^{-1}(x) = x \oplus \mathrm{rotr}(x,2) \oplus \mathrm{rotr}(x,4) \oplus \mathrm{rotr}(x,8) \oplus \mathrm{rotr}(x,12) \oplus \mathrm{rotr}(x,14) \oplus \mathrm{rotr}(x,16) \oplus \mathrm{rotr}(x,18) \oplus \mathrm{rotr}(x,22) \oplus \mathrm{rotr}(x,24) \oplus \mathrm{rotr}(x,30).$$

Dynamic diffusion masks

Two helper functions generate a 32-bit mask from the input word X:

Mask0 (code: `generate_dynamic_diffusion_mask0`).

$$\begin{aligned}\mathsf{mask0}(X) = {} & \mathrm{rotl}(X,2) \oplus \mathrm{rotl}(X,3) \oplus \mathrm{rotl}(X,6) \oplus \mathrm{rotl}(X,9) \\ & \oplus \mathrm{rotl}(X,10) \oplus \mathrm{rotl}(X,13) \oplus \mathrm{rotl}(X,16) \oplus \mathrm{rotl}(X,17) \\ & \oplus \mathrm{rotl}(X,20) \oplus \mathrm{rotl}(X,24) \oplus \mathrm{rotl}(X,27) \oplus \mathrm{rotl}(X,31).\end{aligned}$$

Mask1 (code: `generate_dynamic_diffusion_mask1`).

$$\begin{aligned}\mathsf{mask1}(X) = {} & \mathrm{rotr}(X,2) \oplus \mathrm{rotr}(X,3) \oplus \mathrm{rotr}(X,6) \oplus \mathrm{rotr}(X,9) \\ & \oplus \mathrm{rotr}(X,10) \oplus \mathrm{rotr}(X,13) \oplus \mathrm{rotr}(X,16) \oplus \mathrm{rotr}(X,17) \\ & \oplus \mathrm{rotr}(X,20) \oplus \mathrm{rotr}(X,24) \oplus \mathrm{rotr}(X,27) \oplus \mathrm{rotr}(X,31).\end{aligned}$$

Cross-branch injection

The primitive defines two injection functions producing a pair (c,d) from a source word and two injected constants (rc_0, rc_1). All bitwise NOT $\neg(\cdot)$ is taken over 32 bits.

Injection from B into A (code: `cd_injection_from_B`). Given B and (rc_0, rc_1):

1. Compute the boolean term:
$$s_B = (B \oplus \mathrm{RC}[2]) \;\oplus\; \neg\big(B \wedge \mathsf{mask0}(B)\big).$$
2. Set:
$$c = L_2(B), \qquad d = L_1(B) \oplus rc_0.$$
3. Let $t = c \oplus d$, then update:
$$c \leftarrow c \oplus d \oplus s_B, \qquad d \leftarrow d \oplus \mathrm{rotr}(t,16) \oplus rc_1.$$
4. Output (c,d).

Injection from A into B (code: `cd_injection_from_A`). Given A and (rc_0, rc_1):

1. Compute the boolean term:
$$s_A = (A \oplus \mathrm{RC}[7]) \;\oplus\; \neg\big(A \vee \mathsf{mask1}(A)\big).$$
2. Set:
$$c = L_1(A), \qquad d = L_2(A) \oplus rc_0.$$
3. Let $t = c \oplus d$, then update:
$$c \leftarrow c \oplus d \oplus s_A, \qquad d \leftarrow d \oplus \mathrm{rotl}(t,16) \oplus rc_1.$$
4. Output (c,d).

NeoAlzette box (forward direction)

Given input (A, B), the forward transformation is:

First subround

1. **ARX mixing on B:**

$$B \leftarrow B + \Big(\mathrm{rotl}(A, 31) \oplus \mathrm{rotl}(A, 17) \oplus \mathrm{RC}[0]\Big).$$

2. **Subtract constant on A:**

$$A \leftarrow A - \mathrm{RC}[1].$$

3. **Cross XOR/ROT mixing (fixed R_0, R_1):**

$$A \leftarrow A \oplus \mathrm{rotl}(B, R_0), \qquad B \leftarrow B \oplus \mathrm{rotl}(A, R_1).$$

4. **Injection from B into A:**
   Compute $(C_0, D_0) = \mathrm{cd\_inj\_B}\big(B,\ (\mathrm{RC}[2] \vee \mathrm{RC}[3]),\ \mathrm{RC}[3]\big)$, then:

$$A \leftarrow A \oplus \mathrm{rotl}(C_0, 24) \oplus \mathrm{rotl}(D_0, 16) \oplus \mathrm{RC}[4].$$

5. **Apply L_1^{-1} to B:**

$$B \leftarrow L_1^{-1}(B).$$

Second subround

1. **ARX mixing on A:**

$$A \leftarrow A + \Big(\mathrm{rotl}(B, 31) \oplus \mathrm{rotl}(B, 17) \oplus \mathrm{RC}[5]\Big).$$

2. **Subtract constant on B:**

$$B \leftarrow B - \mathrm{RC}[6].$$

3. **Cross XOR/ROT mixing (fixed R_0, R_1):**

$$B \leftarrow B \oplus \mathrm{rotl}(A, R_0), \qquad A \leftarrow A \oplus \mathrm{rotl}(B, R_1).$$

4. **Injection from A into B:**
   Compute $(C_1, D_1) = \mathrm{cd\_inj\_A}\big(A,\ (\mathrm{RC}[7] \wedge \mathrm{RC}[8]),\ \mathrm{RC}[8]\big)$, then:

$$B \leftarrow B \oplus \mathrm{rotl}(C_1, 24) \oplus \mathrm{rotl}(D_1, 16) \oplus \mathrm{RC}[9].$$

5. **Apply L_2^{-1} to A:**

$$A \leftarrow L_2^{-1}(A).$$

Final constant XOR

$$A \leftarrow A \oplus \mathrm{RC}[10], \qquad B \leftarrow B \oplus \mathrm{RC}[11].$$

The output (A, B) is the result of one NeoAlzette box (forward).

Inverse box (backward direction)

The inverse transformation is defined as the exact reverse of the forward steps. Equivalently, it is the procedure implemented by `NeoAlzetteCore::backward`:

1. Undo final XOR:
$$B \leftarrow B \oplus \mathrm{RC}[11], \qquad A \leftarrow A \oplus \mathrm{RC}[10].$$
2. Reverse second subround:
   (a) $A \leftarrow L_2(A)$.
   (b) Compute $(C_1, D_1) = \mathrm{cd\_inj\_A}\big(A, (\mathrm{RC}[7] \wedge \mathrm{RC}[8]), \mathrm{RC}[8]\big)$ and
$$B \leftarrow B \oplus \mathrm{rotl}(C_1, 24) \oplus \mathrm{rotl}(D_1, 16) \oplus \mathrm{RC}[9].$$
   (c) Undo cross XOR/ROT mixing:
$$A \leftarrow A \oplus \mathrm{rotl}(B, R_1), \qquad B \leftarrow B \oplus \mathrm{rotl}(A, R_0).$$
   (d) Undo subtraction and ARX addition:
$$B \leftarrow B + \mathrm{RC}[6], \qquad A \leftarrow A - \Big(\mathrm{rotl}(B, 31) \oplus \mathrm{rotl}(B, 17) \oplus \mathrm{RC}[5]\Big).$$
3. Reverse first subround:
   (a) $B \leftarrow L_1(B)$.
   (b) Compute $(C_0, D_0) = \mathrm{cd\_inj\_B}\big(B, (\mathrm{RC}[2] \vee \mathrm{RC}[3]), \mathrm{RC}[3]\big)$ and
$$A \leftarrow A \oplus \mathrm{rotl}(C_0, 24) \oplus \mathrm{rotl}(D_0, 16) \oplus \mathrm{RC}[4].$$
   (c) Undo cross XOR/ROT mixing:
$$B \leftarrow B \oplus \mathrm{rotl}(A, R_1), \qquad A \leftarrow A \oplus \mathrm{rotl}(B, R_0).$$
   (d) Undo subtraction and ARX addition:
$$A \leftarrow A + \mathrm{RC}[1], \qquad B \leftarrow B - \Big(\mathrm{rotl}(A, 31) \oplus \mathrm{rotl}(A, 17) \oplus \mathrm{RC}[0]\Big).$$

Notes

- This document specifies the **current internal version V6** of the NeoAlzette box.
- This specification intentionally contains **no** security claims.
- Test vectors / benchmarking data are not included in this document at this stage.