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, dynamic diffusion masks, and the cross-branch injection procedures. 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
- Dynamic diffusion mask generation functions
- Cross-branch injection procedures

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.

Dynamic diffusion masks

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

Mask0 (code: `generate_dynamic_diffusion_mask0`).

```
v0 = X
v1 = v0 ^ rotl(v0, 2)
v2 = v0 ^ rotl(v1, 17)
v3 = v0 ^ rotl(v2, 4)
v4 = v3 ^ rotl(v3, 24)
mask0(X) = v2 ^ rotl(v4, 7)
```

Mask1 (code: `generate_dynamic_diffusion_mask1`).

```
v0 = X
v1 = v0 ^ rotr(v0, 2)
v2 = v0 ^ rotr(v1, 17)
v3 = v0 ^ rotr(v2, 4)
v4 = v3 ^ rotr(v3, 24)
mask1(X) = v2 ^ rotr(v4, 7)
```

Mask properties (implementation note)

- Both $\mathsf{mask0}(\cdot)$ and $\mathsf{mask1}(\cdot)$ are linear mappings over $\mathbb{F}_2$ (rotations and XOR only), and are full-rank (invertible) as 32-bit linear maps.
- Witness values (used by the reference implementation): $\mathsf{mask0}(1) =$ `0xd05a0889` and $\mathsf{mask1}(1) =$ `0x2220b417`.
- Verified diffusion quality for the 5-XOR construction: the reference search/verification tools report exact differential branch number = 12 and exact linear branch number = 12 (witness input `0x00000001`).

Formal statement of the 5-XOR mask construction and branch numbers

Linear-map view. Let $\mathbb{F}_2^{32}$ denote the 32-bit vector space. The procedures $\mathsf{mask0}(\cdot)$ and $\mathsf{mask1}(\cdot)$ are $\mathbb{F}_2$-linear (XOR and rotations only), hence there exist 32×32 binary matrices M_0, M_1 such that

$$\mathsf{mask0}(x) = M_0 x, \qquad \mathsf{mask1}(x) = M_1 x, \qquad \forall x \in \mathbb{F}_2^{32}.$$

Moreover, since the construction uses only rotations and XOR, it is rotation-equivariant:

$$\mathsf{maski}(\mathrm{rotl}(x,k)) = \mathrm{rotl}(\mathsf{maski}(x),k), \quad \text{for } i \in \{0,1\},\ k \in \{0,\dots,31\},$$

therefore M_0 and M_1 are circulant linear maps. They are uniquely determined by the image of the basis vector $e_0 =$ `0x00000001`:

$$M_0 e_0 = \texttt{0xd05a0889}, \qquad M_1 e_0 = \texttt{0x2220b417}.$$

Bit-branch-number metric. For a linear map M on 32-bit words, define the (bit-level) differential branch number

$$B_\Delta(M) = \min_{x \neq 0} \big(\mathrm{wt}(x) + \mathrm{wt}(Mx)\big),$$

where $\mathrm{wt}(\cdot)$ is the Hamming weight of a 32-bit word. We also record the corresponding linear branch-number metric reported by the reference verification tools as $B_\lambda(M)$.

Verified values (exact). For the 5-XOR constructions M_0 and M_1 above, the reference search/verification tools report

$$B_\Delta(M_0) = B_\lambda(M_0) = 12, \qquad B_\Delta(M_1) = B_\lambda(M_1) = 12,$$

with witness input $x = e_0 =$ `0x00000001`.

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:
$$c = B, \qquad d = \mathsf{mask0}(B) \oplus rc_0.$$

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:
$$c = A, \qquad d = \mathsf{mask1}(A) \oplus rc_0.$$

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

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

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) Compute $(C_1, D_1) = \mathrm{cd\_inj\_A}\big(A, (\mathrm{RC}[7] \wedge \mathrm{RC}[8]), \mathrm{RC}[8]\big)$ and
   $$B \leftarrow B \oplus \mathrm{rotl}(C_1, 24) \oplus \mathrm{rotl}(D_1, 16) \oplus \mathrm{RC}[9].$$
   (b) Undo cross XOR/ROT mixing:
   $$A \leftarrow A \oplus \mathrm{rotl}(B, R_1), \qquad B \leftarrow B \oplus \mathrm{rotl}(A, R_0).$$
   (c) 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) Compute $(C_0, D_0) = \mathrm{cd\_inj\_B}\big(B, (\mathrm{RC}[2] \vee \mathrm{RC}[3]), \mathrm{RC}[3]\big)$ and
   $$A \leftarrow A \oplus \mathrm{rotl}(C_0, 24) \oplus \mathrm{rotl}(D_0, 16) \oplus \mathrm{RC}[4].$$
   (b) Undo cross XOR/ROT mixing:
   $$B \leftarrow B \oplus \mathrm{rotl}(A, R_1), \qquad A \leftarrow A \oplus \mathrm{rotl}(B, R_0).$$
   (c) 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.
- **Implementation-alignment note (removing L_1/L_2):** earlier internal drafts placed additional linear layers L_1/L_2 (and their inverses) around the injection steps. The current reference implementation removes these layers and instead reuses the dynamic diffusion masks as the linear pre-mix inside the injection procedures (see the definitions of c and d above). This eliminates the extra rotate/XOR cost and simplifies the spec while keeping the injection's intended "dynamic mask + NOT-(AND/OR)" structure.