The Algorithm OaldresPuzzle_Cryptic

Technical Details

An Innovative Lightweight Symmetric Encryption Algorithm Integrating NeoAlzette ARX S-box and XCR CSPRNG

Twilight-Dream

April 19, 2024

Abstract

This paper introduces "Little OaldresPuzzle_Cryptic," an innovative lightweight symmetric encryption algorithm.

Central to this algorithm is the integration of two main cryptographic components: the NeoAlzette permutation S-box based on ARX primitives and the novel pseudo-random number generator XorConstantRotation (XCR) used solely in the key expansion process. The NeoAlzette S-box, a non-linear function for 32-bit pairs, is meticulously designed for encryption strength and operational efficiency, ensuring robust security in resource-constrained environments. In the encryption and decryption processes, a pseudo-randomly selected mixed linear diffusion function, distinct from XCR, is applied, thereby enhancing the complexity and unpredictability of the encryption.

We thoroughly explore the various technical aspects of the Little OaldresPuzzle_Cryptic algorithm.

Its design aims to balance speed and security in the encryption process, particularly for high-speed data transmission scenarios. Recognizing that resource efficiency and execution speed are crucial for lightweight encryption algorithms, but not at the expense of security, we employed a series of statistical tests to validate the cryptographic security of our algorithm. This included assessments of resistance to linear and differential cryptanalysis, among other measures.

By combining the NeoAlzette S-box with the sophisticated key expansion using XCR, and integrating the pseudo-randomly selected mixed linear diffusion function in its encryption and decryption processes, our algorithm significantly enhances its capability to withstand advanced cryptographic analysis techniques. This is while maintaining lightweight and efficient operation. Our test results demonstrate that the Little OaldresPuzzle_Cryptic algorithm effectively supports the encryption and decryption needs of high-speed data, ensuring robust security and making it an ideal choice for various modern cryptographic application scenarios.

Contents

1 Introduction

This paper presents "Little OaldresPuzzle_Cryptic", a symmetric sequence cryptographic algorithm tailored for the demands of our digital age. As data generation and consumption rates increase, so do threats to data integrity and security. This scenario necessitates cryptographic algorithms that can guarantee secure data exchange and high-speed performance.

Our algorithm uses the same cryptographic key for both encryption and decryption, providing significant speed advantages over asymmetric alternatives. The sequence cryptographic approach adds complexity to the encryption and decryption process, making it more challenging for unauthorized entities to interpret encrypted data.

The algorithm combines speed and security by utilizing a cryptographically secure pseudo-random number generator (CSPRNG) called "XorConstantRotation" (XCR), which produces highly unpredictable number sequences.

The remainder of this paper will examine the algorithm's mechanics, demonstrating how it utilizes mathematical constants, bitwise operations, and sequence-based cryptography to ensure speed and security in data transmission.

2 XCR/Little OaldresPuzzle_Cryptic Stream Cipher Design Background:

2.1 What's ARX structure and Salsa20, Chacha20 Algorithm ?

ARX, or Addition/Bit-Rotation/Exclusive-OR, is a class of symmetric key algorithms constructed using the following simple operations: modulo addition, bit rotation, and Exclusive-OR. Unlike S-box based designs, where the only nonlinear element is the substitution box (S-box), ARX designs rely on nonlinear hybrid functions such as addition and rotation. [Ranea et al., 2022] [Liu et al., 2021] These functions are easy to implement in both software and hardware and have good diffusion and resistance to differential and linear analysis. [Fei, 2012] [Aumasson et al., 2007] There are some ECRYPT Power Points that mention this design structure in summary. ARX-based Cryptography

Salsa20 is a stream cipher algorithm proposed by Daniel J. Bernstein in 2005, which utilizes the ARX structure. Salsa20/8 and Salsa20/12 are two variants of Salsa20 that run 8 and 12 rounds of encryption respectively. [Bernstein, 2005] [Bernstein, 2008] These algorithms were evaluated in the eSTREAM project and accepted as finalists in 2008. [Tsunoo et al., 2007] Salsa20 was designed with a focus on simplicity and efficiency, and it is favored in the cryptography community for its speed and security. [Maitra et al., 2015] [Ghafoori and Miyaji, 2022]

ChaCha20 [Bernstein et al., 2008a] [Bernstein et al., 2008b] is an ARX-based high-speed stream cipher proposed by Daniel J. Bernstein in 2008 as an improved version of Salsa20 . ChaCha20 uses a 512-bit permutation function to convert a 512-bit input vector into a 512-bit output vector, and then The output vector is then added to the input vector to obtain a 512-bit keystream block. The input vector consists of four parts: a constant, a key, a counter, and a random number. The security of ChaCha20 relies on the complexity and irreversibility of the substitution function, as well as the randomness and uniqueness of the input vector. ChaCha20 has been utilized in a wide range of applications, including TLS v1.3, SSH, IPsec, WireGuard, etc. [Serrano et al., 2022] [Cai et al., 2022].

2.2 About the XCR Structure of Our Lightweight Symmetric Encryption Technology

2.2.1 Designing XCR from the ARX Structure: Insights from NIST and Lightweight Cryptography

The National Institute of Standards and Technology (NIST) has acknowledged the urgent need for lightweight encryption algorithms to address the security challenges posed by the proliferation of resource-constrained devices, especially in the context of the Internet of Things (IoT). NIST-LCS Website These devices, such as sensors and RFID tags, require encryption algorithms that are efficient and secure. This balance that traditional cryptographic methods because of their high computational overhead.

Our approach to designing the XCR structure is informed by the insights presented in the paper "State of the Art in Lightweight Symmetric Cryptography" by Biryukov and Perrin [Biryukov and Perrin, 2017]. This seminal work outlines the design constraints and trends in lightweight symmetric cryptography, emphasizing the importance of algorithms customized for specific hardware and use cases. It highlights the critical trade-offs among performance, security, and resource consumption, which are essential in the development of lightweight encryption algorithms.

The paper outlines criteria for lightweight cryptographic algorithms, especially in resource-constrained environments. It advocates for a small block size, ideally 64-bit or less, and a small key size of at least 80 bits to balance security and efficiency. The round function should be simple, relying on straightforward operations that are easy to implement on low-power devices. A simple key scheduling mechanism is also crucial to prevent vulnerabilities and reduce complexity.

The potential risks of poorly implemented lightweight cryptography are underscored by the study of "Speck-R: an ultra-lightweight encryption scheme for the Internet of Things" ([Sleem and Couturier, 2021]). This paper and the references it cites illustrate the disastrous consequences of not using lightweight encryption, which can lead to security breaches and even denial-of-service attacks on small devices. This paper serves as a cautionary tale, emphasizing the importance of meticulous design and implementation of such schemes.

Our "Little OaldresPuzzle_Cryptic" algorithm is designed with these considerations in mind, adhering to the standards set by the state-of-the-art in lightweight cryptography. It aims to provide a robust and efficient solution for secure data transmission in IoT and other resource-constrained environments.

By leveraging the ARX structure, known for its simplicity and efficiency, we have designed the XCR to produce a sequence of pseudo-random numbers that are both unpredictable and computationally intensive. This structure allows for a consistent architecture adaptable across various device types, enabling faster encryption and decryption processes.

3 XCR Algorithm Structure and Operations

The XCR structure is a novel ARX (Addition, Rotation, and Exclusive-OR) design proposed to enhance the randomness and security of Cryptographically Secure Pseudo-Random Number Generators (CSPRNGs). It comprises three primary operations: Exclusive-OR (XOR) for state mixing, constant addition using irrational numbers to introduce randomness, and bitwise rotation for spreading and scrambling of bits. These operations were selected for their simplicity, efficiency, and effectiveness in improving randomness and security without adding to algorithmic complexity.

The XOR operation combines the CSPRNG state with random data, which can be derived from key elements such as the key, nonce, or counter. The constant addition involves adding mathematical constants chosen from well-known irrational numbers, such as π or e. These numbers are preferred for their infinitely non-repeating fractional parts, providing a high-quality source of randomness. The bitwise rotation operation shifts the bits of the constant addition output by a predetermined amount, enhancing the unpredictability of the output. The XCR structure produces outputs that are highly unpredictable and non-repeating, making them suitable for use as keystreams in encryption or as tags in authentication.

3.1 Preliminaries and Notations

3.1.1 Mathematical Operators

- $a \wedge b$ and $a \vee b$: bitwise AND, bitwise OR.
- $\neg a$: bitwise NOT.
- $a \oplus b$: bitwise XOR.
- $a \ominus b$: bitwise NOT XOR.
- $a \boxplus_{64} b$ and $a \boxminus_{64} b$: modular addition and modular subtraction operator 2^{64}, that is $a + b \mod 2^{64}$ and $a - b \mod 2^{64}$.
- $a \lll_{64} r$ and $a \ggg_{64} r$: left and right bitwise rotation operator, r up to 64 - 1 bits.
- $a \ll_{64} s$ and $a \gg_{64} s$: left and right bitwise shift operator, s up to 64 - 1 bits.

In establishing the foundational elements of the XCR algorithm, we define the following:

- Let x, y, and *state* be 64-bit unsigned integers. Initially, $x = y = 0$ and *state* represents the CSPRNG's internal state.
- Define *number_once* as a 64-bit unsigned integer representing the current round number, derived from cryptographic elements like the key, nonce, and counter.
- **ROUND_CONSTANTS**: A collection of 64-bit unsigned integers forming the round constants, chosen from well-established irrational numbers for their randomness and complexity.
- Define I (Input) and O (Output) as 64-bit unsigned integers, where I is the deterministic input, and O is the stochastic output generated by the algorithm.

The output O of the XCR algorithm is computed as follows:

$$O = \mathbf{PRG}(I, number\_once) = I \oplus XCR(number\_once) = I \oplus y$$

In this expression, **PRG** denotes a Pseudo-Random Generator, and $XCR(number\_once)$ represents the transformation process of the XCR algorithm for a given round specified by *number_once*. The XOR operation $\oplus$ is applied between the input I and the transformed state y to generate the final output O.

3.2 Overall Steps

The XCR algorithm progresses through several distinct steps, each vital to its operation:

1. **Initialization**: Set state variables x, y, *state*, and *counter*.
2. **Round Constant Selection**: Choose $RC0$, $RC1$, and $RC2$ based on the iteration number and state variables.
3. **State Update**: Apply diffusion and confusion layer operations.
4. **Incrementation Of Counter**
5. **Output Generation**: Produce output from the updated y.

These steps provide a high-level view of the algorithm's process, forming the backbone upon which the detailed operations are built.

3.3 Diffusion and Confusion Layers

The core of the XCR algorithm lies in its diffusion and confusion layers, designed to enhance the unpredictability of its output.

- **Diffusion Layer:** The diffusion layer aims to distribute changes across the state variables.
- **Confusion Layer:** The confusion layer introduces complex transformations to enhance security.

3.4 Selection of Rotation Amounts

The selection of rotation amounts is crucial. The pair of 1 and 63 is used for their complementary characteristics in 64-bit operations. These rotations must satisfy the mutual (prime number/co-prime number) condition: $gcd(r_i, r_{i+1}) = 1$, $gcd(r_i, r_{i+1}, r_{i+2}) = 1$, and so forth.

3.5 Algorithm Efficiency and Security

The XCR structure's efficiency and security are rooted in its layered approach, combining diffusion and confusion techniques with strategically chosen rotation amounts and XOR operations, achieving high complexity and unpredictability.

4 Detailed Description of the XCR Algorithm Structure

This section presents an in-depth view of the XCR algorithm, delineating its structure and operational mechanics.

4.1 Specific Rotation Amounts

For the XCR algorithm, specific rotation amounts are chosen as follows:
$r1 = 7, r2 = 19, r3 = 32, r4 = 47$

Additionally, we utilize 1 and 63 as complementary rotation amounts in 64-bit operations for their obvious complementary properties.

4.2 Dynamic Round Constant Selection

$RC0 = \textbf{ROUND\_CONSTANTS}[number\_once \mod ROUND\_CONSTANTS.size()]$,
$RC1 = \textbf{ROUND\_CONSTANTS}[(counter \boxplus_{64} number\_once) \mod ROUND\_CONSTANTS.size()]$,
$RC2 = \textbf{ROUND\_CONSTANTS}[state \mod ROUND\_CONSTANTS.size()]$.

4.3 Diffusion and Confusion Layers

Diffusion Layer

$$
\begin{aligned}
\text{If } x = 0: \quad & x \leftarrow RC0, \\
\text{Else:} \quad & y \leftarrow y \oplus ((x \lll_{64} r2) \oplus (x \lll_{64} r3)), \\
& state \leftarrow state \oplus ((y \lll_{64} r3) \oplus (y \lll_{64} r4) \oplus (y \lll_{64} 63)) \oplus counter, \\
& x \leftarrow x \oplus ((state \lll_{64} r1) \oplus (state \lll_{64} r2)) \oplus RC0 \oplus number\_once.
\end{aligned}
$$

Confusion Layer

$$
\begin{aligned}
state \leftarrow & \; state \boxplus_{64} (y \oplus (y \ggg_{64} 1) \oplus RC0), \\
x \leftarrow & \; x \oplus (state \boxplus_{64} (state \ggg_{64} 1) \boxplus_{64} RC1), \\
y \leftarrow & \; y \boxplus_{64} (x \oplus (x \ggg_{64} 1) \oplus RC2).
\end{aligned}
$$

4.4 Incrementation of Counter

$$counter \leftarrow counter + 1.$$

5 NeoAlzette Substitution Box

The NeoAlzette Substitution Box (S-box), a core component of the Little OaldresPuzzle_Cryptic algorithm, marks a significant advancement in cryptographic design over its predecessor, the Alzette S-box referenced in [Beierle et al., 2019]. This novel S-box is ingeniously crafted based on ARX (Add-Rotate-XOR) primitives and tailored for 32-bit pair operations. A key distinction of the NeoAlzette S-box lies in its enhanced structure, which offers substantial improvements in passing rigorous statistical tests, a crucial benchmark for evaluating the strength and reliability of cryptographic algorithms.

In contrast, when the Alzette S-box was subjected to similar statistical evaluations, it failed to meet the same stringent criteria. This stark difference underscores the superiority of the NeoAlzette design in terms of non-linearity and its ability to thwart patterns that are vulnerable to cryptographic attacks. By integrating this innovative S-box into the Little OaldresPuzzle_Cryptic algorithm, we have fortified the encryption process, ensuring a higher degree of security and robustness against advanced cryptographic analyses.

5.1 NeoAlzette Used Unique ROUND_CONSTANTS

The **NeoAlzette ROUND_CONSTANTS** array is a crucial part of the NeoAlzette S-box, used in both the encryption and decryption processes. It is composed of predefined constants, including a concatenation of Fibonacci numbers and specific mathematical constants:

```
    constexpr std::array<std::uint32_t, 16> ROUND_CONSTANT
    {
        //1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,4181 (Fibonacci numbers)
        //Concatenation of Fibonacci numbers : 1235813213455891442333776109871597258441 81
        //Hexadecimal : 16b2c40bc117176a0f9a2598a1563aca6d5
        0x16B2C40B,0xC117176A,0x0F9A2598,0xA1563ACA,

        /*
            Mathematical Constants - Millions of Digits
            http://www.numberworld.org/constants.html
        */

        // Pi (3.243f6a8885a308d313198a2e0370734)
        0x243F6A88,0x85A308D3,0x13198102,0xE0370734,
        // Golden ratio (1.9e3779b97f4a7c15f39cc0605cedc834)
        0x9E3779B9,0x7F4A7C15,0xF39CC060,0x5CEDC834,
        //e Natural Constant (2.b7e151628aed2a6abf7158809cf4f3c7)
        0xB7E15162,0x8AED2A6A,0xBF715880,0x9CF4F3C7
    };
```

5.2 Forward and Backward Layers

The NeoAlzette S-box operates through forward and backward layers, providing encryption and decryption functionality, respectively.

Algorithm 1 NeoAlzette ARX S-box Layers

function NeoAlzette_ForwardLayer(a, b, rc)
  $b \leftarrow b \oplus a$
  $a \leftarrow (a \boxplus_{64} b) \ggg 31$
  $a \leftarrow a \oplus rc$
  $b \leftarrow b \boxplus_{64} a$
  $a \leftarrow (a \oplus b) \lll 24$
  $a \leftarrow a \boxplus_{64} rc$
  $b \leftarrow (b \lll 8) \oplus rc$
  $a \leftarrow a \boxplus_{64} b$
  $a \leftarrow a \oplus b$
  $b \leftarrow (a \boxplus_{64} b) \ggg 17$
  $b \leftarrow b \oplus rc$
  $a \leftarrow a \boxplus_{64} b$
  $b \leftarrow (a \oplus b) \lll 16$
  $b \leftarrow b \boxplus_{64} rc$
  return a, b
end function

function NeoAlzette_BackwardLayer(a, b, rc)
    $b \leftarrow b \boxminus_{64} rc$
    $b \leftarrow (b \ggg 16) \oplus a$
    $a \leftarrow a \boxminus_{64} b$
    $b \leftarrow b \oplus rc$
    $b \leftarrow (b \lll 17) \boxminus_{64} a$
    $a \leftarrow a \oplus b$
    $a \leftarrow a \boxminus_{64} b$
    $b \leftarrow (b \oplus rc) \ggg 8$
    $a \leftarrow a \boxminus_{64} rc$
    $a \leftarrow (a \ggg 24) \oplus b$
    $b \leftarrow b \boxminus_{64} a$
    $a \leftarrow a \oplus rc$
    $a \leftarrow (a \lll 31) \boxminus_{64} b$
    $b \leftarrow b \oplus a$
    return a, b
end function

5.3 Applied Little OaldresPuzzle_Cryptic algorithm Encryption and Decryption Process

In the Little OaldresPuzzle_Cryptic algorithm, the encryption and decryption processes involve multiple rounds of the forward and backward layers, respectively. Each round uses a different value from the ROUND_CONSTANT array. The number of rounds is predefined and ensures the algorithm's security and efficiency.

6 Little OaldresPuzzle_Cryptic Algorithm Description

Building upon the foundation laid by the ARX structure and the innovative XCR design, the "Little OaldresPuzzle_Cryptic" algorithm is a sophisticated cryptographic tool designed for high-speed data transmission with enhanced security. The precomputed constants used in each round of this algorithm are composed of complex mathematical principles and operations, such as bitwise operations, hexadecimal representation, irrational and transcendental numbers, sequence generation, pseudo-random number generation, etc, a combination of the above methods generate.

First, We need to specifically outline an algorithm for generator — computing the hexadecimal representation of the XCR round constants, which serves as a cornerstone for our cryptographic functions. This algorithm leverages the inherent unpredictability of several mathematical constants, including Euler's number, Pi, the Golden ratio, the square roots of 2 and 3(the cube roots of 2 and 3), Euler-Mascheroni constant, Feigenbaum constant, and the Plastic number. These constants, known for their irrational and transcendental characteristics, inject a fundamental element of randomness, thus augmenting the cryptographic strength of our algorithm. This binary sequence of constants, generated by a nonlinear Boolean function, is further segmented into 64-bit groups and returned as a hexadecimal string.

Second, in the previous introduction we illustrate our cryptographically secure pseudo-random number generator (CSPRNG), called "XorConstantRotation" or "XCR structural design". The XCR algorithm relies heavily on the constant data for each round that has been precomputed by the above process.

Subsequently, The heart of the discussion is devoted to our symmetric sequence cryptographic algorithm, the "Little OaldresPuzzle_Cryptic". The algorithm, designed as a class, employs the XorConstantRotation for its operations. In the coming sections, the paper lays out pseudo-codes for these algorithms, bringing their mechanics to light.

Flow of the algorithms

Algorithm 2 Generator - Computing XCR Round Constant Hexadecimal Representation

function GenerateRoundConstant ▷ This function generates the round constant
    $e \leftarrow 2.71828182845904523536028747135266249775724709369995957496696762772407663035354759457138217852516642 7$ ▷ Euler's number
    $\pi \leftarrow 3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706 8$ ▷ Pi
    $\phi \leftarrow 1.61803398874989484820458683436563811772030917980576286213544862270526046281890244970720720418939113 7$ ▷ Golden ratio $\frac{1+\sqrt{5}}{2}$
    $\sqrt{2} \leftarrow 1.41421356237309504880168872420969807856967187537694807317667973799073247846210703885038753432764157 3$ ▷ Square root of 2
    $\sqrt{3} \leftarrow 1.73205080756887729352744634150587236694280525381038062805580697945193301690880003708114618675724857 6$ ▷ Square root of 3
    $\gamma \leftarrow 0.5772156649$ ▷ Euler-Mascheroni constant
    $\delta \leftarrow 4.6692016091$ ▷ Feigenbaum constant
    $\rho \leftarrow 1.3247179572$ ▷ Plastic number $\sqrt[3]{\frac{9+\sqrt{69}}{18}} + \sqrt[3]{\frac{9-\sqrt{69}}{18}}$
    $x \leftarrow 1$
    $binary\_string \leftarrow$ ""
    for $index \leftarrow 0$ **to** 140 **do**
        $result \leftarrow (e^x - \cos(\pi x)) \times (\phi x^2 - \phi x - 1) \times (\sqrt[2]{2}x - \lfloor \sqrt[2]{2}x \rfloor) \times (\sqrt[2]{3}x - \lfloor \sqrt[2]{3}x \rfloor) \times \ln(1+x) \times (x\delta - \lfloor x\delta \rfloor) \times (x\rho - \lfloor x\rho \rfloor)$
    ▷ Original plan: $(e^x - \cos(\pi x)) \times (\phi x^2 - \phi x - 1) \times (\sqrt[2]{x} - \lfloor \sqrt[2]{x} \rfloor) \times (\sqrt[3]{x} - \lfloor \sqrt[3]{x} \rfloor) \times \ln(1+x) \times (x\delta - \lfloor x\delta \rfloor) \times (x\rho - \lfloor x\rho \rfloor)$
        $fractional\_part \leftarrow result - \lfloor result \rfloor$ ▷ Isolate the fractional part
        $binary\_fractional\_part \leftarrow$ Binary of $(fractional\_part \times 2^{128})$
        $hexadecimal\_fractional\_part \leftarrow$ Hexadecimal of $(fractional\_part \times 2^{128})$
        $integer\_part \leftarrow \lfloor result \rfloor$
        $x \leftarrow x + 1$
        $binary\_string \leftarrow binary\_string$.Append($binary\_fractional\_part$)
    end for
    $integer\_value \leftarrow$ Integer of $binary\_string$ ▷ print result
    $hexadecimal\_string \leftarrow$ Hexadecimal of $integer\_value$
    return $hexadecimal\_string$ ▷ Return the hexadecimal string of the round constant

24: **end function**

Algorithm 3 Cryptographically Secure Pseudo-Random Number Generator - XorConstantRotation (XCR)

function XCR INITIALIZE($seed$)
  $seed \in \mathbb{F}_2^{64}$
  $state \in \mathbb{F}_2^{64}$
  $x, y \in \mathbb{F}_2^{64}$
  $ROUND\_CONSTANT_i \in \mathbb{F}_2^{64}$
  $x, y, counter = 0$
  $state = seed$ ▷ Initial state is the seed
  if $state == 0$ **then**
    $state = 1$
  end if
  $\mathbb{F}_{2^{64}}$ $state_0 = state$
  $\mathbb{F}_{2^{64}}$ $random = state$
  for $round \leftarrow 0$ **to** $4 - 1$ **do** ▷ Goldreich-Goldwasser-Micali Construct PRF
    $\mathbb{F}_{2^{64}}$ $next\_random = 0$
    for $bit\_index \leftarrow 0$ **to** $64 - 1$ **do**
      $random \leftarrow$ XCR GENERATION($random$)
      if $random \mod 2 == 1$ **then**
        SETBIT(next_random, 0)
        Shift left by one bit or multiply by 2. (use $next\_random$)
      else
        SETBIT(next_random, 1)
        Shift left by one bit or multiply by 2. (use $next\_random$)
      end if
    end for
    $random = next\_random$
  end for
  $state = state \oplus (state_0 \boxplus_{64} random)$ ▷ Securely whitened uniformly randomized seeds
end function

function XCR GENERATION($number\_once$)
  $RC0, RC1, RC2 \leftarrow$ **Use Dynamic Round Constant Selection**
  if $x == 0$ **then**
    $x \leftarrow RC0$
  else
    DIFFUSION LAYER(x, y. state, counter, RC0) ▷ Update states with Diffusion layer
  end if
  CONFUSION LAYER(x, y. state, RC0, RC1, RC2) ▷ Update states with Confusion layer
  $counter = counter + 1$
  return y
end function

Algorithm 4 LittleOaldresPuzzle_Cryptic Class Implementation

$seed \leftarrow$ initial seed value ▷ Set the initial seed for CSPRNG
$csprng \leftarrow$ XORCONSTANTROTATION($number\_once$) ▷ The XorConstantRotation CSPRNG Instance
$rounds \leftarrow 8, 16, 32, 64 \ldots$ ▷ Set the number of rounds
function KEYSTATE($subkey, choise\_function, bit\_rotation\_amount\_a, bit\_rotation\_amount\_b, constant\_index$)
  return KeyState with the following attributes:
    subkey = $subkey$
    choise_function = $choise\_function$
    bit_ra = $bit\_rotation\_amount\_a$
    bit_rb = $bit\_rotation\_amount\_b$
    rc_index = $constant\_index$
end function
function GENERATEANDSTOREKEYSTATES($key, number\_once$)
  $rc\_index = 0$
  for $round \leftarrow 0$ **to** $rounds - 1$ **do**
    $key\_state \leftarrow$ KeyState()$_{round}$ ▷ Initialize KeyState
    $key\_state.subkey \leftarrow key \oplus$ CSPRNG($number\_once \oplus round$) ▷ Generate subkey using PRNG
    $key\_state.choise\_function \leftarrow$ CSPRNG($key\_state.subkey \oplus (key \gg 1)$) ▷ Generate choice function
    $key\_state.bit\_ra \leftarrow$ CSPRNG($key\_state.subkey \oplus key\_state.choise\_function$) ▷ Calculate bit rotation amount
    $key\_state.bit\_rb \leftarrow (key\_state.bit\_ra \gg 6) \mod 64$ ▷ Select bit position 6 to 11
    $key\_state.bit\_ra \leftarrow key\_state.bit\_ra \mod 64$ ▷ Select bit position 0 to 5
    $key\_state.choise\_function \leftarrow key\_state.choise\_function \mod 4$ ▷ Ensure choice function is in range [0, 3]
    $key\_state.rc\_index \leftarrow (rc\_index \gg 1) \mod 16$ ▷ NeoAlzette ROUND_CONSTANT array index
    $rc\_index = rc\_index + 2$
  end for
end function
function ADDROUNDKEY($result, key, subkey$)
  $result \leftarrow result \boxplus_{64} (key \oplus subkey)$
  $result \leftarrow (result \oplus key) \ggg 16$
  $result \leftarrow result \oplus ((key \boxplus_{64} subkey) \lll 48)$
end function
function SUBTRACTROUNDKEY($result, key, subkey$)
  $result \leftarrow result \oplus ((key \boxplus_{64} subkey) \lll 48)$
  $result \leftarrow (result \lll 16) \oplus key$
  $result \leftarrow result \boxminus_{64} (key \oplus subkey)$
end function
function ENCRYPTION($data, key, number\_once$)
  $result \leftarrow data$ ▷ Initialize result with the data

```
    GenerateAndStoreKeyStates(key, number_once)
    for round ← 0 to rounds − 1 do
        key_state ← key_state_round                                   ▷ Get KeyState
        F_{2^32} left32, right32 = BitSplit(result)
        rc = NeoAlzette ROUND_CONSTANTS_{key_state.rc_index}
        NeoAlzette_ForwardLayer(left32, right32, rc)
        result ← BitCombination(left32, right32)
        if key_state.choise_function = 0 then
            result ← result ⊕ key_state.subkey                        ▷ XOR operation
        else if key_state.choise_function = 1 then
            result ← result ⊖ key_state.subkey                        ▷ NOT XOR operation
        else if key_state.choise_function = 2 then
            result ← (result ⋘ key_state.bit_rb)                      ▷ Left bitwise rotation
        else if key_state.choise_function = 3 then
            result ← (result ⋙ key_state.bit_rb)                      ▷ Right bitwise rotation
        end if
        result ← result ⊕ (1 ≪ key_state.bit_ra)                      ▷ Random Bit Tweak (Nonlinear)
        AddRoundKey(result, key, key_state.subkey)                    ▷ Update result with AddRoundKey
    end for
    return result
end function
function Decryption(data, key, number_once)
    result ← data                                                     ▷ Initialize result with the data
    GenerateAndStoreKeyStates(key, number_once)
    for round ← rounds down to 1 do
        key_state ← key_state_{round−1}                               ▷ Get KeyState
        SubtractRoundKey(result, key, key_state.subkey)               ▷ Update result with SubtractRoundKey
        result ← result ⊕ (1 ≪ key_state.bit_ra)
        if key_state.choise_function = 0 then
            result ← result ⊕ key_state.subkey                        ▷ XOR operation
        else if key_state.choise_function = 1 then
            result ← result ⊖ key_state.subkey                        ▷ NOT XOR operation
        else if key_state.choise_function = 2 then
            result ← (result ⋙ key_state.bit_rb)                      ▷ Left bitwise rotation
        else if key_state.choise_function = 3 then
            result ← (result ⋘ key_state.bit_rb)                      ▷ Right bitwise rotation
        end if
        F_{2^32} left32, right32 = BitSplit(result)
        rc = NeoAlzette ROUND_CONSTANTS_{key_state.rc_index}
        NeoAlzette_BackwardLayer(left32, right32, rc)
        result ← BitCombination(left32, right32)
    end for
    return result
end function
function ResetPRNG(seed)
    XorConstantRotation.Seed(seed)                                    ▷ Reset the PRNG with a new seed
end function
```

We believe the readers are familiar with the three closely related algorithms mentioned above. However, they might now be grappling with several questions regarding their composition and purpose. Thus, it is essential to address these queries and provide a deeper understanding of our design intentions.

Each algorithm is essential for the system's integrity, offering unique functionalities:

XCR Round Constant Generator

- Establishes the basis for our cryptographic functions.
- Leverages a series of mathematical constants, each chosen for their irrational and transcendental qualities, infusing randomness and intricacy into the cryptographic algorithm.

XorConstantRotation (XCR)

- Serves as the primary cryptographically secure pseudo-random number generator.
- The algorithm pre-computes its own built-in round constant generator to enhance the unpredictability and security of the sequences generated by the **XCR** algorithm.
- Relies on ARX primitives and Generated XCR Round Constants, known for their superior confusion and diffusion capabilities, to distribute input changes uniformly across the output.

LittleOaldresPuzzle_Cryptic

- A versatile and lightweight encryption and decryption solution.
- Capable of functioning both as a sequential cipher for stream encryption and as a parallel encryption mechanism for block ciphers.
- Employs complex bitwise operations and ARX primitives in the encryption algorithm, with each step enhancing the overall security.
- The decryption algorithm mirrors the encryption steps, effectively reversing the encryption process using subkeys generated by **XCR**.

Together, these algorithms interlace to form a secure, cohesive, and efficient cryptographic framework, ensuring data security while maintaining operational versatility and efficiency.

7 Performance Evaluation and Security Evaluation

7.1 Performance Evaluation

This document's appendices present **Tables 1 to 3**, meticulously detailing the performance and operational characteristics of lightweight cryptographic algorithms. These tables are designed to provide an exhaustive comparison of algorithm features, offering experts in cryptography a comprehensive understanding of their performance and functionality.

A comprehensive performance evaluation methodology has been established to assess the efficiency of lightweight cryptographic algorithms, with a particular focus on the ASCON algorithm. The goal is to offer a comparative analysis of encryption and decryption operations across various data sizes, incorporating insights from multiple lightweight algorithms.

Experimental Setup:

The experimental setup involved the implementation of the **ASCON** cryptographic algorithm, utilizing a fixed key (0x0123456789ABCDEF, 0xFEDCBA9876543210) and nonce (0x0000000000000000, 0x0000000000000000). The associated data was set to a constant value of 0x12345678. For the **XCR/Little_OaldresPuzzle_Cryptic** algorithm, a 64-bit nonce was employed in counter mode, with no associated data and the same key. As for the Chacha20 algorithm, a predetermined key (0x00000000,0x00000000,0x00000000,0x00000000,0x01234567,0x89ABCDEF,0xFEDCBA98,0x76543210) and nonce (0x00000000, 0x00000000) were utilized.

To broaden the study's scope, lightweight encryption and decryption algorithms were also included in the evaluation. The experimental setup aimed to compare **ASCON** with these algorithms under identical conditions. The MT19937-64Bit pseudorandom number generator, seeded with 1, was used to generate 2^n bits of random data for each iteration.

Data Size Variation:

Ensuring a comprehensive analysis, the performance evaluation spanned a range of data sizes, from 128 bits to 671088640 bits (10 GB). The data sizes were selected in a doubling pattern (128, 256, 512, 1024, and so on) to represent a diverse array of input sizes.

After an extensive comparison, taking into account the simplicity of software implementation, alignment with standardization efforts, and an acute awareness of our limitations, we have selected the algorithms detailed in **Tables 4 to 6** for comparative performance analysis. This selection is based on performance metrics derived from our evaluation methodology, emphasizing the significance of both efficiency and reliability in cryptographic algorithms.

The performance evaluation was conducted on a CPU platform of Intel64 Family 6 Model 151 Stepping 2 GenuineIntel, operating at approximately 3610 MHz, with a RAM capacity of 65,277 MB. This hardware configuration was chosen to ensure that the results are representative of a modern computing environment, providing a realistic benchmark for the algorithms' performance.

Evaluation Result:

The comprehensive analysis of the **XCR/Little_OaldresPuzzle_Cryptic** algorithm in comparison to **ASCON** and **ChaCha20** algorithms, as detailed in our internal data and graphical representations, reveals a nuanced performance landscape.

The **XCR/Little_OaldresPuzzle_Cryptic** algorithm, while consistently slower than **ASCON**, exhibits a manageable performance degradation, particularly in the context of larger data sizes. Specifically, our measurements indicate that the **XCR/Little_OaldresPuzzle_Cryptic** algorithm operates at approximately half the speed of **ASCON** under certain time measurement conditions, yet it remains competitive, suggesting that its utility in practical applications is not significantly hindered by this discrepancy.

The performance metrics, as depicted in **Figures 1, 2**, further support these findings. The scalability and efficiency of **ASCON** are underscored, especially when handling larger data sizes, while **ChaCha20** shows a similar trend, albeit with different performance characteristics. The **XCR/Little_OaldresPuzzle_Cryptic** algorithm, despite its slower performance, maintains a steady pace, indicating a potential for use in environments where computational overhead is a consideration.

In conclusion, the performance evaluation positions **ASCON** as a leading candidate for lightweight cryptographic applications, particularly where rapid processing is critical. The **XCR/Little_OaldresPuzzle_Cryptic** algorithm, while not matching the speed of **ASCON** or **ChaCha20**, remains a viable alternative, especially in scenarios where a balance between speed and computational resources is essential. The choice between these algorithms should be guided by the specific requirements of the application, with careful consideration of the trade-offs between speed, efficiency, and resource utilization.

It is important to acknowledge that the performance metrics presented are based on a controlled environment and may vary under different conditions or with different hardware configurations. Future research should explore the robustness of these algorithms across a broader range of scenarios and hardware platforms to provide a more comprehensive understanding of their practical applicability.

The findings contribute to the ongoing discourse on the selection and optimization of cryptographic algorithms for modern security applications, emphasizing the need for algorithms that balance performance, efficiency, and resource consumption. The results provide valuable insights for developers and researchers in the field of cryptography, aiding in the development of secure and efficient systems.

The study's limitations, including the specific hardware used for the evaluation and the potential for varying results under different conditions, should be recognized. Further research is recommended to validate these findings and to explore the performance of these algorithms in a wider array of environments. This will ensure that the algorithms' performance is thoroughly understood in diverse settings, allowing for informed decisions in the implementation of cryptographic solutions.

7.2 Security Evaluation with Statistical Tests

In the context of data security assessments, while our expertise in advanced mathematical techniques is not extensive, we employ a methodology that emulates the properties of uniform randomness. This necessitates the application of an efficacious and robust statistical analysis to substantiate the unpredictability of the subsequent bit generation within our algorithm. Our methodology entails restricting each encryption operation to a 128-bit data segment, and subsequently writing the resultant samples into files, each comprising 128 kilobytes, amounting to a total of 128 sample files. The evaluation process is bifurcated into two distinct phases.

In the **first phase**, the plain-text is uniformly set to all zeros, while the key is used as a unique incrementing counter. This phase represents a rigorous test as it examines the scenario where the plain-text is completely devoid of randomness, and all information is derived from the key and any seed that may be used by the **XCR CSPRNG** algorithm. Theoretically, this initialization is an exceedingly subtle test intended to observe specific properties within probability distributions.

Conversely, in the **second phase**, the plain-text is derived from a sequence of random data generated by the MT19937-64Bit algorithm, with the key maintaining its role as a unique incrementing counter. The significance of this phase lies in the statistical test outcomes under the conditions where the plain-text is random but potentially intermixed with differential malicious data (Theoretically analogous distinguisher between the outputs of true random and algorithmic pseudo-randomization), along with a unique incrementing counter key.

7.3 The Credibility of Statistical Tests and the Rationale for Their Use

The credibility of these tests stems from their mathematical rigor and empirical validation. They are based on well-established statistical principles and have been extensively analyzed and tested across various scenarios. The tests are not only theoretically sound but also practically effective, having been applied in numerous security evaluations and standards, such as the National Institute of Standards and Technology (NIST SP 800-22) and China Randomness test specification(GM/T 0005-2021 standards). A description of the specifics of each of the Chinese randomness test standards we used is provided in the Appendix.

The rationale for employing these tests in our evaluation process is multifaceted. Firstly, they offer an objective and systematic approach to assessing the randomness of our encryption algorithm's output. By subjecting the generated bits to a battery of tests, we can gain confidence in the algorithm's ability to produce unpredictable sequences, which is essential for thwarting potential attacks that rely on the predictability of the encryption process.

Secondly, the use of multiple tests provides a comprehensive assessment. Each test targets a different aspect of randomness, and together they form a robust evaluation framework. This ensures that any weaknesses in the algorithm's randomness are likely to be detected, as no single test can cover all possible scenarios.

Lastly, the choice of tests and parameters is informed by the specific requirements of our encryption algorithm and the nature of the data being encrypted.

7.4 Integrating P-Values, Q-Values, and Conditional Probability in Cryptographic Analysis

In the realm of cryptographic analysis, the application of statistical hypothesis testing, particularly through the use of P-values and Q-values, alongside the assessment of conditional probabilities, is essential for ensuring the reliability and security of cryptographic algorithms. The P-value, representing the probability of obtaining an observed result under the null hypothesis, serves as a critical measure for determining the statistical significance of findings in the context of encryption.

P-values are instrumental in identifying instances where the behavior of an algorithm deviates significantly from what would be expected under random conditions, thus flagging potential vulnerabilities. However, given the constraints of P-values, including their susceptibility to sample size effects and their inability to directly convey the probability of the hypothesis, Q-values are introduced in multiple hypothesis testing scenarios. Q-values, an adjustment of P-values, play a pivotal role in controlling the False Discovery Rate (FDR), thereby reducing the risk of false positives that can misguide the evaluation process. For details on Q-value, please refer to the paper [J, 2016].

The evaluation of conditional probability, denoted as $\mathbf{Pr}(value_P|value_Q)$, is equally vital in cryptographic analysis. It measures the likelihood of observing a specific bit pattern $value_P$ in the encrypted output, given the occurrence of another pattern $value_Q$. This probabilistic assessment helps in understanding the dependencies or correlations between different bit patterns, enabling a more nuanced analysis of the encryption algorithm's behavior and its potential vulnerabilities.

Together, the integrated application of P-values, Q-values, and conditional probability analysis significantly enhances the robustness and credibility of cryptographic evaluations. This triad of statistical tools allows for a more comprehensive scrutiny of algorithms, revealing not just anomalies but also ensuring that the findings are statistically sound and less prone to false discoveries.

7.4.1 Test Result With Phases 1

In Phase 1 of the **Chacha20** evaluation, wherein the data is set to all zeros and the key operates in Counter Mode, all statistical tests conducted failed to pass, regardless of attempts at both 32-bit and 64-bit trials. This highlights Chacha20's suboptimal performance when subjected to non-random data, particularly in the 64-bit scenario.

Our test results reveal that the **XCR/Little_OaldresPuzzle_Cryptic** algorithm not only closely mirrors the performance characteristics of the **ASCON** algorithm, as inferred from the aforementioned conclusions, but also maintains a level of random uniformity indistinguishable from the **ASCON** algorithm. This assertion will be visually substantiated through a table presented in the **Figure 3 to 6** included in the appendix.

It is essential to note that due to the extensive nature of our testing, yielding 128 individual data files, we have chosen the final binary file as our reference dataset. Consequently, all data presented in the tables within our screenshots is derived exclusively from the results of this last binary file. The comprehensive statistical test results, including the complete Excel spreadsheet, will be made available in our code repository. HereIsLink

7.4.2 Test Result With Phases 2

Transitioning to Phase 2, where data is generated using MT19937-64Bit with Seed 1 and the key is a random 64-bit value (32-bit * 2), **Chacha20** exhibits significantly improved performance. In this phase, all tests yielded satisfactory or excellent results.

It becomes evident that **Chacha20** encounters challenges in maintaining security when confronted with non-random input data, underscoring the algorithm's dependence on randomness, especially in the 64-bit context.

Chacha20's strengths lie in its capacity to provide ample bit distribution, approaching pseudo-randomness without fully achieving it, even with minimal input data. It excels in scenarios where the input data exhibits sufficient chaos, showcasing the intricacies of the algorithm.

In conclusion, while **Chacha20** demonstrates merits, especially in Phase 2 where randomness is better preserved, the **XCR/Little_OaldresPuzzle_Cryptic** algorithm emerges as the superior choice when **Chacha20** struggles to uphold security and robustness, particularly in non-random data scenarios. The test results will be reflected in **Figure 7 to 10**.

7.4.3 Evaluate Test Results

Upon synthesizing the insights garnered from both Phase 1 and Phase 2 evaluations, a nuanced appreciation of the Chacha20 algorithm's efficacy is achieved. Phase 1 revealed its vulnerability to non-random data, which resulted in suboptimal outcomes across multiple iterations. Conversely, Phase 2 demonstrated a significant enhancement in performance, particularly when the input data conformed to a more stochastic distribution.

XCR/Little_OaldresPuzzle_Cryptic algorithm has consistently exhibited competitive performance, maintaining a level of randomness akin to the ASCON algorithm. This resilience to data patterns underscores its potential in environments where randomness is paramount.

Given the observed performance of **Chacha20**, it becomes apparent that it may not be an ideal candidate for consideration in subsequent evaluations. Consequently, **Chacha20** has been excluded from our pool of reference algorithms for further assessment. This decision narrows our focus to the **ASCON** algorithm and the **XCR/Little_OaldresPuzzle_Cryptic** algorithm as the primary contenders for comparison and evaluation. The upcoming analysis will scrutinize and compare the strengths and weaknesses of **ASCON** and **XCR/Little_OaldresPuzzle_Cryptic** to determine their suitability for our intended application.

In addressing concerns regarding the perceived issues with **Chacha20**, it is imperative to substantiate our assertions with robust and compelling data. To enhance the persuasiveness of our findings, comprehensive statistical results are meticulously presented in the tables within the appendix, accompanied by graphical representations in **Figures 11 to 14**. This transparent approach aims to provide readers with direct access to the raw data, fostering a clearer understanding of the intricacies and nuances of Chacha20's performance. We believe that this meticulous presentation of data in our supplementary materials will dispel any reservations and contribute to the confidence in the validity of our analysis.

8 Mathematically prove and deduce the security of our algorithms

In the foregoing analysis, the XCR/Little_OaldresPuzzle_Cryptic lightweight cryptographic algorithm has successfully passed the statistical tests of GM/T 0005-2021. This achievement is attributed to the layered application of ARX primitives, each contributing a degree of non-linearity to the transformation process. As each layer cumulatively integrates data patterns, the resultant distribution tends to approach uniformity. This is further complemented by our design approach in the encryption and decryption processes, where the application of simple linear functions results in an efficient blend of complexity.

Furthermore, the XorConstantRotation CSPRNG, integral to our algorithm, is designed with well-defined diffusion and confusion layers. The initial seeding of this CSPRNG undergoes a whitening process utilizing the Goldreich-Goldwasser-Micali (GGM) construction. The synergistic effect of these complex structures, combined with the statistical indistinguishability of the algorithm's output, supports our assertion that the algorithm aligns with the IND-CPA & IND-CCA semantic security models (chosen plaintext and ciphertext security). Regardless of the attacker's methodology, be it exhaustive search or the intricate examination of linear and non-linear layer interactions, the complexity of emergent behaviors presents substantial difficulties. This applies to differential analysis, linear analysis, and previously used rotation analysis specific to ARX primitive structures. Consequently, the probability of compromising any individual component of this complex structure is negligible, making the theoretical distinction of our pseudorandom algorithm highly improbable, with a lower bound probability of less than 2^{-64}.

8.1 Security Analysis of the XorConstantRotation Algorithm

Lemma 8.1. *Given the cryptographic properties of the XorConstantRotation (XCR) algorithm, the probability that an adversary can successfully compromise the algorithm's security under various attack models is negligible.*

Proof. **Initialization and Iteration Process:** The XCR algorithm initializes by defining a set X with uniquely seeded elements x_i. In the transformation phase, for each $x_i \in X$, a corresponding $y_i \in Y$ is generated through a deterministic process involving a one-time number n_i and a uniformly distributed random variable u_i, ensuring Y's elements are derived through a process mimicking a uniform distribution.

Probabilistic Analysis: To assert the uniformity of the distribution from which Y is generated, consider the transformation function $f : X \times N \to Y$, where N is the set of one-time numbers. $f(x_i, n_i)$ yields y_i, with each y_i appearing with equal probability if and only if X and N are uniformly distributed.

For any subset $S \subseteq Y$, the probability of selecting an element from S is proportional to the size of S, i.e.,

$$P(y_i \in S) = \frac{|S|}{|Y|},$$

assuming $|Y| = |X| = |N|$ and all sets are uniformly distributed. This relation showcases the inherent uniformity in the distribution of Y's elements.

Enhancing Security through Complexity: The complexity of bitwise operations within the XCR algorithm (such as rotations and XOR operations) alongside the integration of round constants further randomizes the transformation process, making Y statistically indistinguishable from a truly random uniform distribution.

Negligible Adversary Advantage: Given the algorithm's resistance to analysis and prediction, the advantage of any adversary $\mathcal{A}$ in distinguishing the output set Y from a uniform distribution is negligible, mathematically expressed as

$$\mathrm{Adv}_{\mathcal{A}}(Y, U) = |\Pr[\mathcal{A}(Y) = 1] - \Pr[\mathcal{A}(U) = 1]| \leq 2^{-\lambda},$$

where λ is the security parameter. This formalizes the assertion that the XCR algorithm's outputs are indistinguishable from a uniform distribution, thereby underpinning its cryptographic security. □

8.2 ARX Probabilistic Analysis [Ya, 2017]

ARX operations are pivotal for the security of block ciphers. For an ARX-based cryptographic algorithm, we analyze the probabilities of differential and linear properties.

8.2.1 Differential Analysis

where EvaluationEquation(α, β, γ) is defined as:

$$\text{EvaluationEquation}(\alpha, \beta, \gamma) = (\neg(\alpha \wedge \beta)) \oplus (\neg(\alpha \wedge \gamma))$$

The probability of a differential (α, β) leading to a differential γ over an addition operation can be estimated as:

If : EvaluationEquation$(\alpha \ll 1, \beta \ll 1, \gamma \ll 1) \wedge (\alpha \oplus \beta \oplus \gamma \oplus (\beta \ll 1)) = 0$
Then :

$$Pr(\alpha, \beta \to \gamma) = 2^{-wt(\text{EvaluationEquation}(\alpha,\beta) \wedge \text{mask}(n-1))}$$

Else :

$$Pr(\alpha, \beta \to \gamma) = 0$$

where mask$(n-1)$ denotes a bitmask with $n-1$ least significant bits set to 1.

For rotation and XOR operations, the differential probabilities can be considered as:

$$Pr(\alpha \to \beta) = \begin{cases} 1 & \text{if } \beta = \alpha \lll r \\ 0 & \text{otherwise} \end{cases}$$

$$P_r(\alpha, \beta \to \gamma) = \begin{cases} 1 & \text{if } \gamma = \alpha \oplus \beta \\ 0 & \text{otherwise} \end{cases}$$

8.2.2 Linear Analysis

The linear analysis of ARX operations involves studying the bias of linear approximations over the cipher's operations. For a given linear mask (μ, ν) and an output mask ω, the bias ϵ of the approximation can be calculated as:

$$\epsilon(\mu, \nu \to \omega) = \left| \Pr[\mu \cdot x \oplus \nu \cdot y = \omega \cdot (x \oplus y)] - \frac{1}{2} \right|$$

where x and y are the input and output of an ARX operation, $\mu \cdot x$ denotes the bitwise dot product between the mask μ and the vector x, and similarly for $\nu \cdot y$ and $\omega \cdot (x \oplus y)$.

On the Additive Differential Characteristics of ARX Constructions: A Comprehensive Analysis

In the realm of cryptographic research, the security evaluation of algorithms predicated on ARX (Add-Rotate-XOR) constructs is of paramount importance. This is particularly true when it comes to scrutinizing the additive differential characteristics that are intrinsic to these cryptographic primitives. A seminal paper [Niu et al., 2023] serves as a cornerstone for this exploration, meticulously delineating the additive differential probabilities within the context of ARX-based cryptographic mechanisms.

The study at hand meticulously examines the additive differential probabilities that emerge from the ARX framework, considering the variables $x, y \in \mathbb{F}_{2^n}$ alongside the rotation parameters $0 \leq r1, r2 < n$. The focal point of the analysis is the structural configuration $(x \boxplus_n y) \lll r1 \oplus y \lll r2$, which is a critical element in a plethora of cryptographic systems that leverage ARX.

The research methodology incorporates a sophisticated partitioning strategy, segmenting $\mathbb{F}_{2^m}$ into distinct subsets. These subsets are characterized by elements that adhere to particular equations, facilitating an exact quantification of the additive differential probabilities for ARX constructions. This partitioning technique, though complex, yields profound understanding into the behavior of differential propagation through ARX primitives.

It is important to note that the computational complexity associated with the analytical method presented in the study is equivalent to executing 4n matrix multiplications, each of size 8×8. This computational intensity is indicative of the substantial computational effort required to thoroughly analyze ARX structures, further emphasizing the methodological rigor and depth of the research.

8.2.3 Probability Range Estimation

The weakest link in the ARX operations defines the maximum probability of non-random behavior, while the strongest link defines the minimum probability. The overall security of the cipher is determined by the weakest link, which should have a probability as low as possible, ideally close to 2^{-n}.

In the Little_OaldresPuzzle_Cryptic algorithm, the ARX operations are designed to ensure that these probabilities are minimized, thereby maximizing the cipher's resistance to both differential and linear cryptanalysis.

8.3 Advanced Security Analysis of the NeoAlzette ARX Layer

8.3.1 NeoAlzette ARX Layer Differential Path

Lemma 8.2. *Given two different input differences Δa and Δb leading to a consistent output difference Δc in the NeoAlzette ARX Layer, the probability that the same output difference Δc is produced by subsequent operations is approximately 2^{-n}, where n is the bit length of the inputs.*

Proof. Let us formalize the ARX operation within the NeoAlzette cipher as follows. For inputs a and b, the output c after one round of ARX operation can be described by:

$$c = (a \boxplus b) \oplus (a \oplus \Delta a \boxplus b \oplus \Delta b),$$

where $\boxplus$ denotes addition modulo 2^n and $\oplus$ represents the bitwise XOR operation. The difference Δc can thus be expressed as:

$$\Delta c = c \oplus (a \boxplus b).$$

The probability $P(\Delta a, \Delta b \to \Delta c)$ for a specific $(\Delta a, \Delta b)$ pair leading to a specific Δc can be modeled by considering all possible intermediate states of the cipher, $x \in \mathbb{F}_2^n$, as:

$$P(\Delta a, \Delta b \to \Delta c) = \sum_{x \in \mathbb{F}_2^n} \Pr[\Delta a, \Delta b \to \Delta c \,|\, x],$$

assuming that the cipher state transitions are statistically independent.

Considering the bitwise independence of the operations and the uniform probability distribution of Δa_i, Δb_i, and Δc_i across all n bits, the overall probability of achieving a consistent Δc across subsequent operations can be calculated as:

$$\begin{aligned} P(\text{consistent } \Delta c) &= \prod_{i=1}^{n} P(\Delta a_i, \Delta b_i \to \Delta c_i) \\ &= \left(\frac{1}{2}\right)^n \\ &= 2^{-n}, \end{aligned}$$

where we have utilized the fact that for each bit, there is a $\frac{1}{2}$ chance of maintaining the difference due to the properties of the XOR operation in a uniformly random environment. This concludes that the probability of maintaining a consistent output difference Δc through subsequent ARX operations is 2^{-n}. □

8.3.2 NeoAlzette ARX Layer Linear Approximation

Lemma 8.3. *Given a function $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$ involved in the NeoAlzette ARX Layer, if there exists an input mask μ and an output mask ν such that the distribution of $f(x \oplus \mu) \oplus f(x) \oplus \nu$ is non-uniform, then a linear approximation of f relative to (μ, ν) has been identified.*

Proof. Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a function within the ARX cipher, where $\mathbb{F}_2^n$ denotes the n-dimensional vector space over the binary field $\mathbb{F}_2$. Consider an input mask $\mu \in \mathbb{F}_2^n$ and an output mask $\nu \in \mathbb{F}_2^n$. Define the approximation probability as:

$$P_{\mu,\nu} = \Pr_{x \in \mathbb{F}_2^n}[f(x \oplus \mu) \oplus f(x) = \nu],$$

where x is chosen uniformly at random from $\mathbb{F}_2^n$.

The bias of this approximation, $\epsilon_{\mu,\nu}$, is defined as the deviation of $P_{\mu,\nu}$ from the uniform distribution:

$$\epsilon_{\mu,\nu} = |P_{\mu,\nu} - \frac{1}{2}|.$$

A significant non-zero bias $\epsilon_{\mu,\nu}$ indicates that the linear approximation $\mu \cdot x = \nu \cdot f(x)$ holds with a probability different from what would be expected under a purely random distribution. Specifically, for a non-uniform distribution of $f(x \oplus \mu) \oplus f(x) \oplus \nu$, the correlation between input and output masks through f deviates from randomness, signaling a linear approximation.

Mathematically, the strength of this approximation can be quantified over the sample space $\mathbb{F}_2^n$ as:

$$\epsilon_{\mu,\nu} = \left| \frac{1}{2^n} \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \mu, x \rangle \oplus \langle \nu, f(x) \rangle} - \frac{1}{2} \right|,$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product over $\mathbb{F}_2^n$.

A linear approximation is thus identified by demonstrating a statistically significant $\epsilon_{\mu,\nu}$, indicating that the behavior of f with respect to μ and ν deviates from that expected of a random function. □

8.4 In-Depth Uniform Distribution Analysis of Subkeys and Key States

Lemma 8.4. *In the Little_OaldresPuzzle_Cryptic algorithm, the distribution of all subkeys and key states is uniform and unpredictable, assuming the XCR CSPRNG is cryptographically secure.*

Proof. Consider the XCR CSPRNG as the underlying generator for both subkeys and key states in the Little_OaldresPuzzle_Cryptic algorithm. We define the statistical distance Δ to quantify the deviation of a given distribution from the ideal uniform distribution across the space Ω.

For a particular round r within a total of n rounds, the generation of a subkey ks.subkey_r involves the XOR operation with the output of the XCR CSPRNG, parameterized by a nonce and the round number:

$$\text{ks.subkey}_r = \text{key} \oplus \text{XCR\_CSPRNG}(\text{nonce} \oplus r).$$

The statistical distance from uniformity for ks.subkey_r is given by:

$$\Delta(\text{ks.subkey}_r, U) = \frac{1}{2} \sum_{y \in \Omega} \left| \Pr[\text{ks.subkey}_r = y] - \frac{1}{|\Omega|} \right|.$$

Given the CSPRNG's cryptographic security, its output distribution approximates the uniform distribution over Ω. Hence, for any specific value y within Ω, $\Pr[\text{ks.subkey}_r = y]$ is approximately $\frac{1}{|\Omega|}$, making $\Delta(\text{ks.subkey}_r, U)$ negligible.

Further operations, such as the choice function selection and bit rotation amounts calculation, leverage the CSPRNG's output in a similar manner, preserving the uniformity and unpredictability:

$$\begin{aligned} \text{ks.choice\_function}_r &= \text{XCR\_CSPRNG}(\text{keystate.subkey}_r \oplus (\text{key} \gg 1)), \\ \text{ks.bit\_rotation\_amount\_a}_r &= \text{XCR\_CSPRNG}(\text{subkey}_r \oplus \text{ks.choice\_function}_r), \\ \text{ks.bit\_rotation\_amount\_b}_r &= (\text{ks.bit\_rotation\_amount\_a}_r \gg 6) \mod 64. \end{aligned}$$

The negligible statistical distance from the uniform distribution for each component underlines the efficacy of the XCR CSPRNG in ensuring the uniformity and unpredictability of the generated subkeys and key states. This uniform distribution is essential for the security properties of the Little_OaldresPuzzle_Cryptic algorithm, as it obfuscates the linear and non-linear relationships between the plaintext, ciphertext, and the secret keys.

□

8.5 Key Mixing in Little_OaldresPuzzle_Cryptic and Cryptanalysis

Key Mixing Definitions

$$\begin{aligned} \text{mixed\_key} &= (key \boxplus_{64} \text{ks.subkey}) \lll 48, \\ \text{probabilistic\_key} &= key \oplus \text{ks.subkey}. \end{aligned}$$

8.5.1 Encryption and Decryption Processes

Equivalent use steps with Encryption

$$\begin{aligned} &\text{result} = \text{result} \boxplus_{64} \text{probabilistic\_key}, \\ &\text{result} = \text{result} \oplus \text{key}, \\ &\text{result} = \text{result} \ggg 16, \\ &\text{result} = \text{result} \oplus \text{mixed\_key}. \end{aligned}$$

Equivalent use steps with Decryption

$$\begin{aligned}
&\text{result} = \text{result} \boxminus_{64} \text{probabilistic_key},\\
&\text{result} = \text{result} \oplus \text{key},\\
&\text{result} = \text{result} \lll 16,\\
&\text{result} = \text{result} \oplus \text{mixed_key}.
\end{aligned}$$

8.5.2 Analysis of Potential Attacks

Key Mixing Mechanism Given the key mixing operations, the encryption and decryption can be represented as:

$$\begin{aligned}
\text{Encrypted Result} &= ((\text{result} \boxplus_{64} \text{probabilistic_key}) \oplus \text{key} \ggg 16) \oplus \text{mixed_key},\\
\text{Decrypted Result} &= ((\text{result} \boxminus_{64} \text{probabilistic_key}) \oplus \text{key} \lll 16) \oplus \text{mixed_key}.
\end{aligned}$$

Lemma 8.5 (Probability Analysis in Key Mixing)**.** *The non-linear nature of key mixing results in a scenario where random guessing approaches the optimal strategy.*

Proof. Let $P_{\text{non-linear}}$ be the probability that a non-linear operation introduces a non-predictable transformation. In our algorithm, non-linear operations (like $\boxplus_{64}$ and $\boxminus_{64}$ with possible wrap-around) increase $P_{\text{non-linear}}$. We can approximate $P_{\text{non-linear}}$ by evaluating the likelihood of non-linear behavior in these operations:

$$P_{\text{non-linear}} = P_{\text{wrap-around in } \boxplus_{64} \text{ or } \boxminus_{64}} \text{ with } \mathbb{F}_{2^n}$$

Given a sufficiently random and large key space, the wrap-around probability and hence $P_{\text{non-linear}}$ are non-negligible. This non-linearity reduces the effectiveness of both differential and linear cryptanalysis, as their linear models become less accurate.

For random guessing, the probability of correctly guessing a key, P_{guess}, is inversely proportional to the key space size. As the key space size is large, P_{guess} becomes significantly small, yet it remains the most effective strategy against high $P_{\text{non-linear}}$ scenarios. Therefore, in the face of substantial non-linearity, random guessing emerges as the most feasible attack strategy, albeit with a very low success rate. □

Conclusion and Invitation for Collaboration

This enhanced analysis, fortified with specific mathematical formulations, further solidifies the understanding of the algorithm's robustness against various cryptanalysis forms. It reveals that, in light of the complex, non-linear key mixing process, the most effective strategy for an adversary would be random guessing, which is still impractical due to the vast key space. We encourage and welcome further mathematical analysis and collaboration from the cryptographic community to deepen the understanding and explore new dimensions of this cryptographic system.

8.6 Cryptographic Game Analysis of Encryption and Decryption Functions

8.6.1 Differential-Linear Attack Analysis [Lv et al., 2023]

Lemma 8.6. *The success probability of a differential-linear attack on the Little_OaldresPuzzle_Cryptic algorithm is bounded by 2^{-n}, under the assumption that the algorithm behaves like a random permutation.*

Proof. Consider a differential-linear attack comprising two phases: differential attack and linear attack. The overall success probability of this attack is determined by the joint probability of both phases succeeding. Given a set of parallel experiments $\{E_i(key_i)\}_{i=1}^{N}$ with randomly generated keys key_i from the keyspace, the average probability of success and the average correlation of a differential-linear attack on the Little_OaldresPuzzle_Cryptic algorithm can be computed as follows.

The process of each experiment E_i for a differential-linear attack is as follows:

Key Generation: Each key key_i for the experiment E_i is generated randomly.

$$key_i \sim \text{UniformRandom}(KeySpace) \tag{1}$$

Differential Analysis: For each generated key key_i, the following steps are performed to analyze the differential aspect:

- Calculate $Y = Enc(X, key_i)$ and $Y' = Enc(X \oplus \text{in\_diff}, key_i)$.
- Compute the output difference $\Delta Y = Y \oplus Y'$.

Linear Analysis: In the linear analysis step, the **dot_product** function is used to determine the correlation between the output approximation and the actual output difference. Mathematically, this function is defined as follows:

$$\text{dot\_product}(x, y) = \left(\sum_{i=0}^{n} ((x \wedge y) \gg i) \mod 2 \right) \mod 2 \tag{2}$$

For the same key key_i, the following steps are performed to analyze the linear aspect:

- Check if the dot product with the output approximation is zero: $\text{dot\_product}(\text{out\_approx}, \Delta Y) = 0$. If true, increment **Correct linear correlation counter**.

Combining Differential and Linear Analysis:

$$\text{Correct linear correlation counter}_i \text{ is incremented when both differential and linear conditions are satisfied.} \tag{3}$$

Probability and Correlation Computation: For each experiment E_i, calculate the probability of satisfying both conditions and the correlation:

$$\text{Probability: } P_i = \frac{\text{Correct linear correlation counter}_i}{2^n} \tag{4}$$

$$\text{Correlation: } C_i = \text{CorrelationFunction}(P_i) = P_i \times 2 - 1 \tag{5}$$

Then, calculate the average probability and correlation over all experiments:

$$\text{Average Probability: } \bar{P} = \frac{1}{N}\sum_{i=1}^{N} P_i \tag{6}$$

$$\text{Average Correlation: } \bar{C} = \frac{1}{N}\sum_{i=1}^{N} C_i \tag{7}$$

In the context of differential-linear analysis, two key parameters are used: **in_diff** and **out_approx**. The **in_diff** represents the input differential, which is a binary number formed by specific bit positions that are zero in the original input but are flipped to one in the differential input. Mathematically, it is represented as:

$$\text{in\_diff} = \bigoplus_{i \in S} 2^i \tag{8}$$

where S is the set of bit positions, and $\bigoplus$ denotes the bitwise XOR operation.

Similarly, **out_approx** is the output approximation used in linear analysis. It is also a binary number defined by specific bit positions that are set to one in the approximate output representation:

$$\text{out\_approx} = \bigoplus_{j \in T} 2^j \tag{9}$$

where T is the set of bit positions for the output approximation.

These parameters are crucial in differential-linear analysis, especially when breaking down algorithms with well-obfuscated and linearly complex component functions. The negligible probability of successfully breaking such algorithms using comprehensive differential-linear analysis underlines the robustness of these cryptographic systems.

Therefore, the overall success probability of a differential-linear attack on Little_OaldresPuzzle_Cryptic algorithm is bounded by 2^{-n}. □

Based on the proof of the priming formed by all the above cryptographic components, we can then arrive at the following theory.

Theorem 8.7. *The Little_OaldresPuzzle_Cryptic algorithm's encryption and decryption functions, when executed as per defined protocols, satisfy the IND-CPA and IND-CCA security requirements within a cryptographic "game" framework.*

Proof. **Game Setup:**

- Challenger ($\mathcal{C}$) operates the Little_OaldresPuzzle_Cryptic algorithm.
- Adversary ($\mathcal{A}$) attempts to break the algorithm's security.
- The game consists of two phases: encryption and decryption.

Encryption Phase:

1. $\mathcal{C}$ initializes the key states using `GenerateAndStoreKeyStates`.
2. For each round r, $\mathcal{C}$ performs the following steps on the input data:

$$X \xrightarrow{\text{NeoAlzette}} X'$$
$$X' \xrightarrow{\text{Mix Linear Transform}} Y$$
$$Y \xrightarrow{\text{Key Mixing}} Y'$$

3. The final state Y' is output as the encrypted data.

Decryption Phase:

1. $\mathcal{C}$ uses the same key states in reverse order.
2. For each round r, $\mathcal{C}$ reverses the encryption steps:

$$Y' \xrightarrow{\text{Key Unmixing}} Y$$
$$Y \xrightarrow{\text{Mix Linear Transform}^{-1}} X'$$
$$X' \xrightarrow{\text{NeoAlzette}^{-1}} X$$

3. The final state X is output as the decrypted data.

Game Play (IND-CPA and IND-CCA):

- **Under IND-CPA:**
  1. Adversary $\mathcal{A}$ selects two distinct plaintexts m_0 and m_1 and presents them to the challenger $\mathcal{C}$.
  2. $\mathcal{C}$ randomly chooses one of the plaintexts, say m_b, encrypts it, and gives the ciphertext c to $\mathcal{A}$.
  3. $\mathcal{A}$'s goal is to determine whether c corresponds to m_0 or m_1, represented by guessing b'.

4. The complexity of the Little_OaldresPuzzle_Cryptic algorithm ensures that, without additional information, $\mathcal{A}$'s best strategy is random guessing, leading to a probability of success $P_{\text{IND-CPA}} \approx \frac{1}{2}$.

- **Under IND-CCA:**

1. Similar to IND-CPA, but $\mathcal{A}$ has access to a decryption oracle for any ciphertext other than the challenge ciphertext c.
2. $\mathcal{A}$ tries to use the decryption oracle to gain additional information about the key or encryption process.
3. Due to the unpredictability of key states and the algorithm's transformation complexity, $\mathcal{A}$'s ability to derive useful information is significantly limited.
4. The success probability in this scenario is similarly bounded, $P_{\text{IND-CCA}} \approx \frac{1}{2} + \text{negl}(\lambda)$, where $\text{negl}(\lambda)$ is a negligible function representing the complexity of breaking the encryption. Here, λ represents the security parameter, typically measured in bits, indicating the theoretical complexity and security level of the encryption algorithm.

- **Overall Security:** The security of the Little_OaldresPuzzle_Cryptic algorithm in both scenarios is bolstered by the intricate design of each cryptographic round and the unpredictability of the key states, making any differential-linear attacks or oracle-based strategies infeasible within practical computational bounds.

Game Conclusion:

For an adversary $\mathcal{A}$ attempting to develop a distinguisher $\mathcal{D}$
for the Little_OaldresPuzzle_Cryptic algorithm, achieving success
without extensive computational time and resources is computationally infeasible.
Therefore, the probability of $\mathcal{A}$ successfully discriminating
is bounded by $P_{\text{successful discrimination}}(\mathcal{A}) \leq \frac{1}{2^n}$,
demonstrating the algorithm's resilience against both IND-CPA
and IND-CCA attacks, outperforming random guessing strategies.

Therefore, the defined encryption and decryption protocols ensure that the Little_OaldresPuzzle_Cryptic algorithm is secure in the cryptographic game context, meeting the stringent requirements of IND-CPA and IND-CCA security models.

□

8.7 For a small summary of the mathematical proofs of these of our algorithms

The Little_OaldresPuzzle_Cryptic algorithm, through its complex and intricate design, achieves a high level of security as proven under the IND-CPA and IND-CCA models. It is well-suited for applications requiring robust cryptographic security.

9 Conclusion

In the holistic development of our algorithm, **XCR/Little_OaldresPuzzle_Cryptic**, we drew inspiration from the proven efficiency of the **ASCON** algorithm in real-world test evaluations and its pseudo-random indistinguishability, closely approaching a uniform random distribution. Leveraging these insights, we ensured our algorithm provides robust security while addressing the challenge of a substantial number of constant arrays needed for the values of nonlinear pseudo-random functions. To streamline the design structure, we focused on concise enhancements, distinguishing our approach from both the **ASCON** algorithm and **Chacha20**. This strategic refinement empowers users to make informed and balanced decisions based on their specific requirements.

This paper introduces our innovative symmetric sequence cryptographic algorithm, **XCR/Little_OaldresPuzzle_Cryptic**, showcasing its exceptional speed in encryption and decryption along with robust security features. In the ever-evolving digital landscape, our algorithm emerges as a cutting-edge solution for securely and efficiently managing large-scale binary data files. Notably, the inclusion of a quantum-resistant block cipher algorithm in the same repository highlights our commitment to anticipating and addressing future cryptographic challenges.

A Statistical Tests for Randomness Assessment

Monobit Frequency Test

This test evaluates the balance between the occurrences of 0s and 1s in the generated binary sequence. It is predicated on the hypothesis that a truly random sequence should exhibit an equal frequency of both bits.

Block Frequency Test (m=10000)

This test examines the frequency distribution of 1s within blocks of a specified size (m=10000). It is used to detect any deviation from the expected uniform distribution, which could indicate a lack of randomness.

Poker Test (m=4, m=8)

The Poker Test assesses the frequency of specific subsequence patterns within the binary sequence. For m=4 and m=8, it evaluates the occurrence of 2-bit and 4-bit patterns, respectively, to ensure their distribution is uniform.

Overlapping Subsequence Test (m=3, P1, P2; m=5, P1, P2)

This test analyzes the frequency of overlapping subsequences of length m (3 or 5) and their permutations (P1, P2). It is designed to detect any non-random clustering of bit patterns.

Run Tests (Run Count, Run Distribution)

Run Tests measure the total number of runs (sequences of consecutive identical bits) and their distribution across the sequence. These tests are sensitive to the presence of long runs, which may indicate a deviation from randomness.

Longest Run Test (m=10000)

Specifically, the Longest Run Test identifies the longest run of 1s (or 0s) within blocks of size m=10000. It is used to detect any anomalies in the distribution of run lengths.

Binary Derivative Test (k=3, k=7)

The Binary Derivative Test evaluates the randomness of the sequence by considering the differences between consecutive bits (k=3, k=7). It is based on the principle that a random sequence should exhibit no correlation between adjacent bits.

Autocorrelation Test (d=1, d=2, d=8, d=16)
Autocorrelation Tests analyze the correlation between a sequence and its shifted versions (with delays d=1, d=2, d=8, d=16). A random sequence should exhibit minimal autocorrelation.

Matrix Rank Test
This test involves constructing a matrix from the binary sequence and determining its rank. The rank is then compared against expected values for a random sequence, providing insights into the sequence's complexity.

Cumulative Sum Test (Forward, Backward)
Cumulative Sum Tests assess the distribution of the running sum of the binary sequence in both forward and backward directions. These tests are sensitive to the presence of systematic trends in the sequence.

Approximate Entropy Test (m=2, m=5)
Approximate Entropy Tests measure the unpredictability of the sequence by comparing the frequency of patterns of length m (2 or 5) with their overlapping counterparts. It is a measure of the sequence's complexity and unpredictability.

Linear Complexity Test (m=500, m=1000)
Linear Complexity Tests estimate the shortest linear feedback shift register (LFSR) that can generate the sequence. A higher complexity indicates a more random sequence.

Maurer's Universal Statistical Test (L=7, Q=1280)
This test evaluates the sequence against a universal statistical model, comparing the observed frequencies with those expected from a truly random sequence. It is a comprehensive test that assesses multiple statistical properties.

Discrete Fourier Transform Test
The Discrete Fourier Transform Test analyzes the frequency spectrum of the sequence. A random sequence should exhibit a uniform frequency distribution across all frequencies.

In summary, these statistical tests provide a comprehensive and systematic framework for assessing the randomness of our encryption algorithm's output. By ensuring that the generated bits pass these rigorous evaluations, we can assert the cryptographic strength of our algorithm, thereby safeguarding the security of the data it encrypts.

The statistical tests mentioned above are widely recognized and utilized in the field of cryptography and information security due to their ability to provide a quantitative measure of randomness. These tests are designed to detect patterns and biases that may indicate a lack of randomness, which is a critical property for secure cryptographic operations.

B About Git repository and run test

While our focus in this paper revolves around the symmetric sequence algorithm, our repository offers a comprehensive perspective on our cryptographic contributions. We invite interested readers and researchers to explore both algorithms, contribute insights, and collaborate on potential enhancements. In the dynamic realm of cryptography, collective efforts and continuous innovation are indispensable for staying prepared for future challenges.

Github Link: README.md

Within the repository, you will find two primary directories, *OOP* and *Template*. The *OOP* directory offers an object-oriented version of the algorithm, ideal for developers familiar with object-oriented programming. Conversely, the *Template* directory presents a simplified implementation, suitable for beginners or those seeking a more straightforward understanding of the algorithm.

Choose the implementation that suits your requirements, navigate to the appropriate directory, and follow the README.md instructions to compile and execute the tests. These tests will furnish a holistic understanding of the algorithm's cryptographic robustness, speed, and overall performance.

It's crucial to note the peculiar characteristics and necessary precautions while using these algorithms. For instance, encryption and decryption operations demand a reset of the internal key state after each use. Therefore, if an encryption operation follows a decryption operation (or vice versa), the internal key state must be reset first to ensure correct functioning.

B.1 Contributions

We value your feedback and contributions. If you encounter possible improvements or any issues, feel free to submit a pull request or open an issue in the GitHub repository.

C Additional Quantum-Resistant OaldresPuzzle_Cryptic Block Cipher Algorithm

This repository houses an additional robust block cipher algorithm, developed independently from the symmetric sequence algorithm that is the primary focus of this paper. Conceived with the anticipation of future cryptographic challenges, particularly those presented by quantum computing, this block cipher algorithm offers several defining characteristics. It is designed to mitigate risks associated with brute force attacks, withstand analytical attacks on the key, and resist potential quantum computer intrusions.

Although slower in encrypting and decrypting packets (as evidenced by tests with 10MB data packets and a 5120-byte key, which required approximately one and a half minutes to execute), the algorithm confers a significant advantage by future-proofing cryptographic systems against potential advancements in quantum computing. This symmetric block cipher cryptographic algorithm has been tailored to meet contemporary cryptographic requirements. It prioritizes unpredictability and a high level of analytical complexity, making it suitable for managing protected, large-scale binary data files while ensuring requisite cryptographic robustness.

For a more comprehensive understanding, detailed information regarding the implementation and unique characteristics of this block cipher algorithm can be found in the corresponding directory of the repository. Despite being outside the primary scope of this paper, which is dedicated to the symmetric sequence algorithm, we encourage interested readers and researchers to explore this quantum-resistant block cipher algorithm and its potential applications.

D Documents referenced

References

[Aumasson et al., 2007] Aumasson, J.-P., Fischer, S., Khazaei, S., Meier, W., and Rechberger, C. (2007). New features of latin dances: Analysis of salsa, chacha, and rumba. Cryptology ePrint Archive, Paper 2007/472. https://eprint.iacr.org/2007/472.

[Beierle et al., 2019] Beierle, C., Biryukov, A., dos Santos, L. C., Großschädl, J., Perrin, L., Udovenko, A., Velichkov, V., and Wang, Q. (2019). Alzette: a 64-bit arx-box (feat. crax and trax). Cryptology ePrint Archive, Paper 2019/1378. https://eprint.iacr.org/2019/1378.

[Bernstein, 2005] Bernstein, D. J. (2005). Salsa20 design. *Department of Mathematics, Statistics, and Computer Science. The University of Illinois at Chicago. Chicago*.

[Bernstein, 2008] Bernstein, D. J. (2008). The salsa20 family of stream ciphers. In *New stream cipher designs: the eSTREAM finalists*, pages 84–97. Springer.

[Bernstein et al., 2008a] Bernstein, D. J. et al. (2008a). Chacha, a variant of salsa20. In *Workshop record of SASC*, volume 8, pages 3–5. Citeseer.

[Bernstein et al., 2008b] Bernstein, D. J. et al. (2008b). The chacha family of stream ciphers. In *Workshop record of SASC*. Citeseer.

[Biryukov and Perrin, 2017] Biryukov, A. and Perrin, L. (2017). State of the art in lightweight symmetric cryptography. Cryptology ePrint Archive, Paper 2017/511. https://eprint.iacr.org/2017/511.

[Cai et al., 2022] Cai, W., Chen, H., Wang, Z., and Zhang, X. (2022). Implementation and optimization of chacha20 stream cipher on sunway taihulight supercomputer. *The Journal of Supercomputing*, 78(3):4199–4216.

[Fei, 2012] Fei, D. (2012). Research on safety of arx structures. Master's thesis, Xi'an University of Electronic Science and Technology, China.

[Ghafoori and Miyaji, 2022] Ghafoori, N. and Miyaji, A. (2022). Differential cryptanalysis of salsa20 based on comprehensive analysis of pnbs. In Su, C., Gritzalis, D., and Piuri, V., editors, *Information Security Practice and Experience*, pages 520–536, Cham. Springer International Publishing.

[J, 2016] J, Z. (2016). Q_value test: A new method on randomness statistical test. *China Journal of Cryptologic Research*, 3(2):192–201.

[Liu et al., 2021] Liu, J., Rijmen, V., Hu, Y., Chen, J., and Wang, B. (2021). Warx: efficient white-box block cipher based on arx primitives and random mds matrix. *Science China Information Sciences*, 65(3):132302.

[Lv et al., 2023] Lv, G., Jin, C., and Cui, T. (2023). A miqcp-based automatic search algorithm for differential-linear trails of arx ciphers(long paper). Cryptology ePrint Archive, Paper 2023/259. https://eprint.iacr.org/2023/259.

[Maitra et al., 2015] Maitra, S., Paul, G., and Meier, W. (2015). Salsa20 cryptanalysis: New moves and revisiting old styles. Cryptology ePrint Archive, Paper 2015/217. https://eprint.iacr.org/2015/217.

[Niu et al., 2023] Niu, Z., Sun, S., and Hu, L. (2023). On the additive differential probability of arx construction. *Journal of Surveillance, Security and Safety*, 4(4).

[Ranea et al., 2022] Ranea, A., Vandersmissen, J., and Preneel, B. (2022). Implicit white-box implementations: White-boxing arx ciphers. In Dodis, Y. and Shrimpton, T., editors, *Advances in Cryptology – CRYPTO 2022*, pages 33–63, Cham. Springer Nature Switzerland.

[Serrano et al., 2022] Serrano, R., Duran, C., Sarmiento, M., Pham, C.-K., and Hoang, T.-T. (2022). Chacha20-poly1305 authenticated encryption with additional data for transport layer security 1.3. *Cryptography*, 6(2).

[Sleem and Couturier, 2021] Sleem, L. and Couturier, R. (2021). Speck-R: An ultra light-weight cryptographic scheme for Internet of Things. *Multimedia Tools and Applications*, 80(11):17067 – 17102.

[Tsunoo et al., 2007] Tsunoo, Y., Saito, T., Kubo, H., Suzaki, T., and Nakashima, H. (2007). Differential cryptanalysis of salsa20/8. In *Workshop Record of SASC*, volume 28. Citeseer.

[Ya, 2017] Ya, H. (2017). Automatic method for searching impossible differentials and zero-correlation linear hulls of arx block ciphers. *Chinese Journal of Network and Information Security*, 3(7):58.

E Data Images

F Data Tables

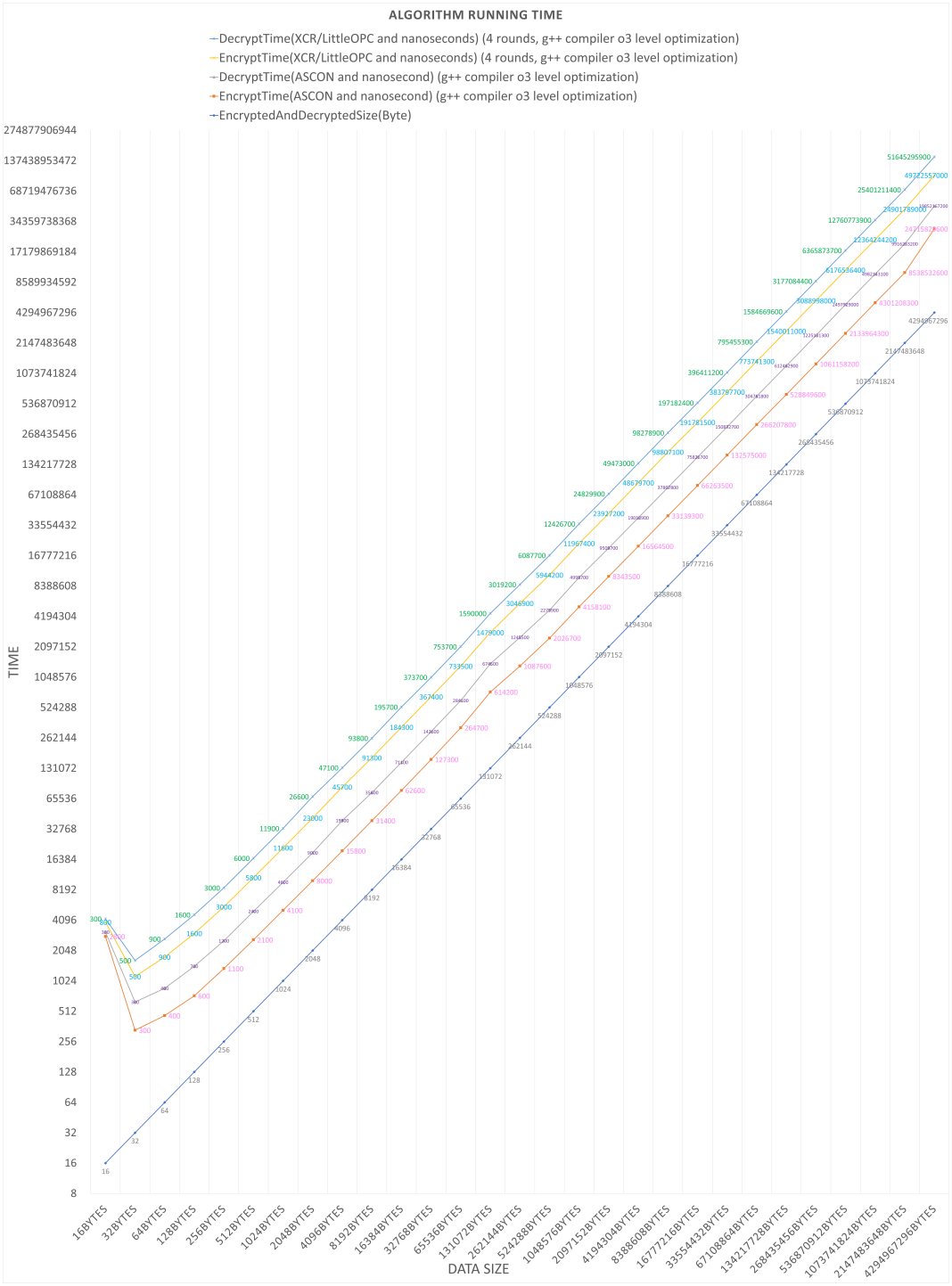

Figure 1: ASCON vs XCR/Little_OaldresPuzzle_Cryptic Algorithm Benchmark

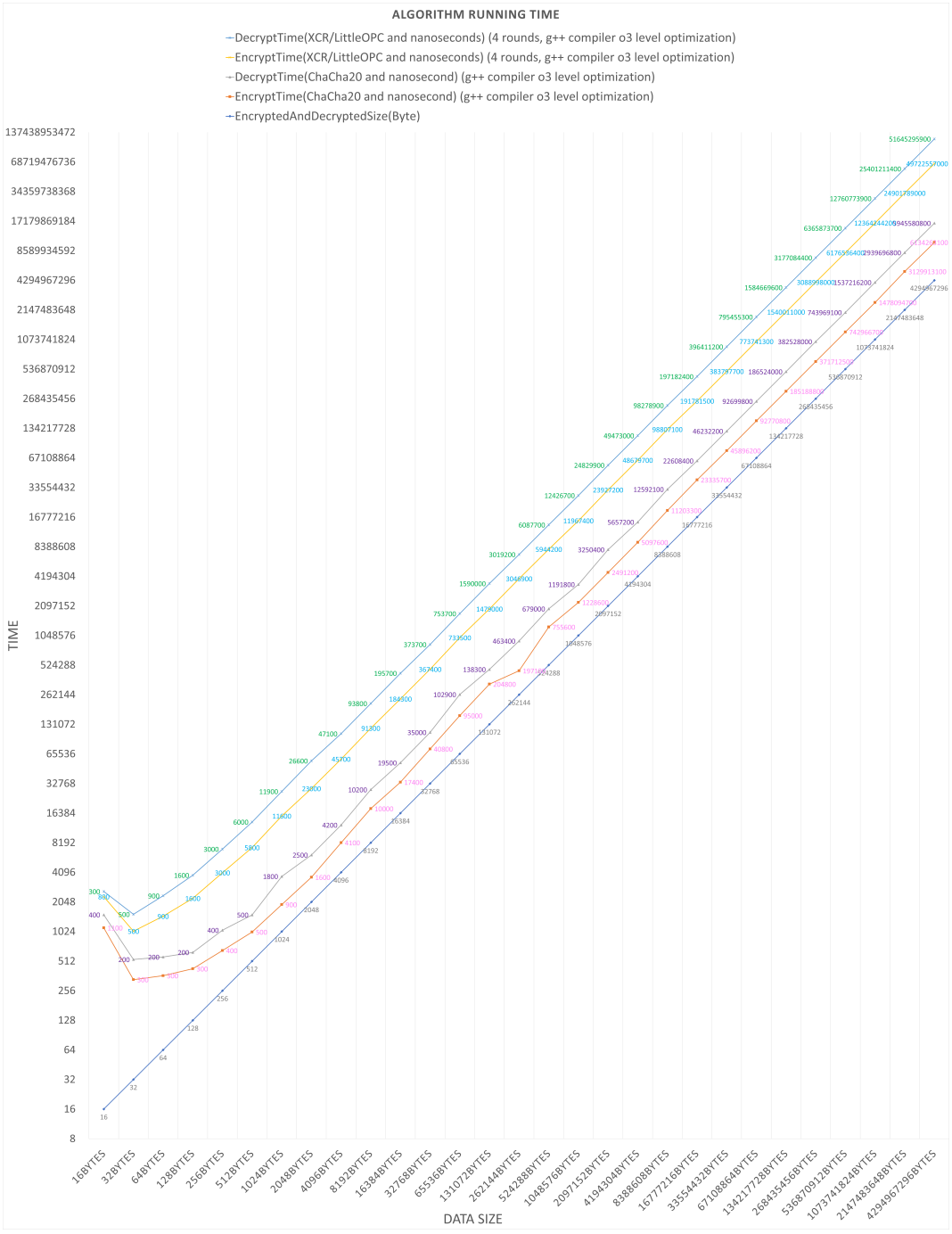

Figure 2: Chacha20 vs XCR/Little_OaldresPuzzle_Cryptic Algorithm Benchmark

SourceData Algoithm(ASCON)	Bit Frequency	Block Frequency m=10000	Poker Test m=4	Poker Test m=8	Overlapping Subsequence Test m=3 P1	Overlapping Subsequence Tes m=3 P2	Overlapping Subsequence Tes m=5 P1	Overlapping Subsequence Tes m=5 P2
Distribution uniformity	0.49378	0.509162	0.271449	0.3282	0.977331	0.096217	0.588437	0.365877
encrypted_data_127.bin	0.426659|0.213329	0.080330|0.080330	0.628529|0.628529	0.905010|0.905010	0.788751|0.788751	0.908157|0.908157	0.621571|0.621571	0.391326|0.391326
success count of the 128	125	126	128	128	126	128	126	126
Data Algoithm(XCR/Little OaldresPuzzle	Bit Frequency	Block Frequency m=10000	Poker Test m=4	Poker Test m=8	Overlapping Subsequence Test m=3 P1	Overlapping Subsequence Tes m=3 P2	Overlapping Subsequence Tes m=5 P1	Overlapping Subsequence Tes m=5 P2
Distribution uniformity	0.151616	0.478598	0.909344	0.250878	0.091798	0.620855	0.875539	0.966721
encrypted_data_127.bin	0.832936|0.583532	0.061707|0.061707	0.388531|0.388531	0.839661|0.839661	0.638679|0.638679	0.667996|0.667996	0.860652|0.860652	0.712521|0.712521
success count of the 128	125	126	128	128	126	128	126	126

Figure 3: ASCON vs XCR/Little_OaldresPuzzle_Cryptic(Phase 1) Statistical Test

Run Total	Run Distribution	Maximum 1 Run Test m=10000	Maximum 0 Run Test m=10000	Binary Deduction k=3	Binary Deduction k=7	Autocorrelation d=1	Autocorrelation d=2	Autocorrelation d=8
0.875539	0.180322	0.222262	0.353021	0.823278	0.701879	0.875539	0.096217	0.49378
0.346180|0.173090	0.703419|0.703419	0.180247|0.180247	0.568201|0.568201	0.482586|0.758707	0.918329|0.540836	0.346997|0.173498	0.664583|0.332292	0.269801|0.134901
126	126	127	128	128	128	126	127	127
Run Total	Run Distribution	Maximum 1 Run Test m=10000	Maximum 0 Run Test m=10000	Binary Deduction k=3	Binary Deduction k=7	Autocorrelation d=1	Autocorrelation d=2	Autocorrelation d=8
0.945993	0.717841	0.316241	0.733647	0.353021	0.733647	0.966721	0.653383	0.701879
0.654531|0.327266	0.133614|0.133614	0.741892|0.741892	0.003010|0.003010	0.132857|0.933572	0.838274|0.580863	0.658212|0.329106	0.253213|0.126607	0.579107|0.710447
128	126	127	125	127	128	128	126	125

Figure 4: ASCON vs XCR/Little_OaldresPuzzle_Cryptic(Phase 1) Statistical Test

Autocorrelation d=16	Matrix Rank Detection	Cumulative Sum Forward Detection	Cumulative Sum Backward Detection	Approximate Entropy m=2	Approximate Entropy m=5	Linear Complexity m=500	Linear Complexity m=1000	aurer Universal Statistical Test L=7 Q=12
0.15841	0.271449	0.293235	0.293235	0.991468	0.669618	0.075865	0.49378	0.056583
0.094546|0.047273	0.566020|0.566020	0.273852|0.273852	0.735775|0.735775	0.788349|0.788349	0.711070|0.711070	0.621920|0.621920	0.567503|0.567503	0.792955|0.603523
126	125	126	125	127	128	127	127	126
Autocorrelation d=16	Matrix Rank Detection	Cumulative Sum Forward Detection	Cumulative Sum Backward Detection	Approximate Entropy m=2	Approximate Entropy m=5	Linear Complexity m=500	Linear Complexity m=1000	aurer Universal Statistical Test L=7 Q=12
0.434395	0.145069	0.032381	0.764655	0.068865	0.524727	0.083495	0.97234	0.809134
0.269800|0.134900	0.490214|0.490214	0.393657|0.393857	0.558720|0.558672	0.638707|0.638707	0.791496|0.791496	0.867092|0.867092	0.005901|0.005901	0.420968|0.210484
126	125	126	125	127	128	127	127	126

Figure 5: ASCON vs XCR/Little_OaldresPuzzle_Cryptic(Phase 1) Statistical Test

Discrete Fourier
0.985508
0.440030|0.220015
127
Discrete Fourier
0.19626
0.983275|0.491638
127

Figure 6: ASCON vs XCR/Little_OaldresPuzzle_Cryptic(Phase 1) Statistical Test

SourceData Algoithm(ASCON)	Bit Frequency	Block Frequency m=10000	Poker Test m=4	Poker Test m=8	Overlapping Subsequence Test m=3 P1	Overlapping Subsequence Tes m=3 P2	Overlapping Subsequence Tes m=5 P1	Overlapping Subsequence Tes m=5 P2
Distribution uniformity	0.110612	0.180322	0.100821	0.809134	0.540457	0.261012	0.478598	0.379023
encrypted_data_127.bin	0.948609|0.525695	0.821650|0.821650	0.081845|0.081845	0.049667|0.049667	0.843075|0.843075	0.994702|0.994702	0.962718|0.962718	0.713031|0.713031
success count of the 128	125	124	126	127	125	127	125	127
Data Algoithm(XCR/Little OaldresPuzzle	Bit Frequency	Block Frequency m=10000	Poker Test m=4	Poker Test m=8	Overlapping Subsequence Test m=3 P1	Overlapping Subsequence Tes m=3 P2	Overlapping Subsequence Tes m=5 P1	Overlapping Subsequence Tes m=5 P2
Distribution uniformity	0.406167	0.231505	0.701879	0.013073	0.11581	0.379023	0.012376	0.213309
encrypted_data_127.bin	0.351522|0.824239	0.742014|0.742014	0.006374|0.006374	0.023008|0.023008	0.636326|0.636326	0.473802|0.473802	0.897367|0.897367	0.611916|0.611916
success count of the 128	127	125	126	128	127	128	127	126

Figure 7: ASCON vs XCR/Little_OaldresPuzzle_Cryptic(Phase 2) Statistical Test

Run Total	Run Distribution	Maximum 1 Run Test m=10000	Maximum 0 Run Test m=10000	Binary Deduction k=3	Binary Deduction k=7	Autocorrelation d=1	Autocorrelation d=2	Autocorrelation d=8
0.717841	0.524727	0.653383	0.188154	0.353021	0.316241	0.588437	0.034101	0.863186
0.238126|0.880937	0.646790|0.646790	0.091817|0.091817	0.796564|0.796564	0.734709|0.632645	0.086911|0.043456	0.237736|0.881132	0.920654|0.539673	0.992208|0.496104
126	127	125	127	127	125	125	127	125
Run Total	Run Distribution	Maximum 1 Run Test m=10000	Maximum 0 Run Test m=10000	Binary Deduction k=3	Binary Deduction k=7	Autocorrelation d=1	Autocorrelation d=2	Autocorrelation d=8
0.863186	0.11581	0.138761	0.909344	0.909344	0.620855	0.809134	0.875539	0.653383
0.666806|0.333403	0.226913|0.226913	0.461045|0.461045	0.109227|0.109227	0.571781|0.285890	0.628812|0.685594	0.668133|0.334067	0.756145|0.378072	0.044662|0.022331
127	128	124	127	126	128	127	128	127

Figure 8: ASCON vs XCR/Little_OaldresPuzzle_Cryptic(Phase 2) Statistical Test

Autocorrelation d=16	Matrix Rank Detection	Cumulative Sum Forward Detection	Cumulative Sum Backward Detection	Approximate Entropy m=2	Approximate Entropy m=5	Linear Complexity m=500	Linear Complexity m=1000	aurer Universal Statistical Test L=7 Q=12
0.937783	0.448892	0.837024	0.180322	0.540457	0.19626	0.823278	0.406167	0.966721
0.348498|0.825751	0.606737|0.606737	0.978810|0.978810	0.993374|0.993374	0.843125|0.843125	0.741601|0.741601	0.401628|0.401628	0.794323|0.794323	0.706975|0.646512
127	128	126	125	127	127	128	127	126
Autocorrelation d=16	Matrix Rank Detection	Cumulative Sum Forward Detection	Cumulative Sum Backward Detection	Approximate Entropy m=2	Approximate Entropy m=5	Linear Complexity m=500	Linear Complexity m=1000	aurer Universal Statistical Test L=7 Q=12
0.588437	0.448892	0.222262	0.434395	0.145069	0.42015	0.304586	0.316241	0.809134
0.509150|0.745425	0.495724|0.495724	0.117872|0.117872	0.569724|0.569724	0.635630|0.635630	0.540821|0.540821	0.336414|0.336414	0.256558|0.256558	0.445564|0.222782
127	125	126	127	127	128	126	128	127

Figure 9: ASCON vs XCR/Little_OaldresPuzzle_Cryptic(Phase 2) Statistical Test

Discrete Fourier
0.945993
0.711123|0.355561
127
Discrete Fourier
0.588437
0.884715|0.557643
127

Figure 10: ASCON vs XCR/Little_OaldresPuzzle_Cryptic(Phase 2) Statistical Test

SourceData Algoithm(Chacha20 Phase 1	Bit Frequency	Block Frequency m=10000	Poker Test m=4	Poker Test m=8	Overlapping Subsequence Test m=3 P1	Overlapping Subsequence Tes m=3 P2	Overlapping Subsequence Tes m=5 P1	Overlapping Subsequence Tes m=5 P2
Distribution uniformity	0	0	0	0	0	0	0	0
encrypted_data_127.bin	0.000000|0.000000	0.000000|0.000000	0.000000|0.000000	0.000000|0.000000	0.000000|0.000000	0.000000|0.000000	0.000000|0.000000	0.000000|0.000000
success count of the 128	2	23	0	0	0	1	0	0
SourceData Algoithm(Chacha20 Phase 2	Bit Frequency	Block Frequency m=10000	Poker Test m=4	Poker Test m=8	Overlapping Subsequence Test m=3 P1	Overlapping Subsequence Tes m=3 P2	Overlapping Subsequence Tes m=5 P1	Overlapping Subsequence Tes m=5 P2
Distribution uniformity	0.749263	0.919445	0.406167	0.110612	0.3282	0.985508	0.540457	0.041862
encrypted_data_127.bin	0.490538|0.245269	0.061062|0.061062	0.817546|0.817546	0.496781|0.496781	0.810226|0.810226	0.611189|0.611189	0.513439|0.513439	0.213006|0.213006
success count of the 128	126	128	128	124	126	126	125	127

Figure 11: Chacha20((Phase 1 vs Phase 2) Statistical Test

Run Total	Run Distribution	Maximum 1 Run Test m=10000	Maximum 0 Run Test m=10000	Binary Deduction k=3	Binary Deduction k=7	Autocorrelation d=1	Autocorrelation d=2	Autocorrelation d=8
0	0	0	0	0	0	0	0	0
0.000000|0.000000	0.000000|0.000000	0.000000|0.000000	0.000000|0.000000	0.000000|1.000000	0.000000|0.000000	0.000000|0.000000	0.000000|1.000000	0.000000|0.000000
6	0	0	0	10	6	9	5	15
Run Total	Run Distribution	Maximum 1 Run Test m=10000	Maximum 0 Run Test m=10000	Binary Deduction k=3	Binary Deduction k=7	Autocorrelation d=1	Autocorrelation d=2	Autocorrelation d=8
0.669618	0.669618	0.945993	0.669618	0.28219	0.072289	0.669618	0.875539	0.556333
0.714590|0.357295	0.745584|0.745584	0.984422|0.984422	0.386917|0.386917	0.832174|0.583913	0.810905|0.405453	0.715665|0.357833	0.663165|0.331583	0.492997|0.246499
127	125	126	126	127	125	127	128	127

Figure 12: Chacha20((Phase 1 vs Phase 2) Statistical Test

Autocorrelation d=16	Matrix Rank Detection	Cumulative Sum Forward Detection	Cumulative Sum Backward Detection	Approximate Entropy m=2	Approximate Entropy m=5	Linear Complexity m=500	Linear Complexity m=1000	aurer Universal Statistical Test L=7 Q=12
0	0	0	0	0	0	0	0	0
0.000000|1.000000	0.000000|0.000000	0.000000|0.000000	0.000000|0.000000	0.000000|0.000000	0.000000|0.000000	0.000000|0.000000	0.000012|0.000012	0.000000|1.000000
3	0	2	2	0	0	2	0	14
Autocorrelation d=16	Matrix Rank Detection	Cumulative Sum Forward Detection	Cumulative Sum Backward Detection	Approximate Entropy m=2	Approximate Entropy m=5	Linear Complexity m=500	Linear Complexity m=1000	aurer Universal Statistical Test L=7 Q=12
0.809134	0.448892	0.733647	0.637119	0.3282	0.340461	0.059452	0.406167	0.572333
0.990650|0.495325	0.171130|0.171130	0.315216|0.315216	0.879297|0.879297	0.810323|0.810323	0.213800|0.213800	0.598511|0.598511	0.681561|0.681561	0.946864|0.473432
127	126	127	126	126	127	125	128	127

Figure 13: Chacha20((Phase 1 vs Phase 2) Statistical Test

AB
Discrete Fourier
0
0.000000|0.000000
0
Discrete Fourier
0.271449
0.403700|0.798150
126

Figure 14: Chacha20((Phase 1 vs Phase 2) Statistical Test

Algorithm	Type	Key size (bits)	Block size (bits)	AEAD mode
Ascon	Block cipher	80, 128	64	Yes
LED	Block cipher	64, 128	64	No
PHOTON	Hash function	N/A	N/A	No
SPONGENT	Hash function	N/A	N/A	No
PRESENT	Block cipher	80 , 128	64	No
CLEFIA	Block cipher	128, 192, 256	128	No
LEA	Block cipher	128, 192, 256	128	No
Grain-128a	Stream cipher	128	N/A	No
Enocoro-128v2	Stream cipher	128	N/A	No
Lesamnta-LW	Hash function	N/A	N/A	No
ChaCha	Stream cipher	128, 256	N/A	Yes
LBlock	Block cipher	80	64	No
SIMECK	Block cipher	64, 128	32, 48, 64	No
SIMON	Block cipher	64, 72, 96, 128, 144, 192	32, 48, 64, 96, 128	No
PRIDE	Block cipher	128	64	No
TWINE	Block cipher	80, 128	64	No
ESF	Block cipher	128	128	No

Table 1: Cryptography Algorithm Specifications (No Internal State or Initial Vector)

Algorithm	Rounds of use round function
Ascon	12 or 8
LED	48 or 32
PRESENT	31
CLEFIA	18, 22, 26
LEA	24, 28, or 32
SIMECK	32, 36, or 44
SIMON	32, 36, 42, 44, 52, 54, 68, 69, 72, 84
PRIDE	20
TWINE	36

Table 2: Rounds of Use Round Function for Cryptography Algorithms

Algorithm	Internal state size (bits)	Initial vector size (bits)
Ascon	320	128
Grain-128a	256	96
Enocoro-128v2	128	128
Lesamnta-LW	256	N/A
ChaCha	512	64 or 96

Table 3: Cryptography Algorithm Specifications (With Internal State and Initial Vector)

Algorithm	Type	Key size (bits)	Block size (bits)	AEAD mode
Ascon	Stream cipher	80 , 128	64	Yes
ChaCha20	Stream cipher	128, 256	32	Yes
XCR/Little_OaldresPuzzle_Cryptic	Stream cipher	128 or Custom	128 or Mutable	Yes(eg. Poly1305)

Table 4: The opposite algorithm we chose to compare with (Excluding Internal State and Initialization Vector Sizes, and Rounds of Use Round Function)

Algorithm	Internal State Size (bits)	Initialization Vector Size (bits)
Ascon	64	128
ChaCha20	512	64 or 96
XCR/Little_OaldresPuzzle_Cryptic	192	Not specified

Table 5: Internal State and Initialization Vector Sizes for The opposite algorithm we chose to compare with

Algorithm	**Rounds of use round function**
Ascon	12 or 8
ChaCha20	8, 12, or 20
XCR/Little_OaldresPuzzle_Cryptic	Custom

Table 6: Rounds of Use Round Function for The opposite algorithm we chose to compare with