AICE: An Arithmetic-Oriented Stream Cipher for Heterogeneous Computing

Bishwajit Chakraborty[2], Jiahui Gao[2], Kai Hu[1], Tao Huang[2], Zhongfeng Niu[3], Phuong Pham[2], Shuzhou Sun[2], Meiqin Wang[1], Shuang Wu[2], Wenhan Xu[2], Guang Zeng[2] and Chenxu Zhao[1]

[1] School of Cyber Science and Technology, Shandong University, Qingdao, China
kai.hu@sdu.edu.cn
[2] Shield Lab, Huawei Technologies Co., Ltd., China
huangtao80@huawei.com
[3] Independent Researcher

Abstract. Heterogeneous computing platforms increasingly rely on high-throughput data paths spanning CPUs and accelerators, yet most high-speed software ciphers are optimized primarily for CPU-centric execution models. We present AICE, an arithmetic-oriented stream cipher over $\mathbb{Z}/2^{16}\mathbb{Z}$ with a 37-word (592-bit) internal state, a nonlinear feedback combining modular addition, multiplication, bitwise OR, and rotation, a 370-round initialization with post-initialization key feed-forward, and periodic blank updates. We analyze AICE under several cryptanalytic models, including differential trail screening, linear approximation over the abelian group $\mathbb{Z}/2^{16}\mathbb{Z}$, guess-and-determine state recovery, and exact SMT-based cube evaluation, and find that all observable structural phenomena remain confined to reduced-round settings far below the full initialization. On the AI Cores of Huawei Ascend accelerators, AICE reaches a single-core peak throughput of 114.13 Gbps on the Ascend 950 and 55.36 Gbps on the Ascend 910B4, and scales to 1.59 Tbps on 32 Ascend 910B4 cores, roughly 54× the throughput of AES-CTR and 107× that of SM4-CTR on the same 32-core configuration; on the ARM Kunpeng 920 it remains in the same throughput class as hardware instruction accelerated AES-CTR.

Keywords: Stream cipher · Arithmetic-oriented design · Heterogeneous computing · CPU/NPU

1 Introduction

As modern computing workloads become increasingly distributed across heterogeneous hardware platforms, including CPUs, GPUs, NPUs, and other accelerator devices, the need to protect data transferred between such devices is growing rapidly. This trend arises in a wide range of settings, including accelerator-backed cloud services, data-center internal communication, secure storage pipelines, and emerging confidential or privacy-preserving AI systems. In all these scenarios, large volumes of data may move repeatedly across hardware boundaries, making efficient symmetric encryption along inter-device communication paths an increasingly important systems requirement.

In heterogeneous environments, cryptographic throughput becomes part of the end-to-end data path rather than a peripheral functionality. Intermediate tensors, activations, model parameters, or buffered data blocks may be transferred between processing units at high rate during computation. If encryption throughput does not scale with the underlying data movement and parallel execution of the platform, the cryptographic layer can become a bottleneck that limits overall system performance.

However, achieving high software performance for encryption across heterogeneous platforms remains challenging. Many high-speed symmetric ciphers and their most optimized implementations [WP14, SLN+22] have been developed primarily for CPU-centric execution models. In practice, peak software performance often depends on architectural features such as AES round instructions [Gue10], byte-permutation primitives, or word-level SIMD bitwise operations. While such mechanisms are highly effective on general-purpose processors, their fastest software paths frequently rely on instruction-set-specific primitives that are not uniformly available across different classes of accelerators.

This challenge is further amplified by the execution model of modern accelerator architectures. Whereas CPUs commonly exploit word-level SIMD and bitwise-friendly datapaths, many accelerator devices derive throughput from large-scale vector or tensor arithmetic pipelines optimized for data-parallel workloads. As a result, encryption algorithms whose core computation depends heavily on operations such as XOR [De 06, CP08, Ber08, HJM07, EJMY19, ETS11] or on architecture-specific primitives such as AES round instructions may not naturally map onto the throughput-dominant execution paths of all vector architectures. Even when existing ciphers can in principle be re-implemented for accelerator targets, their round structures have been designed around execution assumptions that do not always align with the arithmetic-oriented vector pipelines of accelerator devices, making cross-platform optimization a retrofit rather than a native fit.

Rather than pursuing per-platform re-optimization of existing designs, these observations motivate exploring a different design point for symmetric encryption. In particular, it is desirable to investigate primitives whose core operations align with vector-friendly arithmetic execution models and can be implemented efficiently across heterogeneous vector hardware without relying on architecture-specific cryptographic extensions. Such a goal emphasizes *performance portability*: rather than maximizing performance only on one processor family, the aim is to retain a high-throughput software path across diverse hardware platforms.

In this work, we explore this design direction and introduce AICE, a stream cipher designed with arithmetic-oriented vector execution in mind. AICE is built from a compact set of primitive operations including integer addition, multiplication, bitwise OR, and rotation-based diffusion. These operations admit efficient vectorized implementations across many platforms while avoiding heavy dependence on lookup tables, byte-shuffle mechanisms, or architecture-specific cryptographic instructions. The resulting design aims to provide scalable software throughput across heterogeneous computing environments while preserving the cryptographic structure required for practical cipher design.

AICE is motivated by this gap. Rather than optimizing exclusively for a CPU-centric execution model, it explores an arithmetic-oriented stream-cipher design in which the state update and keystream generation are expressed through regular word-level operations over $\mathbb{Z}/2^{16}\mathbb{Z}$. The goal is not to argue that traditional software ciphers are ineffective in general, but to investigate whether a stream cipher can be designed to remain implementation-friendly across both conventional processors and heterogeneous vector hardware. In this sense, AICE should be viewed as an exploration of performance-portable symmetric encryption for CPU–accelerator environments.

Our contributions are as follows:

1. We formulate a heterogeneous-friendly perspective on high-throughput symmetric encryption, emphasizing performance portability across CPU–accelerator platforms rather than optimization for a single processor family.

2. We present AICE, an arithmetic-oriented stream cipher built from a 37-word state over $\mathbb{Z}/2^{16}\mathbb{Z}$, together with its full specification and design rationale, including initialization, keystream extraction, post-initialization key feed-forward, and periodic

blank updates.

3. We provide a multi-angle security evaluation of AICE under several explicit models. This includes differential screening for nonce-induced initialization trails, guess-and-determine style state-recovery analysis, linear-correlation bounds for the initialization map, exact GF(2) post-initialization skeleton checks in the tested range, and exact SMT-based cube and balancedness analysis.

4. We implement AICE on both an AI-accelerator platform and several CPU platforms, and show that it achieves high throughput on the tested heterogeneous vector hardware while remaining competitive in conventional software environments.

The remainder of this paper is organized as follows. Section 2 presents the design goals and high-level structure of AICE. Section 3 gives the full specification. Section 4 explains the main design choices. Section 5 provides the security analysis. Section 6 describes the implementation and performance results. Section 7 concludes the paper.

2 Design Goals and Overview

This section provides a high-level overview of the design goals behind AICE.

2.1 Design Overview

The overall objective is to produce a stream cipher whose computational workload aligns naturally with modern vector-arithmetic execution models, including both general-purpose CPUs with SIMD extensions and AI accelerator cores, while providing sufficient nonlinear mixing and diffusion for cryptographic security. The design therefore emphasizes:

- **Arithmetic-oriented primitives.** The update function uses modular addition, modular multiplication, bitwise OR, and word rotation, all of which map efficiently onto integer ALUs and vector MAC units while avoiding lookup tables, byte permutations, and Galois-field arithmetic.
- **A regular shift-register structure.** The state update is organized as a one-position left shift with localized nonlinear feedback, concentrating complexity in a single feedback word while keeping the overall dataflow regular and vectorizable.
- **Conservative initialization.** The cipher applies 370 rounds of the same update function during initialization, followed by a key feed-forward step that injects additional key-dependent unknowns into the post-initialization state.
- **Periodic decimation.** During keystream generation, five additional blank updates are inserted after every 32 output blocks to increase the internal-state distance between successive extraction points.

2.2 Security Goals

AICE targets, 256-bit key security and 128-bit privacy security in the nonce-respecting single-key setting. Specifically:

- **Key recovery resistance.** Any key-recovery attack should require no fewer than 2^{256} operations.
- **State recovery resistance.** Recovering the full internal state from observed keystream should be infeasible without exhaustive key search.

We do not claim related-key security, and fault-injection resistance is not addressed in this work. These attack models are left for future investigation.

The detailed rationale for each design choice is discussed in Section 4.

3 Specification of AICE

This section specifies the internal state representation, notation, state update rule, initialization procedure, and keystream generation algorithm of the AICE stream cipher.

3.1 State and Notation

Unless otherwise specified, all arithmetic additions and multiplications are performed in the ring $\mathbb{Z}/2^{16}\mathbb{Z}$. Bitwise operations are applied to 16-bit words.

Let $x, y \in \mathbb{Z}/2^{16}\mathbb{Z}$. We use the following notation throughout the paper:

- $x + y$: addition modulo 2^{16};
- $x \cdot y$: multiplication modulo 2^{16};
- $x \oplus y$: bitwise XOR of x and y;
- $x \vee y$: bitwise OR of x and y;
- $x \lll i$: rotation of the 16-bit word x to the left by i bits, i.e.,

$$x \lll i = (x \ll i) \vee (x \gg (16 - i)),$$

  where the shift result is truncated to 16 bits.

The cipher maintains its internal state in two parts. The first part is a shift register of 37 words of 16 bits each,

$$S = (S_0, S_1, \ldots, S_{36}),$$

so that the register size is 592 bits. The second part is an *output memory* of 32 words of 16 bits each,

$$T = (T_0, T_1, \ldots, T_{31}),$$

of size 512 bits. The shift register drives the evolution of the cipher; the output memory is read and refreshed only by the keystream generation step, and it is initialized by a dedicated phase described in Section 3.5. When it is necessary to indicate the register after round r, we write

$$S^{(r)} = \big(S_0^{(r)}, S_1^{(r)}, \ldots, S_{36}^{(r)}\big).$$

The cipher uses a 256-bit secret key

$$K = K_0 \| K_1 \| \cdots \| K_{15},$$

where each K_i is a 16-bit word, and a 128-bit nonce

$$N = N_0 \| N_1 \| \cdots \| N_7,$$

where each N_i is a 16-bit word.

For encryption, the plaintext is parsed into 16-bit blocks

$$M = M_0 \| M_1 \| \cdots \| M_{t-1},$$

after zero padding if necessary, and the ciphertext is denoted by

$$C = C_0 \| C_1 \| \cdots \| C_{t-1}.$$

We write $|M|$ for the bit length of the original message M.

3.2 Round Constants

AICE uses a round-dependent rotation in the feedback path. The rotation constants form a period-37 sequence

$$\begin{aligned}\mathsf{RC} = (&11, 15, 15, 7, 12, 2, 13, 1, 6, 8, 1, 5, 14, 12, 10, 9, 5, 15, 4,\\ &7, 12, 11, 15, 13, 7, 4, 8, 1, 11, 14, 9, 8, 3, 5, 14, 10, 10).\end{aligned}$$

At round r, the rotation amount is $\mathsf{RC}[r \bmod 37]$.

3.3 State Update Function

The core state transformation of AICE is denoted by

$$S' = \mathsf{Upd}(S, r),$$

where $S = (S_0, \ldots, S_{36})$ is the input register, $S' = (S'_0, \ldots, S'_{36})$ is the updated register, and r is the round index.

For each round, AICE first computes a nonlinear feedback word through the following sequence:

$$\begin{aligned} t_0 &= S_0 + ((S_{15} \vee 1) \cdot S_{25}),\\ t_1 &= t_0 \lll \mathsf{RC}[r \bmod 37],\\ t_2 &= t_1 \oplus S_{22},\\ \mathsf{next} &= t_2 + S_{17}. \end{aligned}$$

All additions and the multiplication above are modulo 2^{16}. The operation $S_{15} \vee 1$ forces the least significant bit of S_{15} to one, so the multiplier entering the product is always odd.

The state update itself is a pure shift:

$$\begin{aligned} S'_j &= S_{j+1}, \qquad 0 \le j \le 35,\\ S'_{36} &= \mathsf{next}. \end{aligned}$$

All rotation is confined to the feedback path: the shift register itself performs a pure left shift, and the round constants determine how the feedback word is rotated before the XOR and the final modular addition.

3.4 Intermediate Output Function

At a given register state S, the *intermediate output word* is computed as

$$\begin{aligned} A &= S_1 + S_5 + S_{19} + S_{32},\\ Y &= A \oplus S_{30}, \end{aligned}$$

where A is computed modulo 2^{16}. The intermediate output Y is never published directly. Instead, it is combined with the output memory to form the keystream, as specified in Section 3.6, and during initialization it is used to seed the output memory.

3.5 Initialization

Given a key $K = K_0 \| \cdots \| K_{15}$ and a nonce $N = N_0 \| \cdots \| N_7$, the initial register is loaded as

$$\begin{aligned} (S_0, \ldots, S_7) &= (N_0, \ldots, N_7),\\ (S_8, \ldots, S_{23}) &= (K_0, \ldots, K_{15}),\\ (S_{24}, \ldots, S_{36}) &= \mathsf{const}, \end{aligned}$$

where const denotes the fixed 13-word constant array

$$\mathsf{const} = (\texttt{0x243F}, \texttt{0x6A88}, \texttt{0x85A3}, \texttt{0x08D3}, \texttt{0x1319}, \texttt{0x8A2E}, \texttt{0x0370},$$
$$\texttt{0x7344}, \texttt{0xA409}, \texttt{0x3822}, \texttt{0x299F}, \texttt{0x31D0}, \texttt{0x082E}).$$

Initialization then proceeds in three phases.

Diffusion phase. The loaded register is diffused by applying the update function 333 times:

$$S \leftarrow \mathsf{Upd}(S, r) \qquad \text{for } r = 0, \ldots, 332.$$

Key feed-forward. The key is then injected again into the register using a full-coverage feed-forward pattern:

$$\begin{aligned} S_i &\leftarrow S_i + K_i, && 0 \le i \le 15, \\ S_{i+16} &\leftarrow S_{i+16} + K_i, && 0 \le i \le 15, \\ S_{i+32} &\leftarrow S_{i+32} + K_i, && 0 \le i \le 4. \end{aligned}$$

This key feed-forward covers all 37 register positions.

Output-memory initialization. The next 37 updates fill the output memory. During the first 32 of these updates the intermediate output of the current register is taken; we denote these 32 intermediate outputs $Y_{-32}, \ldots, Y_{-1}$. They are stored in the output memory, $T_i \leftarrow Y_{i-32}$ for $i = 0, \ldots, 31$, but are never published. The last 5 updates produce no output:

$$S \leftarrow \mathsf{Upd}(S, r) \qquad \text{for } r = 333, \ldots, 369.$$

After this phase the register has been updated 370 times in total. The negative indices $Y_{-32}, \ldots, Y_{-1}$ are chosen so that the keystream relation of Section 3.6 applies uniformly from the first output word on.

3.6 Keystream Generation and Encryption

After initialization, AICE produces keystream in cycles of 32 words. Let $Y_0, Y_1, Y_2, \ldots$ denote the intermediate outputs produced after initialization at successive *output steps*: within each 37-step cycle the first 32 steps each produce one intermediate output, and the last 5 steps update the register without producing output.

The keystream word at output step i is the modular sum of the current intermediate output and the intermediate output produced one cycle — that is, 32 output steps — earlier:

$$Z_i = Y_i + Y_{i-32} \pmod{2^{16}}, \qquad i \ge 0.$$

For $i \ge 32$ both terms are public-phase intermediate outputs; for $0 \le i < 32$ the term Y_{i-32} is one of the secret intermediate outputs $Y_{-32}, \ldots, Y_{-1}$ produced during the output-memory initialization phase (Section 3.5). Hence every keystream word mixes two intermediate outputs that are exactly one 37-step cycle apart, and no intermediate output is ever published on its own.

This 32-step look-back is realized by the output memory: when output step i is processed, $T_{i \bmod 32}$ holds Y_{i-32}, so the cipher emits $Z_i = Y_i + T_{i \bmod 32}$ and then refreshes $T_{i \bmod 32} \leftarrow Y_i$. The register is updated once after each output step, and five further times after every 32 output steps, using the period-37 round-constant schedule with the round index continuing from 370.

The message is encrypted block-wise. Let the padded plaintext be $M_0 \| M_1\| \cdots \| M_{t-1}$, where $t = \lceil |M|/16 \rceil$. If the message length is not a multiple of 16, the final block is padded

with zeros before encryption, and the final ciphertext is truncated back to the original bit length $|M|$. The ciphertext block is

$$C_i = M_i + Z_i.$$

If the message is empty, the encryption phase is skipped.

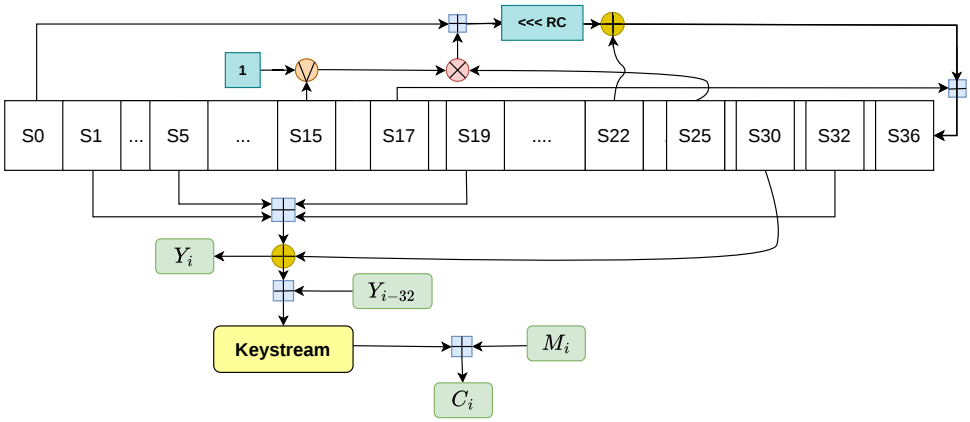

Figure 1: Overall structure of the AICE stream cipher.

4 Design Rationale

4.1 State Size and Register Length

The shift register of AICE consists of 37 words of 16 bits each, resulting in a register size of 592 bits. The choice of 37, a prime number, helps avoid simple positional symmetries and repeated alignment patterns that may arise more naturally in register lengths with small divisors.

A moderately large register increases the complexity of the state evolution and reduces the risk of structural weaknesses associated with short feedback registers. It also allows the feedback positions and the keystream extraction positions to be distributed more widely across the register, thereby reducing direct structural relations between internal state words and output words. From an implementation perspective, a register of this size remains practical on modern vector-oriented processors, which typically provide sufficient register resources for handling multiple 16-bit words in parallel.

In addition to the shift register, AICE maintains a 32-word output memory T (512 bits), which is used only by the keystream generation step. The role of the output memory is discussed in Section 4.5.

4.2 Choice of Primitive Operations

The nonlinear feedback function of AICE combines modular addition, integer multiplication, bitwise OR, and word rotation. These operations were selected based on both cryptographic and implementation considerations.

From a cryptographic perspective, integer multiplication introduces nonlinear cross-bit dependencies between state words. Modular addition further increases algebraic complexity through carry propagation. The bitwise OR is used to force the least significant bit of the multiplier to one, i.e. the term $S_{15} \vee 1$: an odd multiplier is a unit modulo 2^{16}, so the

product $(S_{15} \vee 1) \cdot S_{25}$ never collapses onto a multiplier with reduced order, which would otherwise zero out low-order bits of the feedback word. Taken together, these operations produce a feedback relation with substantial nonlinear mixing capability while remaining purely arithmetic.

Using modular multiplication as the nonlinear core has precedent. Daemen, Govaerts, and Vandewalle's MMB block cipher [DGV93] placed multiplication modulo $2^{32} - 1$ at its center, building on an explicit study of the propagation properties of multiplication modulo $2^n - 1$ [DLGV92, Dae95] and following the multiplication modulo $2^{16} + 1$ of IDEA [LM91, LMM91]. The algebraic structure of that operation was later turned against the design: Wang, Nakahara, and Sun exploited properties of multiplication in $\mathbb{Z}_{2^{32}-1}$ to break the full MMB [WNS09], followed by practical key-recovery attacks [JCWW12, AD13], in the same spirit as the multiplicative-differential analysis of multiplication-based ciphers [BCJW02, Fur02, Yil20]. AICE differs from these designs in two respects: it multiplies modulo 2^{16}, a power of two rather than $2^n \pm 1$, and it forces the multiplier to an odd unit through $S_{15} \vee 1$, avoiding the cyclic structure of $\mathbb{Z}_{2^n-1}$ that enabled the MMB attacks.

General symmetric algorithms often employ table lookups, multiplications over Galois fields, or byte permutations to construct nonlinear round functions. In the design of AICE, we eliminate these conventional choices, as they represent performance bottlenecks or cache-timing vulnerabilities on AI Core architectures. Instead, the nonlinear behavior is realized using modular integer arithmetic (multiplication and addition modulo 2^{16}), bitwise OR with a constant, and rotation. These operations are naturally supported and highly efficient on processors optimized for neural computation.

The feedback path also applies a round-dependent left rotation, governed by the period-37 constant sequence RC. Because the rotation amount varies from round to round, a given bit position is not repeatedly aligned with the same target position across successive rounds, so bit dependencies are redistributed and intra-word diffusion is improved. The rotation period 37, equal to the register length and prime, prevents the rotation schedule from aligning with the register period.

4.3 Feedback and Shift-Register Update Structure

AICE adopts a shift-based state update mechanism. In each round, all register words are shifted by one position, and a single new word, produced by the nonlinear feedback function, enters at the end of the register:

$$\begin{aligned} S_j^{\text{new}} &= S_{j+1} \quad \text{for } 0 \le j \le 35, \\ S_{36}^{\text{new}} &= \mathsf{next}, \end{aligned}$$

where the feedback word is

$$\mathsf{next} = \big((S_0 + (S_{15} \vee 1) \cdot S_{25}) \lll \mathsf{RC}[r \bmod 37]\big) \oplus S_{22} \; + \; S_{17}.$$

This structure offers three advantages: it is regular and easy to implement; it concentrates the nonlinear computation in a single feedback word; and it allows information to propagate gradually across the entire register through repeated updates.

All rotation is confined to the feedback path; the register itself performs a pure left shift, with no specially rotated register position. The feedback word combines four ingredients before it enters the register: the modular sum $S_0 + (S_{15} \vee 1) \cdot S_{25}$, a round-dependent rotation, an XOR with S_{22}, and a final modular addition of S_{17}. The XOR with S_{22} interleaves a Boolean operation between the two arithmetic stages, so that a single feedback word already mixes the additive and the bitwise domains.

4.4 On the Design of Initialization

During initialization, the nonce and key are first loaded into the register and then processed through multiple update rounds. Rather than introducing a separate initialization transformation, AICE reuses the same update rule that is later employed during keystream generation. This choice keeps the overall design uniform and avoids the need for an additional dedicated initialization circuit.

The number of update rounds is selected to ensure complete diffusion, so that even a one-bit change in the key or nonce results in different values across all words of the register. When introducing single-bit perturbations in the key or nonce, the resulting differences were observed to spread across the entire 37-word register within approximately 80 rounds. Initialization applies a 333-round diffusion phase and, after the key feed-forward, a 37-step output-memory initialization phase (Section 4.5); the register is updated 370 times in total. This provides a safety margin of more than $4\times$ relative to the empirically observed diffusion threshold, while also providing substantial security margins against differential and linear attacks during initialization, as detailed in Section 5.

4.5 Output Memory and Keystream Masking

AICE does not publish the intermediate output word directly. Instead, the keystream word at output step i is the modular sum of the current intermediate output and the one produced exactly one 37-step cycle earlier,

$$Z_i = Y_i + Y_{i-32} \pmod{2^{16}}.$$

The cipher forms this sum with the help of the output memory T, which holds the 32 most recent intermediate outputs: T supplies Y_{i-32} when Z_i is computed and is then refreshed with Y_i.

This output layer is motivated by two considerations. First, the individual intermediate output Y is never exposed: an attacker observes only a modular combination of two internal outputs, which adds a masking layer to the keystream and enlarges the hidden quantity that a state-recovery attack must pin down, since the 32-word output memory becomes part of the secret state. Second, the modular addition is an arithmetic operation consistent with the rest of the design and introduces carry-based interaction between the two combined outputs.

The output memory is seeded by a dedicated hidden phase of initialization: the 32 secret intermediate outputs $Y_{-32}, \ldots, Y_{-1}$ produced during the output-memory initialization phase are written into $T_0, \ldots, T_{31}$ but are never published. Because these seed values are themselves secret, the keystream never exposes a freshly generated intermediate output in isolation, not even at the first output word.

4.6 Key Feed-Forward

After the 333-round diffusion phase, the key words are added again into the register. This extra feed-forward step, used previously in [SLN+21, HT16], hardens the cipher against state-recovery attacks by injecting additional key-dependent unknowns. The injection covers all 37 register positions: $K_0, \ldots, K_{15}$ are added to $S_0, \ldots, S_{15}$ and again to $S_{16}, \ldots, S_{31}$, and $K_0, \ldots, K_4$ are added to $S_{32}, \ldots, S_{36}$. Full coverage ensures that every position read by the intermediate output function $(S_1, S_5, S_{19}, S_{30}, S_{32})$, as well as every feedback operand, carries a key-dependent unknown before keystream generation begins, leaving no unprotected backward-tracing window. Because the key feed-forward precedes the output-memory initialization, the output memory and every public output are functions of the post-feed-forward state alone, so a state-recovery attack operates entirely past the key feed-forward.

Without the key feed-forward, an attacker who recovers a sufficiently large portion of the post-initialization register could attempt to trace the recurrence backwards. The update function is invertible given the register: from $S_{36}^{(t)}$ and the shifted words one can write

$$S_{36}^{(t)} = \big((S_0^{(t-1)} + (S_{14}^{(t)} \vee 1) \cdot S_{24}^{(t)}) \lll \mathsf{RC}[\cdot]\big) \oplus S_{21}^{(t)} \; + \; S_{16}^{(t)},$$

which can be solved for the discarded word $S_0^{(t-1)}$, and similarly for earlier discarded words. By iterating this inversion an attacker could algebraically unravel the initialization. With full-coverage key injection, every register word appearing in these equations contains an additive key-dependent unknown, so an attacker would need to guess key words at every position encountered during backward tracing. In the adopted guess-and-determine framework (Section 5), this additional key injection removes the structured backward-recovery path observed for the weakened construction, confirming the intended hardening effect.

4.7 Keystream Decimation

During encryption, the register is updated once after each keystream extraction. In addition, after every 32 output words, the register is updated 5 extra times without producing keystream. The purpose of this periodic decimation is to increase the internal-state distance between nearby output words. By inserting additional update steps at regular intervals, AICE increases the separation between successive extraction points and reduces the risk that short-range correlations between neighboring outputs can be exploited. The chosen decimation frequency reflects a practical compromise between additional state mixing and throughput overhead. The same 32-word period also defines the cycle structure of the output memory.

4.8 Intermediate Output and Keystream Extraction Positions

The intermediate output word is computed as

$$Y = (S_1 + S_5 + S_{19} + S_{32}) \oplus S_{30}.$$

Four register words are first combined by modular addition, and the result is then masked by an XOR with a fifth word, S_{30}. The four additive taps are drawn from positions spread across the register, so that the output mixes words at widely separated positions rather than a localized window. Combining four taps by modular addition keeps the output extraction consistent with the arithmetic-oriented design philosophy of the feedback function, while the final XOR with S_{30} mixes the additive and the Boolean domains in the output itself, so that any linear or differential relation on the output must cross both structures.

The intermediate output Y is not the published keystream word; as described in Section 4.5, the keystream word is $Z_i = Y_i + Y_{i-32}$, the modular sum of the current intermediate output and the one produced one 37-step cycle earlier.

5 Security Analysis

In this section, we analyze AICE under several cryptanalytic viewpoints relevant to stream ciphers. Our goal is to provide evidence under clearly stated attack models, rather than unconditional security claims.

5.1 Differential Security of the Initialization

We bound the probability of differential trails through the 370-round initialization of AICE in the fixed-key, chosen-nonce setting: the key is secret and fixed, and the attacker may freely choose the 128-bit nonce; related-key attacks are out of scope. Since exhaustive key recovery costs 2^{256}, a differential trail is exploitable only if its probability exceeds 2^{-256}. We show that the best trail probability is far below this bound. We bound the probability of differential trails through the 370-round initialization of AICE in the fixed-key, chosen-nonce setting; related-key attacks are out of scope. Since the attacker can only vary the 128-bit nonce, a differential trail is exploitable only if its probability exceeds 2^{-128}. We show that the best trail probability is far below this bound.

5.1.1 Differential Model

Differences are tracked as *additive* (modular-addition) differences and bounded by modeling the differential propagation as a CP problem and then solving it using minizinc solver's CP-SAT tool [SHW+14]. The model is built so that it can only be favorable to the attacker; hence the weight it returns is a valid lower bound on the minimum trail weight of AICE.

Relaxation. Every XOR in the feedback function is replaced by a modular addition. Modular addition admits every XOR difference transition and, through its carry structure, additional ones [LM02]; replacing XOR by addition can therefore only lower the minimum trail weight. The weight obtained on this relaxed cipher is thus a lower bound on that of AICE. In Appendix A, we also provide an analysis of the truncated differential propagation.

Nibble difference distribution tables. Each 16-bit word is split into four 4-bit nibbles, and every feedback operation is described by a nibble-level difference distribution table (DDT). The combined add-and-multiply step on two 4-bit operands is treated as an S-box $(X, Y) \mapsto (Z, C_1, C_2)$, whose 12-bit output is the result nibble Z and two carry nibbles; its DDT is computed by exhaustive enumeration, and the 16-bit modular addition and oddized multiplication are assembled from these S-boxes by propagating the carry nibbles. The round-dependent rotation is treated as a nibble S-box $(X_1, \ldots, X_4) \mapsto (Y_1, \ldots, Y_4)$, whose activity DDT is computed for every rotation amount $1, \ldots, 15$. The CP minimizes the total trail weight $-\log_2 \Pr$ over the modeled rounds.

5.1.2 Minimum Trail Weight

The full 370-round model is too large to solve in one instance, so the bound is assembled from sub-ciphers. Since the weight of a differential trail is additive across a composition of functions, the minimum weight of a composed function is at least the sum of the minimum weights of its parts.

The 370-round initialization is a 74-round prefix followed by a 296 round blocks. With the difference confined to the eight nonce words $S_0, \ldots, S_7$, the prefix has minimum weight 123. The following 296-round initialization can be viewed as two blocks of rounds 0–148 whose minimum weight is solved to be 116; one 296-round *update block* therefore has minimum weight at least $116 \times 2 = 232$. This gives an overall minimum weight for the initialization:

$$w_{\min} \;\geq\; 123 + 232 \;=\; 355.$$

The best differential trail through the initialization therefore has probability at most 2^{-355}, which is 227 bits below the 2^{-128} nonce space bound Because this figure is obtained on a model strictly weaker than AICE, the true margin is expected to be larger.

Table 1: Lower bound on the minimum differential trail weight of the AICE initialization, in the relaxed XOR-as-addition nibble model.

Initialization segment	Rounds	Min. trail weight
unrestricted differential	148	116
unrestricted differential	111	84
nonce restricted differential	74	123
Differential after 370 rounds	370	355

5.2 Guess-and-Determine State-Recovery Analysis

We evaluate guess-and-determine (GD) attacks against AICE by giving the attacker a set of public keystream words and asking whether a small number of guessed internal words is sufficient to determine the remaining hidden state. The experiment is intentionally formulated as a state-recovery problem after initialization and key feed-forward, rather than as a key-recovery problem. The hidden variables are the 37 post-feed-forward state words and the 32 output memory words,

$$(S_0, \ldots, S_{36}, T_0, \ldots, T_{31}) \in (\mathbb{Z}/2^{16}\mathbb{Z})^{69}.$$

Thus the SMT instance contains 69 16-bit words, or 1104 hidden bits. The attacker is assumed to know the nonce and the public output words, but the SMT model starts from the post-feed-forward state and treats the output memory as unknown. This is an attacker-favorable formulation because it does not require the solver to invert the full initialization or key-feed-forward computation.

For each 37-step output cycle, only the first 32 steps produce public words. Let $Y_{j,i}$ be the intermediate output produced in cycle j and position i, where $0 \le i < 32$. The public output equation is

$$Z_{j,i} = Y_{j,i} + T_{j,i} \pmod{2^{16}},$$

where

$$Y_{j,i} = (S_1 + S_5 + S_{19} + S_{32}) \bmod 2^{16} \oplus S_{30}$$

is evaluated on the current state word vector. After producing $Z_{j,i}$, the memory word is updated as $T_{j+1,i} = Y_{j,i}$, and the state is updated normally. The final five steps of each 37-step cycle update only the state and produce no public output. Consequently, for all cycles after the first one, the public output can also be viewed as a modular sum of two raw intermediate outputs at the same position in two consecutive cycles:

$$Z_{j,i} = Y_{j,i} + Y_{j-1,i} \pmod{2^{16}}, \qquad j \ge 1.$$

SMT model. The GD validator encodes the state update, the intermediate output function, the output-memory update, and all modular additions as 16-bit bit-vector constraints in Z3. Given n public output words, the model introduces the initial symbolic words $S_0, \ldots, S_{36}$ and $T_0, \ldots, T_{31}$, unfolds the generator for enough 37-step cycles to cover the n outputs, and adds one constraint

$$Y_t + T_t = Z_t \pmod{2^{16}}$$

for each observed word. A GD candidate is specified by a set $G \subseteq \{0, \ldots, 36\}$ of state-word positions. For a tested assignment, the solver also receives constraints $S_g = s_g$ for all $g \in G$. In the validation experiments, the correct assignment is taken from the generated test instance to measure whether the remaining hidden state is determined efficiently. Wrong assignments are tested separately to estimate the rejection cost of non-solution branches.

We restrict the guess set to state words, not memory words. This matches the intended GD attack structure: once the post-feed-forward state trajectory is fixed, the first public output block determines the initial memory values by

$$T_{0,i} = Z_{0,i} - Y_{0,i} \pmod{2^{16}}.$$

The remaining public outputs then constrain the same state trajectory through the cross-cycle equations.

Guess-set search. The candidate guess sets are selected by a lightweight heuristic search before the exact SMT validation. The score function symbolically propagates which state words become known through the output equations and through the feedback update, rewards early determination of words that participate in future output and feedback computations, and penalizes unresolved nonlinear couplings. This screening stage is only used to propose promising sets G; every reported attack candidate is then checked by the exact SMT model described above.

For AICE, the best validated 28-word state guess set is

$$G_{28} = \{1, 3, 4, 6, 7, 8, 9, 11, 13, 16, 18, 19, 20, 21, \\ 22, 23, 24, 25, 26, 28, 29, 30, 31, 32, 33, 34, 35, 36\}.$$

With 128 public output words and the correct values of these 28 state words, Z3 returns a satisfying assignment that matches the planted post-feed-forward state and output memory. The recorded solving time is 85.03 seconds; an independent rerun gives 92.19 seconds. This confirms that, conditioned on the correct 28 guessed words, the remaining 41 hidden words are efficiently determined by the public output constraints.

Wrong-guess rejection cost. The cost of a GD attack is governed by wrong branches, not by the single correct-guess run. To measure this cost, we fixed the same 28 positions G_{28} to random incorrect 16-bit values and ran the same 128-output SMT model with a timeout of 1800 seconds. All 10 trials were fully rejected as UNSAT. The average UNSAT time was 153.1 seconds, with a minimum of 133.16 seconds and a maximum of 172.17 seconds. Therefore, in this implementation, an incorrect 28-word branch is not eliminated by a cheap syntactic test; it requires roughly 153 seconds of SMT solving on average.

Experiment	Outputs	Timeout	Result
Correct G_{28} assignment	128	300 s	SAT, 85.03 s, verified
Wrong G_{28} assignments	128	1800 s	10/10 UNSAT, avg. 153.1 s

Search below 28 guessed words. We also tested whether the number of guessed state words could be reduced to 27. The AICE model was tested with 128 and 160 public outputs. For the best 27-word candidate produced by the heuristic search, the 128-output and 160-output runs both timed out after 900 seconds. We then reran the 160-output instance with a 6000-second timeout and the same fixed 27-word candidate; this also timed out without finding a satisfying assignment.

Guessed words	Outputs	Timeout	Result
27	128	900 s	TIMEOUT
27	160	900 s	TIMEOUT
27	160	6000 s	TIMEOUT
28	128	300 s	SAT, verified

These experiments place the current practical SMT threshold at 28 guessed state words for the tested AICE instance and attack strategy. We did not find a working 27-word GD attack, even after increasing both the amount of public keystream and the solver timeout.

Interpretation. The validated result is a conditional state-recovery attack: if the attacker guesses the 28 selected 16-bit state words correctly, the remaining state and memory can be recovered by SMT solving in about 90 seconds for the tested instance. Turning this into a complete attack would require enumerating the assignments to these 28 words, namely $2^{28\cdot 16} = 2^{448}$ possible branches in the direct GD formulation. The wrong-guess timing shows that each incorrect branch costs about 153 seconds on average in our implementation. Thus the fast correct-guess solve should not be interpreted as the cost of the full attack. The relevant conclusion is that the best validated state-recovery attempt against AICE requires guessing 28 of the 37 state words and has expensive wrong-branch rejection, so the feasible GD guess size is large and a complete GD search stays far above the cost of exhaustive key search.

5.3 Linear Analysis over $\mathbb{Z}_{2^{16}}$

This section explores linear cryptanalysis of AICE. Since the cipher is defined over $\mathbb{Z}/2^{16}\mathbb{Z}$, the natural framework uses additive characters

$$\chi_\gamma(x) = \exp(2\pi i\gamma x/2^{16}) := \omega^{\gamma x}$$

where $\omega = \exp(2\pi i/2^{16})$ is a primitive 2^{16}-th root of unity and $\gamma \in \mathbb{Z}/2^{16}\mathbb{Z}$. Such a generalized linear cryptanalysis has been explored deeply in papers like [Bey21, BSV07, NSYW26].

5.3.1 Linear Approximation over $\mathbb{Z}/2^{16}\mathbb{Z}$ and Propagation Rules

The correlation over $\mathbb{Z}/2^{16}\mathbb{Z}$ is defined as follows:

Definition 1. ([Bey21]) Let $F : G \to H$ be a function between two Abelian groups G, H, and ψ_1, ψ_2 be two group characters of H and G respectively. The correlation of the linear approximation (ψ_1, ψ_2) is

$$C^F_{\psi_1,\psi_2} = \frac{1}{|G|}\sum_{x\in G}\overline{\psi_1(F(x))}\psi_2(x)$$

where $\overline{\psi_1(F(x))}$ denotes the complex-conjugate of $\psi_1(F(x))$.

For AICE, we have $G = H = \mathbb{Z}/2^{16}\mathbb{Z}$, and the characters are of the form $\chi_\alpha(x) = \omega^{\alpha x}$ for $\alpha \in \mathbb{Z}/2^{16}\mathbb{Z}$. Therefore, we can write $\psi_1 = \chi_\beta$ and $\psi_2 = \chi_\alpha$ for some $\alpha, \beta \in \mathbb{Z}/2^{16}\mathbb{Z}$, and the correlation is also denoted and computed by

$$C^F_{\beta,\alpha} = 2^{-16}\sum_{x\in\mathbb{Z}/2^{16}\mathbb{Z}}\omega^{\alpha x-\beta F(x)}.$$

There are 4 different kinds of component operations in AICE: the modular addition ($\boxplus$), COPY (aka. SPLIT), rotational shift ($\lll k$ for $0 \le k < 16$), and the OR-1 multiplication ($F(x, y) = (x \vee 1) \otimes y$).

Among the four operations, the modular addition and COPY are linear ones. Their correlation propagation rules are well-known and can be found in the literature, for example in [Bey21].

$$C^{\boxplus}_{\beta,(\alpha_0,\alpha_1)} = \begin{cases} 1 & \text{if} \quad \beta = \alpha_0 = \alpha_1 \\ 0 & \text{otherwise.} \end{cases}, \quad C^{\text{COPY}}_{(\beta_0,\beta_1),\alpha} = \begin{cases} 1 & \text{if} \quad \beta_0 + \beta_1 = \alpha \\ 0 & \text{otherwise.} \end{cases}.$$

However, the rotational shift and the OR-1 multiplication are non-linear operations, and their correlation propagation rules are not known in the literature, as far as we know. Thus, we first construct the propagation rules for them, and then we can analyze the linear approximation of AICE based on these rules.

Correlation of Rotational Shift. Consider the operation $\lll k$ on a variable $x \in \mathbb{Z}/2^n\mathbb{Z}$. Denote the higher k bits and the lower $n-k$ bits of x by x_h and x_l respectively, i.e., $x = 2^{n-k}x_h + x_l$; thus

$$x \lll k = (2^{n-k}x_h + x_l) \lll k = 2^k x_l + x_h.$$

According to Definition 1,

$$\begin{aligned} C_{a,b}^{\lll k} &= 2^{-n} \sum_{0\le x<2^n} \omega^{bx-a(x\lll k)} \\ &= 2^{-n} \sum_{0\le x_h<2^k} \sum_{0\le x_l<2^{n-k}} \omega^{b(x_h\cdot 2^{n-k}+x_l)-a(x_h+x_l\cdot 2^k)} \\ &= 2^{-k} \sum_{0\le x_h<2^k} \omega^{x_h(b2^{n-k}-a)} \cdot 2^{k-n} \sum_{0\le x_l<2^{n-k}} \omega^{x_l(b-a2^k)} \end{aligned}$$

The two "sum"s in the above formula are two elementary geometric series, so with the following two facts, we can compute $C_{a,b}^{\lll k}$ easily with given input and output masks.

$$\sum_{0\le x_h<2^k} \omega^{x_h(b2^{n-k}-a)} = \begin{cases} 2^k & \text{if} \quad b2^{n-k}-a=0 \\ \frac{1-\omega^{-a2^k}}{1-\omega^{b2^{n-k}-a}} & \text{otherwise} \end{cases} \tag{1}$$

$$\sum_{0\le x_l<2^{n-k}} \omega^{x_l(b-a2^k)} = \begin{cases} 2^{n-k} & \text{if} \quad b-a2^k=0 \\ \frac{1-\omega^{-b2^{n-k}}}{1-\omega^{a2^k-b}} & \text{otherwise} \end{cases} \tag{2}$$

It is easy to verify that the input and output masks are either both zero or both nonzero, if we want to have a nonzero correlation. For single-bit masks 2^i (including the zero mask), the rotation $\lll k$ approximately maps $2^i \to 2^{(i-k) \bmod n}$ with $|C| \approx 1$ (more precisely, $|C| = |\cos(\pi a/2^n)| \cdot |\cos(\pi b/2^n)|$-type factor, which differs from 1 by $O(2^{-n})$ and is therefore numerically indistinguishable from 1 for $n = 16$); the zero mask maps to itself: $0 \to 0$ with $|C| = 1$ exactly.

Correlation of the OR-1 Multiplication. Let $H = \mathbb{Z}/2^{16}\mathbb{Z}$, (a, b) and c be the masks of the OR-1 multiplication $F(x, y) = (x \vee 1) \otimes y$. Writing $x = 2x' + r$ with $r \in \{0, 1\}$ and $0 \le x' < 2^{n-1}$, we have $x \vee 1 = 2x' + 1$ regardless of the parity bit r, hence

$$\begin{aligned} C_{c,(a,b)}^F &= 2^{-2n} \sum_{x,y\in H\times H} \omega^{ax+by-c((x\vee 1)\otimes y)} \\ &= 2^{-2n} \sum_{y\in H} \omega^{(b-c)y} \sum_{r\in\{0,1\}} \omega^{ar} \sum_{x'=0}^{2^{n-1}-1} \omega^{2x'(a-cy)} \\ &= \frac{1+\omega^a}{2N} \sum_{y\in H} \omega^{(b-c)y} \cdot \mathbb{1}\left[a \equiv cy \pmod{2^{n-1}}\right], \end{aligned}$$

where in the last step we used $\sum_{x'=0}^{2^{n-1}-1} \omega^{2x'(a-cy)} = (N/2)\, \mathbb{1}[a \equiv cy \pmod{2^{n-1}}]$ (orthogonality on $\mathbb{Z}/2^{n-1}\mathbb{Z}$). The notation $\mathbb{1}[\text{condition}]$ is the indicator function, which equals 1 if the condition is true and 0 otherwise.

Let $d' = \gcd(c, 2^{n-1})$. The congruence $cy \equiv a \pmod{2^{n-1}}$ has solutions iff $d' \mid a$, in which case its solutions in H form the arithmetic progression $\{y_0 + j \cdot 2^{n-1}/d' : 0 \le j < 2d'\}$ for some particular solution y_0. Summing the resulting geometric series yields

$$C_{c,(a,b)}^F = \begin{cases} \dfrac{d'\,(1+\omega^a)}{N}\,\omega^{(b-c)y_0} & \text{if } d' \mid a \text{ and } 2d' \mid (b-c), \\ 0 & \text{otherwise.} \end{cases}$$

In particular, when the support condition holds, $|C^F_{c,(a,b)}| = (2d'/N)\cdot\left|\cos(\pi a/2^n)\right|$, attaining its maximum value of 1 at $(a,b,c) = (0, 2^{n-1}, 2^{n-1})$.

Correlation of the XOR operation. Let $F(x,y) = x \oplus y$ denote the bitwise XOR, viewed as a function $(\mathbb{Z}/2^n\mathbb{Z})^2 \to \mathbb{Z}/2^n\mathbb{Z}$. Writing $x = \sum_{i=0}^{n-1} x_i 2^i$ and $y = \sum_{i=0}^{n-1} y_i 2^i$ with $x_i, y_i \in \{0,1\}$, the exponent $ax + by - c(x \oplus y)$ decomposes bit-by-bit, so the double sum factors as a product over bit positions:

$$C^{\oplus}_{c,(a,b)} = 2^{-2n} \sum_{x,y \in H\times H} \omega^{ax+by-c(x\oplus y)} = \prod_{i=0}^{n-1} \frac{1}{4} \sum_{x_i,y_i\in\{0,1\}} \omega^{2^i(ax_i+by_i-c(x_i\oplus y_i))}.$$

Evaluating each factor over the four (x_i, y_i) pairs (noting $x_i \oplus y_i = 0$ for $(0,0)$ and $(1,1)$, and 1 for $(1,0)$ and $(0,1)$) gives

$$C^{\oplus}_{c,(a,b)} = \prod_{i=0}^{n-1} \frac{1 + \omega^{2^i(a-c)} + \omega^{2^i(b-c)} + \omega^{2^i(a+b)}}{4}.$$

Each factor has modulus at most 1, with equality iff $2^{n-i} \mid (a-c)$ and $2^{n-i} \mid (b-c)$ (so that all four terms align to 1). In particular, $|C^{\oplus}_{c,(a,b)}| = 1$ iff $a \equiv b \equiv c \pmod{2^n}$, and $C^{\oplus}_{c,(a,b)} = 0$ whenever any factor vanishes.

5.3.2 Correlation of the initialization phase

Intuitively, the rotational shift obstructs most mask propagations under the additive-character framework over $\mathbb{Z}/2^{16}\mathbb{Z}$. However, when the input mask is a single-bit mask of the form 2^i, the rotation $\lll k$ deterministically maps it to another single-bit mask $2^{(i-k) \bmod 16}$ with correlation exactly 1. In this case, both geometric series in the correlation formula simultaneously reach their maximum value, yielding $|C^{\lll k}_{2^{(i-k) \bmod 16},2^i}| = 1$. Therefore, in the linear cryptanalysis of AICE, considering the single-bit mask propagations should give a decent security guranteen.

We first consider the XOR operation. For single-bit masks, the correlation magnitude is shown in Table 2.

Table 2: Correlation of $F(x,y) = x \oplus y$ for single-bit masks (including zero mask). Only diagonal entries have nonzero correlation.

Input x	Input y	Output	$\lvert C_{c,(a,b)}\rvert$
0	0	0	1
2^{15}	2^{15}	2^{15}	1
2^{14}	2^{14}	2^{14}	2^{-1}
2^{13}	2^{13}	2^{13}	0.395
2^{12}	2^{12}	2^{12}	0.373
2^{11}	2^{11}	2^{11}	0.368
2^{10}	2^{10}	2^{10}	0.366
2^i	2^i	2^i	≈ 0.366 for $i \le 9$
All other combinations: $\lvert C\rvert = 0$			

We next consider the components before the rotational shift, i.e., the OR-1 multiplication $F(x,y) = (x \vee 1) \otimes y$. For single-bit masks 2^i (input x), 2^j (input y), and 2^k (output), the correlation $C^F_{2^k,(2^i,2^j)}$ is nonzero only when $k = j$ and either $i = 0$ (the zero x-mask) or $j \le i \le 14$ (single-bit x-mask at a position $\ge j$). When the x-mask is zero, the magnitude is exactly $|C| = 2^{j-15}$; when it is a single bit at position i, the magnitude is

Table 3: Correlation of $F(x, y) = (x \vee 1) \otimes y$ for single-bit masks (including zero masks). For each output mask $c = 2^j$, the y-mask must equal 2^j and the x-mask must be either 0 or 2^i with $j \leq i \leq 14$. The maximum non-trivial correlation $|C| = 1$ is attained at $(a, b, c) = (0, 2^{15}, 2^{15})$.

Output mask c	Input masks $(a, b) = (x, y)$	$\lvert C_{c,(a,b)}\rvert$
0	$(0, 0)$	1
2^{15}	$(0, 2^{15})$	1
2^{14}	$(0, 2^{14})$, $(2^{14}, 2^{14})$	2^{-1}, 0.354
2^{13}	$(0, 2^{13})$, $(2^i, 2^{13})$ for $i \in \{13, 14\}$	2^{-2}, 0.231, 0.177
2^{12}	$(0, 2^{12})$, $(2^i, 2^{12})$ for $i \in \{12, 13, 14\}$	2^{-3}, 0.123, 0.115, 0.088
2^{11}	$(0, 2^{11})$, $(2^i, 2^{11})$ for $i \in \{11, \ldots, 14\}$	2^{-4}, 0.062, 0.061, 0.058, 0.044
2^{10}	$(0, 2^{10})$, $(2^i, 2^{10})$ for $i \in \{10, \ldots, 14\}$	2^{-5}, 0.031, 0.031, 0.031, 0.029, 0.022
2^9	$(0, 2^9)$, $(2^i, 2^9)$ for $i \in \{9, \ldots, 14\}$	2^{-6}, $\approx 2^{-6}$
2^8	$(0, 2^8)$, $(2^i, 2^8)$ for $i \in \{8, \ldots, 14\}$	2^{-7}, $\approx 2^{-7}$
2^7	$(0, 2^7)$, $(2^i, 2^7)$ for $i \in \{7, \ldots, 14\}$	2^{-8}, $\approx 2^{-8}$
2^6	$(0, 2^6)$, $(2^i, 2^6)$ for $i \in \{6, \ldots, 14\}$	2^{-9}, $\approx 2^{-9}$
2^5	$(0, 2^5)$, $(2^i, 2^5)$ for $i \in \{5, \ldots, 14\}$	2^{-10}, $\approx 2^{-10}$
2^4	$(0, 2^4)$, $(2^i, 2^4)$ for $i \in \{4, \ldots, 14\}$	2^{-11}, $\approx 2^{-11}$
2^3	$(0, 2^3)$, $(2^i, 2^3)$ for $i \in \{3, \ldots, 14\}$	2^{-12}, $\approx 2^{-12}$
2^2	$(0, 2^2)$, $(2^i, 2^2)$ for $i \in \{2, \ldots, 14\}$	2^{-13}, $\approx 2^{-13}$
2^1	$(0, 2^1)$, $(2^i, 2^1)$ for $i \in \{1, \ldots, 14\}$	2^{-14}, $\approx 2^{-14}$
2^0	$(0, 2^0)$, $(2^i, 2^0)$ for $i \in \{0, \ldots, 14\}$	2^{-15}, $\approx 2^{-15}$

All other combinations: $|C| = 0$

$|C| = 2^{j-15} \cdot |\cos(\pi \cdot 2^i/2^{16})|$, which is close to 2^{j-16} for $i \leq 14$ and equals 2^{j-15} in the $i = 0$ limit. The propagation pattern is shown in Table 3.

With the propagation pattern, we can construct a MILP model to count the least number of active single-bit update function.

Definition 2 (Active Single-bit Update Function (ASU))**.** An update function is *active* only when its output mask is non-zero.

When all input and output masks are zero (the trivial case), the correlation is exactly 1 for all operations considered (OR-1 multiplication, XOR, rotation, and addition), and the operation contributes no weight to the trail. Therefore, only active update functions (with non-zero output mask) contribute to the MILP objective. In the single-bit setting, an active update function additionally requires that its input masks follow the propagation patterns established above; otherwise its correlation is zero and no valid trail exists through it.

The MILP model for smallest number of ASU. We construct a Mixed Integer Linear Programming (MILP) model to search for the linear trail with the minimum number of active single-bit update functions (ASUs) over R rounds.

Variable setup. For each round $r \in \{0, 1, \ldots, R\}$, each state position $j \in \{0, 1, \ldots, 36\}$, and each bit position $b \in \{0, 1, \ldots, 15\}$, we introduce a binary variable

$$m_{r,j,b} \in \{0, 1\}, \quad m_{r,j,b} = 1 \iff \text{bit } b \text{ is the active bit of the mask on } S_j^{(r)}.$$

The single-bit constraint requires that each state word has at most one active bit:

$$\sum_{b=0}^{15} m_{r,j,b} \leq 1, \quad \forall r, j.$$

For each round $r \in \{0, 1, \ldots, R-1\}$, we additionally introduce an indicator variable $\mathsf{active}_r \in \{0, 1\}$ that equals 1 iff the update function output $\mathsf{next}^{(r)}$ is non-zero, i.e., the update function at round r is an ASU:

$$\mathsf{active}_r = \sum_{b=0}^{15} m_{r+1,36,b}.$$

Initial mask constraints. At round $r = 0$, the attacker may place active masks on any subset of the state. Two modes are considered:

- *Full state mode*: all 37 state words are allowed to carry an active mask, with at least one active bit overall.
- *Nonce-only mode*: active masks are restricted to the nonce-dependent positions $S_0, \ldots, S_7$, modeling the chosen-nonce attack scenario:

$$m_{0,j,b} = 0, \quad \forall j \notin \{0, 1, \ldots, 7\}, \forall b.$$

Model for the update function. At each round r, the AICE feedback function computes

$$\mathsf{next} = (S_0 + (S_{15} \vee 1) \otimes S_{25}) \lll \mathsf{RC}[r \bmod 37] \oplus S_{22} + S_{17},$$

followed by the shift $S'_j = S_{j+1}$ for $j \leq 35$ and $S'_{36} = \mathsf{next}$. Combining the propagation rules from Tables 3 and 2, we encode the following per-round constraints for $r = 0, 1, \ldots, R-1$:

- *State shift*: $m_{r+1,j,b} = m_{r,j+1,b}$ for $j = 0, 1, \ldots, 35$ and all b.
- *Final modular addition* ($\mathsf{next} = t_2 + S_{17}$): single-bit propagation requires $m_{r+1,36,b} = m_{r,17,b}$ for all b.
- *XOR composed with rotation* ($t_2 = (t_0 \lll k) \oplus S_{22}$, where $k = \mathsf{RC}[r \bmod 37]$): single-bit XOR requires the masks of $t_1 = t_0 \lll k$ and S_{22} to coincide. By the modular-mask propagation rule for rotation, an input mask at bit position i propagates to an output mask at bit position $(i - k) \bmod 16$; equivalently, the output mask at bit position b corresponds to the input mask at bit position $(b + k) \bmod 16$. Since t_0's mask is determined by the addition $t_0 = S_0 + (S_{15} \vee 1) \otimes S_{25}$, which forces t_0 to share the mask of S_0, we obtain

$$m_{r,22,b} = m_{r,0,(b+k) \bmod 16}, \quad \forall b.$$

- *OR-1 multiplication* ($(S_{15} \vee 1) \otimes S_{25}$ feeding into $t_0 = S_0 + \cdot$): the addition forces the multiplication output mask to match S_0's mask, denoted c. Then by Table 3,
  - the y-mask must match c: $m_{r,25,b} = m_{r,0,b}$ for all b;
  - the x-mask is restricted: bit 15 is forbidden ($m_{r,15,15} = 0$), and a non-zero x-mask requires the output to be non-zero with the x-mask bit position i no greater than the output bit position:

$$m_{r,15,i} \leq \sum_{j=0}^{i} m_{r,0,j}, \quad i = 0, 1, \ldots, 14.$$

Keystream constraints. The keystream extraction at round $r^\star$ is $Z_{r^\star} = (S_1 + S_5 + S_{19} + S_{32}) \oplus S_{30}$. To enforce that a single-bit linear approximation traces from the chosen-nonce initial state to a non-trivial keystream output mask, we introduce a binary indicator $z_b \in \{0, 1\}$ for each output bit b with

$$\sum_{b=0}^{15} z_b = 1.$$

By the propagation rules of modular addition and XOR (Tables 2 and 3), all five input positions of the keystream extraction must share the output mask at round $r^\star$:

$$m_{r^\star,j,b} = z_b, \quad \forall j \in \{1, 5, 19, 30, 32\}, \forall b.$$

Objective. The MILP minimizes the total number of ASUs over the R rounds:

$$\min \sum_{r=0}^{R-1} \mathsf{active}_r.$$

A lower bound on the linear trail correlation can be derived from the optimal objective value combined with the per-ASU correlation magnitudes summarized in Tables 3 and 2.

5.4 Cube and Division-Property Style Analysis

The division property [Tod15, TM16, XZBL16] generalizes integral reasoning and is a standard tool for studying zero-sum distinguishers and algebraic-degree growth in symmetric primitives. For AICE, such analysis must be applied with care: the cipher operates over $\mathbb{Z}/2^{16}\mathbb{Z}$ and combines modular addition, bitwise XOR, and word-level rotation, so coarse Boolean abstractions may fail to capture structural cancellations arising from the word-level arithmetic. We adopt a sampling-based methodology using empirical cube-sum verification over random key pairs [DS09, ADMS09].

Let $Y_0, Y_1, Y_2, \ldots$ denote the intermediate outputs produced after initialization at successive output steps, and let $Y_{-32}, \ldots, Y_{-1}$ denote the secret intermediate outputs generated during the output-memory initialization phase (Section 3.5). The keystream word at output step $i \geq 0$ is

$$Z_i \;=\; Y_i \;+\; Y_{i-32} \pmod{2^{16}},$$

so every keystream word mixes two intermediate outputs that are exactly one 37-step cycle apart, and no intermediate output is ever published on its own. The first keystream word is $Z_0 = Y_0 + Y_{-32}$, where Y_{-32} is a secret output-memory initialization value.

Each intermediate output Y_i takes the form

$$Y_i \;=\; \big((S_{i+1} + S_{i+5} + S_{i+19} + S_{i+32}) \bmod 2^{16}\big) \oplus S_{i+30}.$$

In this section, let R denotes the *total* number of initialization rounds, counting both the warm-up phase and the 37-step output-memory initialization cycle. The full cipher uses $R = 370$. Under this counting, Y_0 has undergone R rounds of register evolution relative to the nonce, and Y_{-32}—produced one full cycle earlier—has undergone $R - 37$ rounds.

For each cube C (a set of nonce-bit indices) and each total round count R, the cube sum is

$$\Sigma_C(K, N) \;=\; \bigoplus_{\mathbf{v} \in \{0,1\}^{|C|}} Z_0(K,\, N \oplus \mathbf{v} \cdot e_C)\,,$$

where e_C is the indicator vector for the cube variables and the XOR is taken over the full 16-bit word. A cube is *zero-sum at round* R if $\Sigma_C(K, N) = 0$ for all tested keys; the *zero-sum boundary* R^* is the last R at which any zero-sum cube exists. We test candidate nonce bits b_0 and b_{15} of each nonce word $N[0], \ldots, N[7]$, giving 16 candidates in total, and exhaustively test all $\sum_{d=1}^{8} \binom{16}{d} = 39{,}202$ cubes of dimension $d \in \{1, \ldots, 8\}$ at each round count. For each cube, 1,000 random key–nonce pairs are sampled with immediate termination upon the first non-zero sum.

5.4.1 Full-Word Zero-Sum Boundary

Testing zero-sum with respect to the full 16-bit word Z_0 requires all 16 bits of Σ_C to cancel simultaneously, the strictest possible condition.

At $R = 40$, zero-sum cubes exist at dimensions $d \geq 3$: dimension 3 yields 5 zero-sum cubes out of 560 candidates (for example, $\{N[2].b_{15},\ N[4].b_{15},\ N[2].b_0\}$), dimension 4 yields 33 out of 1,820, and dimension 5 yields at least 83 out of 4,368. No zero-sum cube exists at $d = 1$ or $d = 2$.

At $R = 41$, exhaustive testing over all 39,202 cubes of dimensions $d = 1, \ldots, 8$ finds no zero-sum cube at any dimension. The full-word zero-sum boundary is therefore

$$R^*_{\text{word}} \ = \ 40.$$

The zero-sum structure at $R = 40$ reflects the weak algebraic mixing present when both Y_0 and Y_{-32} have undergone few initialization rounds. At $R = 41$ the carry coupling in $Z_0 = Y_0 + Y_{-32}$ is sufficient to destroy all full-word zero-sum structure, as the nonlinear interaction between the two terms prevents global cancellation across all 16 bits. With $R = 370$ initialization rounds the cipher sits at a margin of $370/40 \approx 9\times$ above this boundary.

5.4.2 Per-Bit Zero-Sum Boundary and Its Structural Origin

For each bit $b \in \{0, \ldots, 15\}$ of Z_0, the per-bit cube sum is

$$\Sigma_C^{(b)}(K, N) \ = \ \bigoplus_{\mathbf{v} \in \{0,1\}^{|C|}} \big((Z_0(K, N \oplus \mathbf{v} \cdot e_C) \gg b) \bmod 2\big) \ \in \ \mathrm{GF}(2),$$

extracted from the same cipher evaluations at no additional cost.

The results exhibit a sharp asymmetry across bit positions. At $R = 80$, bit b_0 (the least-significant bit) retains zero-sum structure: dimension 3 yields 12 zero-sum cubes out of 560 candidates, dimension 4 yields 132 out of 1,820, and dimension 5 yields at least 655 out of 4,368. All other bit positions $b_1, \ldots, b_{15}$ show no zero-sum cube at any tested dimension at $R = 80$.

At $R = 81$, no zero-sum cube is found for any bit position at any dimension. The per-bit boundaries are

$$R^*_{b_0} \ = \ 80 \qquad \text{and} \qquad R^*_{b_i} \ < \ 80 \quad \text{for } i = 1, \ldots, 15.$$

The b_0 boundary of 80 admits a precise analytical derivation. Since no carry propagates *into* bit 0, the least-significant bit of a modular sum satisfies $\mathrm{lsb}(a + b) = \mathrm{lsb}(a) \oplus \mathrm{lsb}(b)$, giving

$$\Sigma_C^{(0)}\big[Z_0\big] \ = \ \Sigma_C^{(0)}\big[Y_0\big] \ \oplus \ \Sigma_C^{(0)}\big[Y_{-32}\big].$$

This sum is zero if and only if the b_0 cube sums of both Y_0 and Y_{-32} vanish simultaneously. Empirical experiments show that the per-bit b_0 zero-sum boundary of any individual intermediate output is $R^* = 80$, meaning b_0 of Y_i admits a zero-sum cube only up to round 80. Since Y_{-32} sees only $R - 37$ effective rounds, its zero-sum condition $R \leq 117$ is non-binding. The binding constraint comes from Y_0, which sees the full R rounds, giving $R^*_{b_0} = 80$ in exact agreement with the empirical result.

The earlier collapse of $b_1, \ldots, b_{15}$ is consistent with this framework: for $b > 0$, carry propagation from lower bit positions introduces nonlinear cross-bit coupling that raises the algebraic degree more rapidly, causing zero-sum structure to collapse at lower round counts.

Table 4 summarizes all boundaries. At the full 370-round initialization, no zero-sum cube was found for the full-word output or for any individual bit position of Z_0, across

Table 4: Empirical zero-sum boundaries for AICE under cube testing. R = total initialization rounds (warm-up + output-memory init phase); full cipher uses $R = 370$. Candidates: b_0 and b_{15} of $N[0], \ldots, N[7]$ (16 bits); dimensions $d = 1, \ldots, 8$; 1,000 keys sampled per cube. R^* is the last round at which any zero-sum cube exists.

Output tested	R^*	Example zero-sum cube at R^*	Margin vs. 370R
Full 16-bit word Z_0	40	$\{N[2].b_{15}, N[4].b_{15}, N[2].b_0\}$, $d = 3$	$\approx 9\times$
Per-bit b_0 of Z_0	80	$d = 3$, 12 zero-sum cubes	$\approx 5\times$
Per-bit b_i of Z_0, $i \geq 1$	80	No cube exits	$\approx 5\times$

all 39,202 tested cubes. The zero-sum windows are structural reduced-round phenomena with no extension to the operational cipher: the full cipher sits at margins of $370/40 \approx 9\times$ above the full-word boundary and $370/80 \approx 5\times$ above the per-bit b_0 boundary.

5.5 Summary of Security Margins and Scope

We now summarize the security evidence collected above and clarify the scope of the corresponding claims. Our analysis covers several complementary attack viewpoints, each under its own explicit model. The resulting picture is consistent across these viewpoints: the main reduced-round phenomena we observe occur far below the full 370-round initialization, while the full construction remains significantly beyond the experimentally identified thresholds in the adopted models.

For differential security, we considered the fixed-key, chosen-nonce setting and analyzed a weaker variant of AICE in which every XOR in the feedback is relaxed to a modular addition, so that the resulting trail weight lower-bounds that of AICE itself. Each 16-bit word is split into four 4-bit nibbles; the arithmetic operations are captured by an exact nibble-level add-and-multiply S-box DDT (with carry nibbles), and the rotation by a nibble-activity S-box DDT for every rotation amount, all encoded into a MILP. The initialization weight is built up by composition (Matsui's bound): 111 rounds of the update function have minimum differential weight at least $10 + 41 = 51$, hence 333 rounds at least 153; together with the 37-round nonce-only prefix of weight 18, the full 370-round initialization has minimum differential weight at least $18 + 153 = 171$. The best differential trail therefore has probability at most 2^{-171}, exceeding the 2^{-128} nonce-space threshold by a 43-bit margin.

For state recovery, we adopted a full-state guess-and-determine framework: the 37 post-feed-forward state words together with the 32 output-memory words (1104 hidden bits) are modeled as bit-vector unknowns in an SMT instance constrained by the public output equations. A heuristic search proposes guess sets, each validated by the exact SMT model. The best validated attack guesses 28 of the 37 state words: with the correct values the remaining 41 hidden words are recovered in about 90 seconds, whereas a wrong 28-word guess is rejected as UNSAT only after about 153 seconds on average; no working 27-word attack was found, even with more keystream and longer timeouts. The fast correct-guess solve is therefore only a conditional result; a complete GD attack must enumerate the guessed words, and with expensive wrong-branch rejection it stays far above the cost of exhaustive search.

For linear analysis, we worked in the additive-character framework over $\mathbb{Z}/2^{16}\mathbb{Z}$ and separated the initialization stage from the online keystream stage. For the initialization, a single-bit character MILP shows that the highest-correlation trails become infeasible beyond two rounds once the output constraint is imposed, and a block-level weighted MILP gives, in the chosen-nonce setting at $R = 200$ rounds, an optimal weighted trail value of 131. This translates to a correlation bound $|C| \leq 2^{-65.5}$ and a data requirement $D \geq 2^{131}$, which already exceeds the 2^{128} nonce space well below the full 370-round initialization.

Table 5: Summary of security evidence for AICE under the adopted attack models.

Aspect	Adopted model	Main observation	Interpretation
Init. differential	related-key, chosen-nonce; weaker variant (XOR→ADD), nibble DDT MILP, composed via Matsui's bound	Minimum trail weight ≥ 284 at 370 rounds ($\leq 2^{-284}$)	28-bit margin above the 2^{-256} exhaustive key-search threshold
State recovery	Full-state SMT GD (1104 hidden bits); heuristic guess-set search + exact validation	28-word guess validated (SAT $\approx$ 90 s); 27-word search times out; wrong branch $\approx$ 153 s UNSAT	Conditional recovery only; complete search far above exhaustive cost
Init. linear	Chosen-nonce; additive-character MILP (single-bit and block-level weighted)	Single-bit trails infeasible for $R{\geq}3$; weighted optimum 131 at $R{=}200$ ($\lvert C\rvert{\leq}2^{-65.5}$, $D{\geq}2^{131}$)	D exceeds the 2^{128} nonce space below the full round count
Online linear	Additive-character analysis of keystream combinations	Non-zero trail forced onto a fresh uniform word; output layer adds attenuation	Expected correlation 0; no online distinguisher
Cube / div. property	Propagation screening	Balanced rounds in small reduced-round window; no weak-key enlargement	Reduced-round phenomenon; not a full-round attack surface

For the online stage, any keystream combination that cancels the linear feedback taps necessarily forces a non-zero character onto a fresh, uniformly distributed internal word, whose character expectation is zero; the modular-addition output layer only adds further attenuation. Hence no online linear distinguisher arises.

For cube and division-property style analysis, balancedness claims are established by exact queries on the real keystream output, with propagation-based models used only for screening. Zero-sum cubes are confined to a small reduced-round region in all tested output configurations, with margins of approximately 5 to 9 times against the full 370-round cipher depending on whether the full output word or individual bits are probed. Conditional cube prerequisites are satisfiable on an empirically very sparse weak-key class, but no enlargement of this window is observed across any tested output bit or cube dimension.

Table 5 summarizes these conclusions together with the adopted attack models and their interpretation. Overall, the current evidence supports the view that AICE retains substantial security margins under the attack models studied here, while the strongest observable structural phenomena remain confined to reduced-round settings far below the full initialization schedule.

Finally, we emphasize several boundaries of the present analysis. We do not study related-key attacks. The differential results rely on a weaker variant of AICE in which every XOR is relaxed to a modular addition, together with a nibble-level abstraction, and the full-initialization weight is obtained by composition (Matsui's bound) rather than as a single proven optimum; the resulting weight is a lower bound on the true minimum trail weight. The linear bounds are chosen-nonce trail bounds obtained in the additive-character MILP model, in which each active update function is charged an upper bound on the correlation magnitude it can attain. The GD conclusions are relative to the adopted modeling framework. Within these boundaries, however, the results consistently indicate that the design choices of AICE, including long initialization, post-initialization key feed-forward, and periodic blank updates, provide meaningful security separation between the full construction and the reduced-round or weakened behaviors observed in our experiments.

Table 6: Single-core encryption throughput of AICE on the Ascend AI Cores (Pure-Kernel-Compute, Gbps).

Input size	1 KB	4 KB	16 KB	64 KB	256 KB	1 MB	4 MB	16 MB	64 MB
Ascend 910B4	0.22	0.87	3.45	12.20	29.44	45.82	53.14	55.36	53.14
Ascend 950	0.49	1.96	7.76	27.00	63.50	94.57	109.95	114.13	101.41

6 Performance Evaluation

6.1 Experimental Setup

We evaluate the performance of AICE on both AI-accelerator platforms and several general-purpose CPU platforms. Our goal is twofold: first, to study how an arithmetic-oriented stream cipher maps to a heterogeneous vector architecture; and second, to assess how well the same design remains competitive in conventional software environments.

For the accelerator-side evaluation, we implement AICE on two generations of Huawei Ascend AI Cores [Hua24b], based on the Da Vinci architecture: the Ascend 910B4 and the Ascend 950. The implementations use CANN and Ascend C [Hua24a], and we report both single-core and 32-core throughput across a wide range of input sizes to characterize how the encryption kernel scales with both message length and core count. All AI-Core numbers are measured under the Pure-Kernel-Compute configuration, which excludes host-side DMA and launch overhead.

For the CPU-side evaluation, we benchmark AICE against the standardized symmetric ciphers AES-CTR [DR02] and SM4-CTR on Huawei Kunpeng 920 (ARM, scalar build). Throughput is reported as encryption-only Gbps, so that the comparison reflects the cost of the stream-generation/encryption core rather than the cost of an AEAD construction.

6.2 Implementation on the AI Core

The Ascend implementation of AICE is designed to map the cipher's word-oriented arithmetic directly onto the vector execution model of the AI Core. In particular, the update and keystream-generation routines are organized around 16-bit arithmetic operations and regular state movement, which can be expressed efficiently through Ascend C vector intrinsics.

This implementation strategy is consistent with the design objective of AICE: rather than targeting a single processor family, it aims to expose a regular arithmetic workload that can be scheduled efficiently on heterogeneous vector hardware. On both tested AI Core generations, this leads to substantially higher throughput than the standardized software baselines we compare against.

Table 6 reports single-core AI Core throughput on both generations. Throughput grows rapidly with input size in the small-to-medium range – the kernel-launch and 370-round initialization overhead amortize over progressively larger payloads – and plateaus once the input is large enough to saturate the vector pipelines. On the 950, the kernel reaches a single-core peak of 114.13 Gbps at 16 MB, approximately a 2.06× generational improvement over the 910B4's 55.36 Gbps. The slight drop at 64 MB on both generations is consistent with on-chip cache scheduling pressure on large contiguous working sets. The $\sim 2\times$ single-core uplift on the 950 is attributable to two micro-architectural enhancements: a native 16-bit vector XOR (replacing the three-instruction `Or+And+Sub` emulation used on 910B4), and per-lane-shift-count support in the vector shift unit (which lets multiple rotation steps issue in a single vector instruction within the round function).

Table 7 reports the 32-core multi-core scaling on the Ascend 910B4 AI Core, alongside AES-CTR and SM4-CTR on the same 32-core configuration. AICE's throughput continues to scale strongly with both core count and input size, reaching 1 595.44 Gbps

Table 7: AICE multi-core scaling on the Ascend 910B4 AI Core, compared to AES-CTR and SM4-CTR on the same 32-core configuration (encryption throughput, Gbps).

Input size per core	1 KB	4 KB	16 KB	64 KB	256 KB	1 MB	4 MB	16 MB	64 MB
AICE (this work)	6.82	26.65	105.81	367.18	909.13	1441.99	1582.46	1595.44	1582.56
AES-CTR	5.14	16.42	25.98	28.38	29.15	29.26	29.33	29.42	29.47
SM4-CTR	5.74	6.25	14.37	14.80	14.81	14.83	14.84	14.84	14.84

Table 8: AICE encryption throughput on CPU platform (Gbps).

Input size	2 KB	4 KB	8 KB	16 KB	32 KB	64 KB	256 KB	1 MB	8 MB
ARM Kunpeng 920 (scalar)	1.99	3.53	5.77	8.44	10.97	12.87	14.77	14.86	12.22

at 16 MB – roughly 54× the 32-core AES-CTR (29.42 Gbps) and 107× the SM4-CTR peak (14.84 Gbps). The two standardized baselines saturate near 30 Gbps and 15 Gbps respectively once the AI Core's vector pipelines are filled, whereas the arithmetic-oriented structure of AICE continues to extract throughput from additional parallel cores. At sub-megabyte inputs, AICE remains competitive with the baselines, although the fixed cost of the 370-round initialization constitutes a larger fraction of the total runtime for short messages. Once the input crosses approximately 16 KB the keystream-generation phase becomes the dominant term and AICE's architectural fit takes over.

These results should be interpreted in terms of architectural alignment rather than as a claim about any single primitive in isolation. On both AI Core generations, the arithmetic-oriented structure of AICE maps well to the vectorized execution model available on the accelerator. This is precisely the setting that motivates AICE: a stream cipher whose high-throughput realization is not tied to a purely CPU-centric execution model.

6.3 Cross-Platform Software Performance on CPUs

To evaluate performance portability beyond the AI Core, we benchmark AICE on the Huawei Kunpeng 920 ARM CPU and report throughput across input sizes ranging from 2 KB to 8 MB.

We then compare AICE against AES-CTR and SM4-CTR on the same hardware, using AES-NI acceleration for AES-CTR and the OpenSSL implementation for SM4-CTR.

On the ARM Kunpeng 920 in scalar mode (Table 9), AICE achieves a peak throughput of 12.87 Gbps at 64 KB. While this is lower than the throughput of dedicated AES-CTR hardware acceleration (17.49 Gbps), it is more than 20× higher than that of the comparable software-only SM4-CTR implementation (0.58 Gbps).

The key takeaway is that AICE combines two properties that are rarely optimized together: very high throughput on a heterogeneous vector accelerator (single-core peak 114.13 Gbps on the Ascend 950 AI Core, multi-core 1 595.44 Gbps on 32 Ascend 910B4 AI Cores), and competitive software performance on the ARM CPU. This behavior is consistent with the goal of performance-portable symmetric encryption across CPU–accelerator environments.

7 Conclusion

We presented AICE, an arithmetic-oriented stream cipher motivated by the need for high-throughput symmetric encryption in heterogeneous computing environments. Rather than optimizing exclusively for a CPU-centric execution model, AICE explores a design point in which the core update and keystream-generation procedures are expressed through regular word-oriented arithmetic, with the goal of mapping efficiently to both conventional software platforms and heterogeneous vector hardware.

Table 9: ARM Kunpeng 920: AICE vs AES-CTR and SM4-CTR (Gbps).

Input size	2 KB	4 KB	8 KB	16 KB	64 KB
AICE (this work)	1.99	3.53	5.77	8.44	12.87
AES-CTR	13.47	14.30	16.40	17.03	17.49
SM4-CTR	0.58	0.58	0.58	0.58	0.58

From the design perspective, AICE combines a 37-word state over $\mathbb{Z}/2^{16}\mathbb{Z}$, an arithmetic feedback function, a long initialization phase, post-initialization key feed-forward, and periodic blank updates. The rationale for these choices is to balance implementation regularity with resistance to the main attack directions considered in this paper. Our security analysis does not claim unconditional security; rather, it provides evidence under several explicit models. In the fixed-key, chosen-nonce setting, the initialization differential screening results indicate that high-probability nonce-only trails are already strongly attenuated well before the full 370-round initialization. In the adopted guess-and-determine framework, the full construction is strictly harder than the weakened variants once key feed-forward is restored. From the linear viewpoint, we obtain a strong upper bound on IV-induced correlation magnitude under fixed-key, varying-nonce analysis, and we do not observe a short exact GF(2) post-initialization skeleton relation of MORUS type in the tested range. Exact SMT-based cube analysis further suggests that the observed balancedness phenomena are confined to a small reduced-round regime and do not translate into an enlarged attack surface at full rounds.

On the implementation side, the experimental results support the intended positioning of AICE. On the tested AI Core platform, AICE achieves substantially higher throughput than the compared software-oriented baselines and scales favorably under multi-core execution. At the same time, it remains competitive on the ARM CPU platform, even though it is not intended to outperform the best architecture-specific primitive in every software environment. Taken together, these results suggest that AICE is a promising step toward performance-portable symmetric encryption across CPU–accelerator settings.

Several directions remain open. On the cryptanalytic side, a more general treatment of post-initialization character-based multi-output relations over $\mathbb{Z}/2^{16}\mathbb{Z}$ appears substantially more involved and is left for future work. On the implementation side, it would also be interesting to evaluate AICE on a broader range of heterogeneous vector architectures and to study integration with authenticated-encryption constructions tailored to accelerator-oriented data paths.

References

[AD13] Tomer Ashur and Orr Dunkelman. A practical related-key boomerang attack for the full MMB block cipher. In Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab, editors, *CANS 13*, volume 8257 of *LNCS*, pages 271–290. Springer, Cham, November 2013.

[ADMS09] Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube testers and key recovery attacks on reduced-round MD6 and Trivium. In Orr Dunkelman, editor, *FSE 2009*, volume 5665 of *LNCS*, pages 1–22. Springer, Berlin, Heidelberg, February 2009.

[BCJW02] Nikita Borisov, Monica Chew, Robert Johnson, and David Wagner. Multiplicative differentials. In Joan Daemen and Vincent Rijmen, editors, *FSE 2002*, volume 2365 of *LNCS*, pages 17–33. Springer, Berlin, Heidelberg, February 2002.

[Ber08] Daniel J. Bernstein. ChaCha, a variant of Salsa20. Workshop Record of SASC 2008: The State of the Art of Stream Ciphers, 2008. https://cr.yp.to/chacha/chacha-20080120.pdf.

[Bey21] Tim Beyne. Linear cryptanalysis of FF3-1 and FEA. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I*, Lecture Notes in Computer Science, pages 41–69. Springer, 2021.

[BSV07] Thomas Baignères, Jacques Stern, and Serge Vaudenay. Linear cryptanalysis of non binary ciphers. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers*, Lecture Notes in Computer Science, pages 184–211. Springer, 2007.

[CP08] Christophe De Cannière and Bart Preneel. Trivium. In Matthew Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 244–266. Springer, 2008.

[Dae95] Joan Daemen. *Cipher and Hash Function Design Strategies Based on Linear and Differential Cryptanalysis.* PhD thesis, Katholieke Universiteit Leuven, 1995.

[De 06] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *ISC 2006*, volume 4176 of *LNCS*, pages 171–186. Springer, Berlin, Heidelberg, August / September 2006.

[DGV93] Joan Daemen, René Govaerts, and Joos Vandewalle. Block ciphers based on modular arithmetic. In W. Wolfowicz, editor, *Proc. 3rd Symposium on State and Progress of Research in Cryptography*, pages 80–89, Rome, Italy, 1993. Fondazione Ugo Bordoni.

[DLGV92] Joan Daemen, Luc Van Linden, René Govaerts, and Joos Vandewalle. Propagation properties of multiplication modulo $2^n - 1$. In *Proc. 13th Symposium on Information Theory in the Benelux*, pages 111–118, Enschede, The Netherlands, 1992. Werkgemeenschap voor Informatie- en Communicatietheorie.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Information Security and Cryptography. Springer, 2002.

[DS09] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 278–299. Springer, Berlin, Heidelberg, April 2009.

[EJMY19] Patrik Ekdahl, Thomas Johansson, Alexander Maximov, and Jing Yang. A new SNOW stream cipher called SNOW-V. *IACR Transactions on Symmetric Cryptology*, 2019(3):1–42, 2019.

[ETS11] ETSI/SAGE. Specification of the 3GPP confidentiality and integrity algorithms 128-EEA3 & 128-EIA3. document 2: ZUC specification. ETSI/SAGE Specification, Version 1.6, 2011.

[Fur02] Vladimir Furman. Differential cryptanalysis of Nimbus. In Mitsuru Matsui, editor, *FSE 2001*, volume 2355 of *LNCS*, pages 187–195. Springer, Berlin, Heidelberg, April 2002.

[Gue10] Shay Gueron. Intel advanced encryption standard (AES) new instructions set. Intel White Paper, Document 323641-001, Revision 3.0, 2010.

[HJM07] Martin Hell, Thomas Johansson, and Willi Meier. Grain: A stream cipher for constrained environments. *International Journal of Wireless and Mobile Computing*, 2(1):86–93, 2007.

[HT16] Wu Hongjun and Huang Tao. The authenticated cipher morus (v2), 2016.

[Hua24a] Huawei Technologies Co., Ltd. Ascend C programming guide. https://www.hiascend.com/document/detail/en/canncommercial/80RC1/devaids/auxiliarydevtool/atlasascendc_16_0004.html, 2024. CANN 8.0, Accessed: 2026-03-01.

[Hua24b] Huawei Technologies Co., Ltd. Huawei Ascend 910b ai processor. https://e.huawei.com/en/products/computing/ascend, 2024. Accessed: 2026-03-01.

[JCWW12] Keting Jia, Jiazhe Chen, Meiqin Wang, and Xiaoyun Wang. Practical attack on the full MMB block cipher. In Ali Miri and Serge Vaudenay, editors, *SAC 2011*, volume 7118 of *LNCS*, pages 185–199. Springer, Berlin, Heidelberg, August 2012.

[LM91] Xuejia Lai and James L. Massey. A proposal for a new block encryption standard. In Ivan Damgård, editor, *EUROCRYPT'90*, volume 473 of *LNCS*, pages 389–404. Springer, Berlin, Heidelberg, May 1991.

[LM02] Helger Lipmaa and Shiho Moriai. Efficient algorithms for computing differential properties of addition. In Mitsuru Matsui, editor, *FSE 2001*, volume 2355 of *LNCS*, pages 336–350. Springer, Berlin, Heidelberg, April 2002.

[LMM91] Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 17–38. Springer, Berlin, Heidelberg, April 1991.

[NSYW26] Zhongfeng Niu, Siwei Sun, Hailun Yan, and Qi Wang. Link between the differential cryptanalysis and linear approximations over finite abelian groups and its applications. *J. Cryptol.*, 39(2):13, 2026.

[SHW+14] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 158–178. Springer, Berlin, Heidelberg, December 2014.

[SLN+21] Kosei Sakamoto, Fukang Liu, Yuto Nakano, Shinsaku Kiyomoto, and Takanori Isobe. Rocca: an efficient aes-based encryption scheme for beyond 5g. *IACR Transactions on Symmetric Cryptology*, pages 1–30, 2021.

[SLN+22] Kosei Sakamoto, Fukang Liu, Yuto Nakano, Shinsaku Kiyomoto, and Takanori Isobe. Rocca: An efficient AES-based encryption scheme for beyond 5G (full version). Cryptology ePrint Archive, Report 2022/116, 2022.

[TM16] Yosuke Todo and Masakatu Morii. Bit-based division property and application to simon family. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 357–377. Springer, Berlin, Heidelberg, March 2016.

[Tod15] Yosuke Todo. Structural evaluation by generalized integral property. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 287–314. Springer, 2015.

[WNS09] Meiqin Wang, Jorge Nakahara, Jr., and Yue Sun. Cryptanalysis of the full MMB block cipher. In Michael J. Jacobson, Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *SAC 2009*, volume 5867 of *LNCS*, pages 231–248. Springer, Berlin, Heidelberg, August 2009.

[WP14] Hongjun Wu and Bart Preneel. AEGIS: A fast authenticated encryption algorithm. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 185–201. Springer, Berlin, Heidelberg, August 2014.

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 648–678. Springer, Berlin, Heidelberg, December 2016.

[Yil20] Hamdi Murat Yildirim. Exploiting linearity of modular multiplication. Springer, Lecture Notes in Computer Science, 2020.

A Truncated Differential Propagation in the Initialization

The nibble-level bound of Section 5 treats every active word as fully unknown. We complement it here with a *truncated* differential analysis that follows only the most significant bit (MSB) of the feedback operands. This coarse view yields a lower bound on the number of initialization rounds over which a difference can be kept away from the rotation input, the position where carries make a difference most expensive.

The feedback word rotates the modular sum

$$t \;=\; S_0 + (S_{15} \vee 1) \cdot S_{25}$$

before the round-dependent rotation $\lll \mathsf{RC}[r \bmod 37]$. A difference that reaches the rotation input is the worst case for a trail: in both the modular addition and the modular multiplication, a difference in a low-order bit propagates along the carry chain to higher bits, so its differential probability decays exponentially with the number of rounds. A trail that avoids this cost must keep the difference of t at zero, which constrains the differences of the triple (S_0, S_{15}, S_{25}).

Most-significant-bit transitions. Restricting the difference to the MSB removes the carry interaction. In modular addition and multiplication the carry travels only from low to high bits, so an MSB difference produces no carry difference and is independent of the lower bits, and the term $S_{15} \vee 1$ touches only the least significant bit. As long as the lower bits carry no difference, an MSB difference therefore passes through t without any carry-induced probability loss. Enumerating the MSB differences of (S_0, S_{15}, S_{25}) that keep the MSB difference of t at zero gives the transitions of Table A1.

Table A1: MSB difference patterns of (S_0, S_{15}, S_{25}) and the probability that the MSB difference of $t = S_0 + (S_{15} \vee 1) \cdot S_{25}$ vanishes.

(S_0, S_{15}, S_{25})	Valid	Probability
$(0,0,0)$	✓	1.0
$(0,0,1)$	✗	—
$(0,1,0)$	✓	0.5
$(0,1,1)$	✓	0.5
$(1,0,0)$	✗	—
$(1,0,1)$	✓	1.0
$(1,1,0)$	✓	0.5
$(1,1,1)$	✓	0.5

Reach of an MSB difference. In the 1-bit (MSB-only) model the modular addition behaves exactly as XOR, and the XOR feedback term follows the obvious rule. We encode this MSB propagation as a MILP instance and maximize the number of rounds over which the rotation input stays difference-free. The model admits 8 trails that span 66 update rounds, and no trail reaches 67. Extending the model to the top 2, 3, and 4 bits returns the same 66-round limit; a finer model (5 bits or more) might push it further, but the search becomes substantially more expensive.

A difference can thus be kept off the rotation input for at most 66 of the 370 initialization rounds. Over the remaining rounds every trail pays the carry-induced cost, consistent with the much larger weight bound obtained from the full nibble model in Section 5.