A Hardware-Efficient AEAD Stream Cipher Based on a Hybrid Nonlinear Feedback Register Structure

Arman Allahverdi
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, United States of America
aallahverdi3@gatech.edu

Vincent John Mooney III
School of Electrical and Computer Engineering
School of Computer Science
Georgia Institute of Technology
Atlanta, United States of America
mooney@gatech.edu

***Abstract*—In this paper, we propose the hybrid register stream cipher, a hardware-oriented AEAD-capable stream cipher based on nonlinear feedback shift registers (NLFSRs) and Composite Mersenne Product Registers (CMPRs) designed to balance security and hardware efficiency. Our proposed stream cipher integrates a 128-bit NLFSR with a 128-bit CMPR, achieving highly nonlinear internal state evolution while enabling scalable and lightweight hardware implementations. The hybrid register structure supports a 128-bit key, 96-bit initialization vector, and variable-length messages with associated data, offering 128-bit security with a 64-bit authentication tag. Statistical testing via the NIST Statistical Test Suite and bit contribution tests confirms the pseudorandomness of the output of our design. ASIC hardware implementation results demonstrate that the hybrid register stream cipher outperforms prominent lightweight stream ciphers such as TRIVIUM, Espresso, and Grain-128AEADv2 both in terms of area and energy consumption, with the hybrid register stream cipher achieving up to 25.8% lower hardware area and 67.5% lower energy consumption than the comparison candidates. Finally, we demonstrate that our hybrid register construction can be easily scaled to support different key and IV sizes.**

***Index Terms*—encryption, AEAD, stream cipher, feedback register, authentication**

I. Introduction

The growing demand for cryptographic solutions tailored to hardware-constrained environments has led to a proliferation of lightweight stream ciphers optimized for area, power, and performance. While many existing designs successfully minimize resource utilization, they often do so at the expense of either cryptographic flexibility, such as support for authenticated encryption, or support for different key sizes.

In this work, we present a new scalable hardware-efficient stream cipher capable of authenticated encryption with associated data (AEAD), termed the *hybrid register stream cipher*, that leverages a hybrid feedback register structure combining the nonlinear feedback shift register used in the Grain-128AEADv2 stream cipher [4] with a Composite Mersenne Product Register (CMPR) [1]. Our design balances hardware implementation simplicity, security, and support for AEAD, making the hybrid register stream cipher suitable for embedded and IoT applications. The hybrid register stream cipher proposed in this paper integrates lightweight keystream generation and authentication mechanisms with highly nonlinear evolution of its internal state, thereby offering resistance to classical and structural cryptanalytic attacks. We show that the hybrid register stream cipher outperforms several well-known lightweight stream ciphers in terms of application-specific integrated circuit (ASIC) area and power while delivering comparable (or superior) security. We also demonstrate that our CMPR design principles allow for flexible and scalable stream cipher design by varying the size of the CMPR, allowing for the design of stream ciphers with different key and initialization vector (IV) sizes (and, in turn, different security margins).

II. Background and Prior Work

A. Feedback Registers and Product Registers

Feedback registers are finite-state systems whose state evolves at discrete time steps according to a fixed update function. That is, the state of the register at time t, $A[t]$, is updated according to some fixed function $f : \{0,1\}^n \to \{0,1\}^n$:

$$A[t+1] = f(A[t]) \tag{1}$$

In practice, feedback registers are often implemented as linear feedback shift registers (LFSRs), where f is a linear Boolean function, or nonlinear feedback shift registers (NLFSRs), where f is a nonlinear Boolean function.

One way to implement a feedback register is as a Product Register [1]. An $n-$bit Product Register is parametrized by its update polynomial $U(x)$ (abbreviated as U), where the degree of U is less than or equal to $n-1$, and its feedback polynomial $P(x)$ (abbreviated as P), where the degree of P is equal to n. U and P are elements of $\mathbb{F}_{2^n}$, meaning the only possible coefficients for each term in these polynomials are 0 and 1. Similar to an LFSR, it is possible to ensure that an $n-$bit Product Register has full period, or cycles through 2^n-1 $n-$bit states before returning to its initial state [1]. For a Product Register to exhibit full period, the polynomial P must be a primitive polynomial. The state $A[t]$ of a Product Register updates according to the equation:

$$A[t+1] = (U \times A[t]) \bmod P. \tag{2}$$

From Equation 2, it is clear that the special case where $U = 0$ instantiates a Product Register that only updates to the all-zero state, and the special case of $U = 1$ instantiates a Product

Register that never changes state. Therefore, $U = 0$ and $U = 1$ are generally disallowed when instantiating a Product Register.

B. Mersenne Product Registers

A Mersenne Product Register (MPR) is a special case of a Product Register where the size of the register is a Mersenne exponent, i.e., a prime number n such that the quantity $2^n - 1$ is also a prime number. In the case of an MPR, the polynomial P is of degree n (n a Mersenne exponent), meaning that irreducibility, or the infeasibility of factoring P into the product of two non-constant polynomials, is a sufficient condition for primitivity [1]. For an $n-$bit MPR initialized to a nonzero $n-$bit state, an irreducible P and any $U \neq 0, 1$ ensure that the MPR will cycle through $2^n - 1$ states before returning to its initial state [1].

C. Chaining and Composite Mersenne Product Registers

Chaining is a method for composing several Mersenne Product Registers into a larger interconnected structure referred to as a Composite Mersenne Product Register (CMPR), where the MPRs are interconnected using chaining functions. Chaining functions are sets of Boolean equations that determine how the state of one MPR affects the state of another MPR. Let M_1 and M_2 be two distinct MPRs. Furthermore, let U_2 and P_2 denote the update polynomial and primitive polynomial of M_2, respectively. To chain from M_1 to M_2, define a set of chaining functions $\mathcal{C}$ in terms of the variables from M_1, where the number of chaining functions in $\mathcal{C}$ is equal to the size of M_2. Then, the state of M_2 will update according to the following:

$$M_2[t+1] = [(U_2 \times M_2[t]) \oplus \mathcal{C}] \bmod P_2. \tag{3}$$

$\mathcal{C}$ can be nonlinear, making CMPRs an appealing building block for cryptographic schemes [1]. Generally, using more MPRs to construct a CMPR results in a CMPR construction that is capable of generating sequences with higher linear complexity. Rules to abide by when constructing a CMPR include (i) using unique Mersenne exponents, (ii) only chaining from larger MPRs to smaller MPRs, and (iii) ensuring that chaining functions connect from one MPR to the next smallest MPR in the construction (that is, in a CMPR construction with three or more MPRs, chaining functions should not "skip" over any MPRs). Rules (i) and (ii) ensure that the CMPR has exponential period, with an $n-$bit CMPR having a period of at least $0.45 * 2^n$, and rule (iii) ensures that sequences generated by the CMPR can resist cryptanalytic techniques such as cube attacks and cube testers [1]. Note that the largest MPR in a CMPR does not receive any chaining input and therefore operates according to Equation 2, not Equation 3; in other words, the largest MPR in a CMPR exhibits a linear state sequence.

D. TRIVIUM

TRIVIUM is a synchronous stream cipher designed to balance speed and hardware area [2]. The internal state of TRIVIUM comprises 288 bits, with the internal state distributed across three nonlinear feedback shift registers (NLFSRs) of lengths 93, 84, and 111 bits. TRIVIUM supports an 80-bit secret key and an 80-bit initialization vector (IV). The initialization process of TRIVIUM consists of clocking the 288-bit internal state 1152 times to ensure sufficient state randomization before key stream generation begins.

E. Espresso

Espresso is a synchronous stream cipher geared toward 5G wireless communication systems, offering a trade-off between hardware area and security [3]. The cipher employs a 256-bit internal state implemented as an NLFSR. The keystream output of Espresso is derived through a balanced nonlinear Boolean function of 20 variables from the 256-bit NLFSR. The key and IV sizes for Espresso are 128 and 96 bits, respectively, with an initialization phase consisting of clocking the NLFSR 256 times to ensure the internal state is sufficiently randomized before keystream generation.

F. Grain-128AEADv2

Grain-128AEADv2 is an AEAD-compliant lightweight encryption algorithm designed for use in resource-constrained environments such as IoT and embedded systems [4]. Grain-128AEADv2 employs a 256-bit internal state formed by a 128-bit LFSR and a 128-bit NLFSR. The scheme uses a 128-bit key and a 96-bit IV, providing 128 bits of security, and integrates encryption and authentication via a pre-output function that generates keystream bits and authentication bits on alternating clock cycles. The cipher features an initialization procedure encompassing a 512-clock-cycle state-mixing phase [4].

III. Design Specification

In this section, we propose and specify the hybrid register stream cipher, a synchronous stream cipher that supports both encryption and AEAD applications. The hybrid register stream cipher is designed to support a $128-$bit key, $96-$bit IV, and messages of variable length, where the message may have associated data embedded within it. The output of the hybrid register stream cipher is a ciphertext with length equal to the plaintext length and a $64-$bit authentication tag. The hybrid register stream cipher utilizes a $256-$bit internal state, with the internal state consisting of a 128-bit NLFSR connected to a 128-bit CMPR. Our choices of internal state size, key length, and IV length were motivated by the designs we compare to in this paper, in order to ensure we can provide equal (or better) security while performing better in hardware.

Section III-A provides a detailed specification of the NLFSR and CMPR used in the hybrid register stream cipher, along with an overview of the authentication generator. Section III-B describes the role played by the key and IV in the initialization of the cipher. Section III-C discusses how the authentication structure is initialized. Section III-D examines how the hybrid register stream cipher produces keystream bits to be used for encryption and authentication bits to be used for authentication tag generation. Section III-E describes how our design supports AEAD.

A. Design Overview and Building Blocks

The hybrid register consists of two main building blocks: (i) a keystream generator, which generates pseudorandom bits to be used for encryption or authentication, and (ii) an authentication generator, which generates tags for authenticating messages. While the two main building blocks are interconnected, they utilize different hardware structures.

The keystream generator consists of a 128-bit NLFSR connected to a 128-bit CMPR, meaning the hybrid register stream cipher uses a 256-bit internal state. We use the same 128-bit NLFSR utilized in Grain-128AEADv2 [4], which has a 4th-degree nonlinear feedback polynomial $g(x)$ over $\mathbb{F}_2$ defined as follows:

$$\begin{aligned} g(x) = 1 &+ x^{32} + x^{37} + x^{72} + x^{102} + x^{128} + x^{44}x^{60} + x^{61}x^{125} \\ &+ x^{63}x^{67} + x^{69}x^{101} + x^{80}x^{88} + x^{110}x^{111} + x^{115}x^{117} \\ &+ x^{46}x^{50}x^{58} + x^{103}x^{104}x^{106} + x^{33}x^{35}x^{36}x^{40} \end{aligned} \quad (4)$$

The nonlinear feedback polynomial $g(x)$ is applied to the most significant bit of the NLFSR, meaning the remaining bits of the NLFSR shift to the right when the state of the NLFSR is updated. The 128-bit CMPR is specified in Table I, including the irreducible feedback polynomials and update polynomials used for each MPR.

TABLE I
SPECIFICATION FOR THE 128-BIT CMPR

MPR Size	Update Polynomial	Primitive Polynomial
61 bits	$U_{61}(x) = x^3 + 1$	$P_{61}(x) = x^{61} + x^{44} + x^{19} + x^{15} + 1$
31 bits	$U_{31}(x) = x^2 + 1$	$P_{31}(x) = x^{31} + x^3 + x^2 + x + 1$
19 bits	$U_{19}(x) = x^4 + x^2$	$P_{19}(x) = x^{19} + x^5 + x^2 + x + 1$
7 bits	$U_7(x) = x^4 + x$	$P_7(x) = x^7 + x + 1$
5 bits	$U_5(x) = x^3$	$P_5(x) = x^5 + x^2 + 1$
3 bits	$U_3(x) = x^2$	$P_3(x) = x^3 + x + 1$
2 bits	$U_2(x) = x + 1$	$P_2(x) = x^2 + x + 1$

Let the 256-bit state of the keystream generator be denoted by $s_{255}, \ldots, s_0$. The output of the keystream generator is derived by taking the XOR of state bits 0, 3, and 7. In other words, the keystream generator produces one bit per clock cycle according to the equation $s_7 \oplus s_3 \oplus s_0$.

The 128-bit NLFSR and 128-bit CMPR are connected using nonlinear chaining functions in terms of the state variables from the NLFSR. Specifically, the chaining functions connect from the 128-bit NLFSR to the largest MPR of the 128-bit CMPR, which is a 61-bit MPR (as seen in Table I and Fig. 1). The chaining functions are designed to satisfy the following criteria: (i) each chaining function uses AND and XOR gates of at most 4 inputs, meaning the chaining is of degree 4; (ii) each chaining function must be balanced, meaning that its truth table should have an equal number of zeros and ones; and (iii) the *chaining density* must be 40%, meaning that approximately 40% of the bits of the 61-bit MPR receive a chaining function from the NLFSR (the choice of chaining density is empirical and will be justified in Section IV-A1). Requirements (i)-(iii) apply to all other chaining functions in the design, or the chaining functions between the remaining MPRs. The keystream generator is illustrated in Fig. 1, with the chaining functions, each of which consists of a set of Boolean functions satisfying (i)-(iii) denoted by $C_1, \ldots, C_7$, and the update and primitive polynomial of each $n-$bit MPR denoted by U_n, P_n.

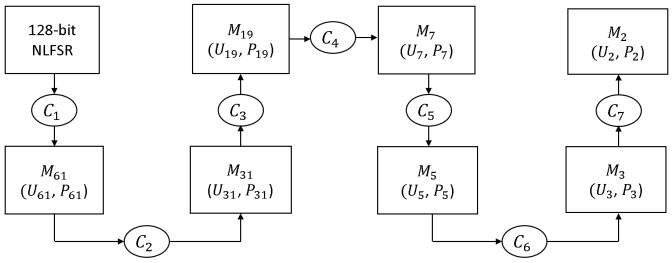

Fig. 1. Hybrid Register Keystream Generator Structure

The authentication generator consists of a 64-bit shift register and a 64-bit accumulator, producing a 64-bit authentication tag. The shift register, which receives pseudorandom bits from the keystream generator, feeds its current state to the accumulator, where the accumulator also receives the message bits to be authenticated. Our design integrates the authentication generator used in Grain-128AEADv2. The authentication generator is illustrated in Fig. 2.

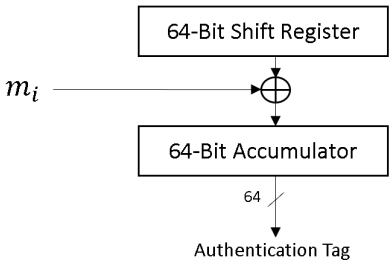

Fig. 2. Hybrid Register Authentication Generator Structure

B. Key and IV Initialization

Before the output of the keystream generator can be used for encryption and authentication, the keystream generator must undergo an initialization phase involving the 128-bit key k and 96-bit IV IV. Let the key bits be denoted by k_j, where $0 \leq j \leq 127$, and the IV bits be denoted by IV_j, where $0 \leq j \leq 95$. Then, we initialize the keystream generator as follows, again denoting the state of the keystream generator by $s_{255}, \ldots, s_0$:

- Initialize the 128 bits of the NLFSR with the 128 bits of the key, such that $s_j = k_j$ for $128 \leq j \leq 255$.
- Initialize the 96 most significant bits of the CMPR with the 96 bits of the IV, such that $s_j = IV_j$ for $32 \leq j \leq 127$.

- Initialize the remaining 32 bits of the CMPR with ones, such that $s_j = 1$ for $0 \leq j \leq 31$.

Subsequently, the 256-bit hybrid register structure is clocked a total of 128 times. Afterwards, we consider key and IV initialization to be complete and proceed to initialize the authentication generator.

C. Authenticator Initialization

Let the state of the shift register at clock cycle t be denoted by $r_{63}^t, \ldots, r_0^t$ and the state of the accumulator be denoted by $a_{63}^t, \ldots, a_0^t$. Moreover, let the output bit of the keystream generator at clock cycle t be denoted by z^t. For example, since the keystream generator was previously clocked 128 times as part of the keystream generator initialization process, the next keystream generator bit that will be produced is z^{128}. Once the keystream generator has been initialized, we clock the keystream generator an additional 128 times to initialize the authentication generator with keystream generator bits as follows:

- $a_j^{128} = z^{128+j}$ for $0 \leq j \leq 63$.
- $r_j^{128} = z^{192+j}$ for $0 \leq j \leq 63$

Once the accumulator and shift register have been initialized with keystream bits, the cipher is ready to produce ciphertexts and authentication tags.

D. Keystream and Tag Generation

For our design to support encryption and authentication, we assign a dual purpose to the keystream generator. The keystream generator is tasked with producing bits to use for encryption and bits to use for authentication (tag generation). We do not concurrently use the same keystream bits for both encryption and authentication, however.

Since the keystream generator has thus far undergone 256 clock cycles due to the initialization processes from the prior sections, the bit that will be generated at clock cycle i is denoted by z^{256+i}. Let y_i denote the i-th keystream bit. Every even keystream generator bit (counting from y_0) is used as an encryption bit. Then, y_i is given by:

$$y_i = z^{256+2i} \tag{5}$$

Similarly, let y_i' denote the i-th authentication bit. We use every odd keystream generator bit (counting from y_0') for authentication. Then, y_i' is given by:

$$y_i' = z^{256+2i+1} \tag{6}$$

Let m denote a message of length L bits. Our design allows for authenticating the message one bit at a time; thus, we denote the message bit currently being authenticated as m_i, where $0 \leq i \leq L-1$. Then, at the i-th clock cycle (after initialization), the shift register in Fig. 2 is updated as follows:

$$\begin{aligned} r_{63}^{i+1} &= y_i' \\ r_j^{i+1} &= r_{j+1}^i, 0 \leq j \leq 62 \end{aligned} \tag{7}$$

Similarly, the accumulator shown in Fig. 2 is updated as follows:

$$a_j^{i+1} = a_j^i \oplus m_i r_j^i \tag{8}$$

Once all message bits have been authenticated, the 64-bit state of the accumulator is taken as the authentication tag.

E. Authenticated Encryption with Associated Data

To support AEAD, we adopt the approach used in Grain-128AEADv2 [4]. That is, we define an AEAD mask d, which is of the same length as the plaintext message m. For each bit d_i in d, we set $d_i = 1$ if m_i should be encrypted. Otherwise, if m_i is an associated data bit and should not be encrypted, we set $d_i = 0$. Then, m_i is encrypted to produce the ciphertext bit c_i according to the following:

$$c_i = m_i \oplus (z_i d_i) \tag{9}$$

IV. Design Rationale

In this section, we discuss and justify the design choices presented in Section III. In Section IV-A, we rationalize our choice of register construction and describe why we believe our chosen construction is conducive to desirable cryptographic properties. In Section IV-B, we discuss our choice of 128 initialization rounds for the keystream generator. In Section IV-C, we elaborate on the factors that motivated our choices for where to initially place the key and IV in the hybrid register construction.

A. Hybrid Register Construction

Our 256-bit hybrid register construction consists of a 128-bit NLFSR connected to a 128-bit CMPR, where the 128-bit CMPR is constructed using the 7 MPRs in Table I. Thus, in total, the 256-bit hybrid register consists of 8 subregisters; we define a subregister to be a self-contained feedback register. In other words, a subregister can be dissected from the overall hybrid register with a known feedback function and therefore well-defined properties as a standalone unit. The subregisters in our 256-bit hybrid register are connected in order of descending size as shown in Fig. 1. This configuration follows the results of [1], which requires that the subregisters in a CMPR construction be chained together in order of descending subregister size to maximize the period of the construction, and allows for the largest subregister in a CMPR construction to be any other form of register (e.g., an LFSR or NLFSR) provided the size of the largest subregister is still coprime with the sizes of the other subregisters in the construction (see Theorem 1, also called the Chaining Period Theorem, in [1]). In our case, the NLFSR size is not coprime with the 2-bit MPR, meaning we do not consider the 2-bit MPR as part of our linear complexity and periodicity analysis.

By using a 128-bit NLFSR as the largest subregister in our construction, we ensure that the entire state of the 256-bit hybrid register evolves in a nonlinear fashion. The nonlinear state sequence of the 128-bit NLFSR also affects the state sequences of the 7 MPRs, as the 128-bit NLFSR is connected to the 128-bit CMPR using nonlinear chaining functions with AND gates of at most 4 inputs.

1) Chaining Function Selection: Our construction uses chaining functions with XOR gates and AND gates with at most 4 inputs. The chaining functions are chosen to be balanced, meaning their truth tables contain an equal number of zeros and ones, in order to ensure that the state of the hybrid register is not biased with an excessive number of zeros or ones. A biased internal state is conducive to statistical vulnerabilities in the ciphertext, which could be exploited by an attacker.

We employ a chaining function density of 40%, meaning that all MPRs in the 128-bit CMPR receive chaining functions to approximately 40% of their bits (a nearly 40% chaining density is not possible in the case of the 2- or 3-bit MPRs; in these instances, the MPRs receive as many chaining functions as needed to exceed 40%). We can use a lower chaining density due to the fact that the largest subregister in the hybrid register construction, the 128-bit NLFSR, has a nonlinear state sequence, meaning we need not rely solely on chaining functions as the source of nonlinearity in our design. The exact percentage for chaining density was chosen empirically, based on applying statistical tests (Section V-C) to the output of the keystream generator to hybrid register instantiations with different chaining densities. The reduced chaining density used in our proposed design is also conducive to a more lightweight hardware implementation, since fewer logic gates are required to implement fewer chaining functions.

2) MPR Selection: Our 128-bit CMPR construction consists of the seven MPRs shown in Table I. While there are several distinct combinations of Mersenne exponents that sum to 128, we chose our CMPR configuration based on two main considerations: (i) linear complexity and (ii) ensuring there are several MPRs that are initialized to fixed values between the portion of the state containing IV bits and the MPR from which the keystream is extracted (we colloquially refer to such MPRs as "fixed MPRs"). In the case of consideration (i), using more MPRs in a CMPR construction results in a CMPR that can produce sequences with higher linear complexity [1]. Regarding consideration (ii), the presence of fixed MPRs is necessary for the design to resist black box polynomial-based cryptanalytic techniques such as cube attacks and cube testers [1] [5].

B. Initialization Rounds

We clock the 256-bit hybrid register 128 times for the keystream generator initialization phase and 128 times for the authentication generator initialization phase, for a total of 256 initialization rounds. This design choice is motivated by the NLFSR, as use of 128 initialization rounds in each initialization phase ensures that the state of the NLFSR is rotated over a full 128-bit cycle. The 128-bit CMPR does not benefit from a large number of initialization rounds: in [1], it was demonstrated that the degree and number of monomials required to represent the least-significant bit of a CMPR in terms of the initial state variables saturate rapidly with respect to initialization rounds due to the aggregated effect of the chaining functions, which directly affect many state bits each time the CMPR is clocked.

C. Key and IV Provisioning

In our proposed scheme, we initialize the 128 most-significant bits of the hybrid register with the secret key and the 96 following bits with the initialization vector. This design choice is motivated by the "feed-forward" structure of the hybrid register, as illustrated in Fig. 1. Since the chaining functions connect from one subregister to the next, the initial state variables in the most significant bits of the hybrid register propagate through more nonlinear chaining functions, eliminating the possibility of a linear or low-degree relationship between these initial state variables and the ciphertext.

D. Keystream Generation

For keystream generation, our design takes the XOR of state bits 7, 3, and 0. This is a simple output function similar to the keystream generation of TRIVIUM, which takes the XOR of six internal state bits, whereas designs like Grain-128AEADv2 and Espresso use more complex and nonlinear output functions for keystream generation. Since CMPRs possess embedded nonlinear logic in the form of chaining functions, we find it unnecessary to use a nonlinear output function depending on many state variables. Using a 3-input XOR as our output function also allows for a smaller hardware implementation while ensuring that the ciphertext depends on different parts of the internal state. Moreover, we extract the keystream from the smallest MPR in the construction (the 2-bit MPR) because the smallest MPR in the construction evolves unpredictably under the influence of all prior nonlinear chaining functions in the design.

Similar to Grain-128AEADv2, we restrict the number of keystream bits that can be generated from a single (key, IV) pair to 2^{81}. While the 128-bit CMPR on its own has an expected period of $0.45 * 2^{128}$ (which can be computed using the Chaining Period Theorem from [1]), a value much larger than 2^{81}, we impose the same keystream limitation as Grain-128AEADv2 due to using a building block from Grain-128AEADv2 in our design.

V. Security Analysis and Keystream Testing

A. Cryptanalysis

In [1], various security analyses were applied to CMPRs, assuming that the attacker possesses full knowledge of the CMPR construction, including the chaining functions, update polynomials, and primitive polynomials used. Thus, in this paper, we focus our cryptanalytic arguments on the hybrid register construction. CMPRs have been demonstrated to resist techniques such as algebraic attacks, fast algebraic attacks, correlation attacks, and linear cryptanalysis [1]. The resistance of CMPRs to the aforementioned cryptanalytic techniques can be largely attributed to the fact that the degree and quantity of monomials needed to model the input-output relation of a CMPR grow rapidly as the CMPR is clocked, due to the nonlinear chaining functions introducing high-degree terms from

larger MPRs to smaller MPRs. This phenomenon indicates that it is computationally infeasible to solve or closely approximate the system of high-degree nonlinear equations describing the state-to-state relationship of a CMPR.

Regarding black-box polynomial-based cryptanalytic techniques such as cube attacks and cube testers [5], CMPRs have also demonstrated resistance, given that for a given CMPR construction, the chaining functions only connect from one MPR to the next smallest MPR in the construction and that there are several MPRs whose states are initialized to fixed values between the MPR containing IV bits and the MPR from which the keystream is extracted [1]. The hybrid register construction proposed in this paper abides by the aforementioned chaining convention and initializes the states of M_7, M_5, M_3, and M_2 to all ones.

B. Keystream Testing using the NIST Statistical Test Suite

To analyze the statistical properties of keystreams generated by the hybrid register stream cipher, a keystream dataset for use with the NIST Statistical Test Suite [6] was generated from a pseudorandom (key, IV) pair. The dataset consisted of $10{,}000{,}000$ keystream bits to satisfy the minimum dataset size required to apply all tests in the NIST Statistical Test Suite. The datasets passed all 15 of the tests in the NIST Statistical Test Suite. This result implies that the keystreams generated by our hash functions exhibit pseudorandom behavior, at least from the perspective of the tests applied in the NIST Statistical Test Suite. However, it is crucial to note that the NIST Statistical Test Suite does not verify the security of the underlying design used to generate the test set, but rather whether the tested data can satisfy the rigorous statistical tests in the suite.

C. Key and IV Testing using Bit Contribution Tests

To evaluate the sensitivity of the hybrid register stream cipher to bit flips in the key and IV, we applied the bit contribution (B.C.) for key and IV tests [7]. These tests consist of a number of *trials*, where each trial entails generating two pseudorandom inputs (key and IV), holding one input fixed, flipping each bit of the other generated input, and computing the keystream for each bit-flipped variant of the non-fixed input. For an i-bit input and j-bit output, each trial contributes to the construction of an i x j *dependence matrix*, a matrix where the entry at index (i, j) represents how many times bit j of the output flips when bit i of the input is flipped, where the maximum value of any matrix entry is equivalent to the number of trials. In the case of the B.C. for key test, the IV is held fixed and the key bits are flipped, and in the case of the B.C. for IV test, the key is held fixed and the IV bits are flipped one at a time. The purpose of these tests is to ascertain whether a cryptosystem exhibits the properties of confusion and diffusion, meaning that outputs should (i) depend on several parts of the key and (ii) a single bit flip in the input should cause roughly 50% of the bits in the output to flip. For a cryptosystem that exhibits confusion and diffusion, we expect the dependence matrix values to be normally distributed. In our tests, we apply $10{,}000$ trials of the B.C. key and B.C. IV tests to the hybrid register stream cipher. The results indicate that all dependence matrix entries pass, with Figures 3 and 4 demonstrating that the dependence matrix values are approximately normally distributed for both tests. These results validate our choice of initialization rounds, at least from the perspective of the statistical properties of the stream cipher output.

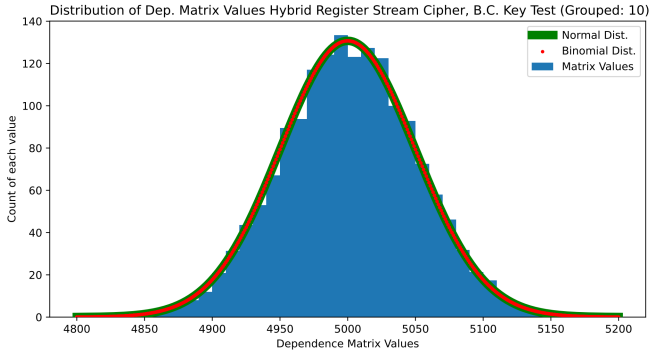

Fig. 3. Bit Contribution for Key Test Results for the Hybrid Register Stream Cipher

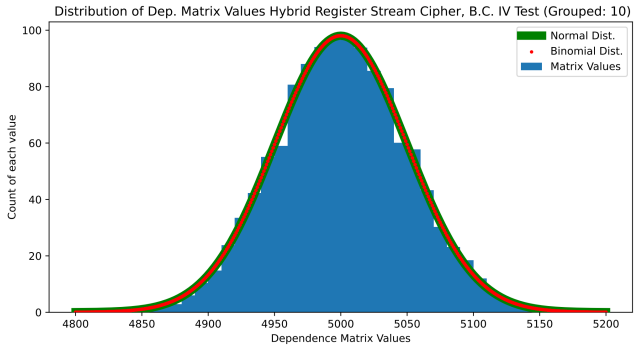

Fig. 4. Bit Contribution for IV Test Results for the Hybrid Register Stream Cipher

VI. Hardware Implementation Analysis

To determine how the hardware for the hybrid register stream cipher compares to TRIVIUM, Espresso, and Grain-128AEADv2, we synthesize RTL implementations of the hybrid register stream cipher and the comparison candidates onto an ASIC target and compare results for hardware area and energy consumption. Synthesis was performed using the Synopsys DesignCompiler synthesis tool [8]. Following synthesis, we perform place and route using the Cadence Innovus Implementation System [9] and layout using the Cadence Virtuoso Layout Suite [10]. The tools were configured to use the FreePDK45 45nm standard cell library [11]. The fastest clock frequency we could reliably obtain was 300MHz for all designs, and any attempts to use a higher frequency resulted in library hold time violations during synthesis.

Table II displays the security claims and authentication capabilities of the hybrid register stream cipher alongside the

comparison candidates, and Table III displays the latency of each cipher in clock cycles, or the number of clock cycles before the cipher begins to generate keystream bits. Table IV shows the ASIC hardware implementation results for the comparison candidates that do not natively support AEAD (TRIVIUM and Espresso), alongside an unauthenticated version of the hybrid register stream cipher (meaning the authentication generator of Fig. 2 is excluded from the implementation). Table V shows the ASIC hardware implementation results for the comparison candidate that natively supports AEAD (Grain-128AEADv2) alongside the hybrid register stream cipher. The area data in the "Area" column was obtained from the layouts generated by Virtuoso, and the power data in the "Power" column was extracted from DesignCompiler reports. The power data represents the average power consumed by the hardware during continuous operation. Since each clock cycle of the stream cipher implementation typically outputs one bit of keystream, this figure corresponds to the average power consumed during the generation of each keystream bit. Tables IV, V, and VIII also include the power-delay product (PDP) for each design in picojoules.

TABLE II
AUTHENTICATION CAPABILITIES AND SECURITY CLAIMS FOR THE HYBRID REGISTER STREAM CIPHER AND COMPARABLE REGISTER-BASED STREAM CIPHERS

Design	Authentication	Security
Hybrid Register	Yes	2^{128}
Grain-128AEADv2	Yes	2^{128}
Espresso	No	2^{128}
TRIVIUM	No	2^{80}

TABLE III
LATENCY COMPARISON FOR THE HYBRID REGISTER STREAM CIPHER AND COMPARABLE REGISTER-BASED STREAM CIPHERS

Design	Latency (Clock Cycles)
Hybrid Register	128
Grain-128AEADv2	512
Espresso	256
TRIVIUM	1152

TABLE IV
NON-AEAD ASIC HARDWARE IMPLEMENTATION COMPARISON BETWEEN THE HYBRID REGISTER STREAM CIPHER (EXCLUDING THE AUTHENTICATION GENERATOR) AND COMPARABLE REGISTER-BASED STREAM CIPHERS AT 300MHZ

Design	Area (μm^2)	Power (mW)	PDP (pJ)
Hybrid Register	4402	0.981	3.27
Espresso	5894	1.949	6.49
TRIVIUM	5627	3.0226	10.08

In both the non-AEAD and AEAD comparisond, the hybrid register stream cipher outperforms the comparison candidates in both area and power. The hybrid register stream cipher uses

TABLE V
AEAD ASIC HARDWARE IMPLEMENTATION COMPARISON BETWEEN THE HYBRID REGISTER STREAM CIPHER AND COMPARABLE REGISTER-BASED STREAM CIPHERS AT 300MHZ

Design	Area (μm^2)	Power (mW)	PDP (pJ)
Hybrid Register	5650	1.313	4.37
Grain-128AEADv2	6401	2.936	9.78

11.7% less hardware area and 55% less power than Grain-128AEADv2, 21.8% less hardware area and 67.5% less power than TRIVIUM, and 25.3% less hardware area and 49.7% less power than Espresso. We attribute the lower hardware area of the hybrid register to the following factors, though factor (i) is more substantial: (i) the hybrid register stream cipher utilizes a smaller internal state than TRIVIUM, which uses a 288-bit internal state, and (ii) the hybrid register stream cipher does not require a complicated nonlinear output function for keystream generation, whereas Grain-128AEADv2 and Espresso both employ nonlinear output functions. Regarding energy consumption, the favorable performance of the hybrid register stream cipher is owed to the low number of initialization rounds: the hybrid register stream cipher uses 128 initialization rounds, whereas TRIVIUM requires 1152 initialization rounds, Espresso requires 256 initialization rounds, and Grain-128AEADv2 requires 512 initialization rounds.

VII. SCALABILITY OF THE HYBRID REGISTER

The major advantage of the hybrid register construction is its scalability, which is mathematically validated by the theory of Composite Mersenne Product Registers [1]. Apart from the 128-bit NLFSR, there is a great deal of flexibility associated with the CMPR portion of the hybrid register construction. Heretofore, our specified design used the 128-bit CMPR of Table I. However, it is possible and straightforward to use larger or smaller CMPRs based on security and hardware efficiency needs. To showcase the flexibility and scalability of the hybrid register construction, we propose two larger variants of the hybrid register stream cipher: one variant that supports a $192-$bit key, and one variant that supports a $256-$bit key, where it is assumed that both variants use a $96-$bit IV. It is also possible to increase the IV size rather than the key size, or to increase both the key and IV sizes; however, our examples

TABLE VI
SPECIFICATION FOR THE 192-BIT CMPR

MPR Size	Update Polynomial	Primitive Polynomial
89 bits	$U_{89}(x) = x^6$	$P_{89}(x) = x^{89} + x^{38} + 1$
61 bits	$U_{61}(x) = x^3$	$P_{61}(x) = x^{61} + x^{44} + x^{19} + x^{15} + 1$
19 bits	$U_{19}(x) = x^4 + x^2$	$P_{19}(x) = x^{19} + x^5 + x^2 + x + 1$
13 bits	$U_{13}(x) = x^5 + x^4$	$P_{13}(x) = x^{13} + x^4 + x^3 + x^1 + 1$
5 bits	$U_5(x) = x^3$	$P_5(x) = x^5 + x^2 + 1$
3 bits	$U_3(x) = x^2$	$P_3(x) = x^3 + x + 1$
2 bits	$U_2(x) = x + 1$	$P_2(x) = x^2 + x + 1$

focus on increasing key size since the key length determines the bits of security. Both of our proposed larger variants utilize the 128-bit NLFSR from Section III.

To support a 192-bit key, we utilize a 192-bit CMPR. In total, the state size is 320 bits due to the presence of the 128-bit NLFSR. The 192-bit CMPR is specified in Table VI.

To support a 256-bit key, we utilize a 256-bit CMPR. In total, the state size is 384 bits (due to the presence of the 128-bit NLFSR). The 256-bit CMPR is specified in Table VII.

TABLE VII
SPECIFICATION FOR THE 256-BIT CMPR

MPR Size	Update Polynomial	Primitive Polynomial
107 bits	$U_{107}(x) = x^7$	$P_{107}(x) = x^{107} + x^{59} + x^{54} + x^{39} + 1$
89 bits	$U_{89}(x) = x^7 + x^6$	$P_{89}(x) = x^{89} + x^{38} + 1$
31 bits	$U_{31}(x) = x^3 + 1$	$P_{31}(x) = x^{31} + x^3 + 1$
17 bits	$U_{17}(x) = x^4$	$P_{17}(x) = x^{17} + x^3 + 1$
7 bits	$U_7(x) = x^5 + x$	$P_7(x) = x^7 + x + 1$
3 bits	$U_3(x) = x^2$	$P_3(x) = x^3 + x + 1$
2 bits	$U_2(x) = x + 1$	$P_2(x) = x^2 + x + 1$

We include ASIC hardware implementation results for the larger hybrid register constructions in Table VIII synthesized at 300MHz. As expected, these designs utilize more hardware due to the larger state size, in return for a larger key size, which provides more bits of security.

TABLE VIII
NON-AEAD ASIC HARDWARE IMPLEMENTATION RESULTS FOR THE LARGER HYBRID REGISTER STREAM CIPHER VARIANTS

Design	Area (μm^2)	Power (mW)	PDP (pJ)
320-bit Hybrid Register	6453	1.435	4.78
384-bit Hybrid Register	8367	1.511	5.03

VIII. DISCUSSION AND CONCLUSION

In this work, we introduced the hybrid register stream cipher, a lightweight AEAD-capable stream cipher that strikes a balance between hardware efficiency and security while also offering scalability for scenarios requiring different security parameters. Our proposed construction, based on a 128-bit NLFSR connected to a 128-bit CMPR, demonstrates that the integration of nonlinear feedback shift registers and Product Registers is conducive to desirable cryptographic properties while maintaining a modest hardware footprint. Furthermore, the theoretical proofs of [1], especially Theorem 1 (Chaining Period Theorem), apply to our hybrid register stream cipher.

Compared to TRIVIUM, Espresso, and Grain-128AEADv2, the hybrid register stream cipher reduces ASIC area and power. In the non-AEAD setting, the hybrid register stream cipher achieves 21.8% lower hardware area and 67.5% lower energy consumption relative to TRIVIUM, while in the AEAD setting, the hybrid register stream cipher achieves 11.7% lower hardware area and 55% lower energy consumption compared to Grain-128AEADv2, validating the hardware-friendly nature of our design choices. Statistical testing, including the NIST Statistical Test Suite and bit contribution tests from Section V, confirms the pseudorandomness and diffusion properties of the keystream output.

Beyond the baseline 128-bit security configuration, we demonstrated the scalability of the hybrid register structure by proposing larger variants supporting 192- and 256-bit keys, with even larger key sizes being possible by specifying a larger hybrid register or smaller IV. This flexibility, inherent to Composite Mersenne Product Registers, enables the construction of stream ciphers with varying security levels without significantly altering the core architecture or incurring prohibitive and unpredictable increases in hardware cost. Overall, the hybrid register stream cipher demonstrates that integrating NLFSRs and CMPRs can yield AEAD-capable stream ciphers that are both secure and hardware-efficient. Future work in this realm includes analyzing the vulnerability of the hybrid register stream cipher to side channel analysis, exploring the viability of the hybrid register construction for other cryptosystems such as MACs, hash functions, and block ciphers.

REFERENCES

[1] D. Gordon *et al.*, "Scalable nonlinear sequence generation using Composite Mersenne Product Registers," *IACR Commun. Cryptol.*, vol. 1, no. 4, Jan. 2025, doi: 10.62056/a3tx11zn4.

[2] C. De Cannière, "Trivium: A stream cipher construction inspired by block cipher design principles," in *Lecture Notes in Computer Science*, vol. 4377, A. Biryukov, Ed. Berlin, Germany: Springer, 2006, pp. 171–186, doi: 10.1007/11836810_13.

[3] E. Dubrova and M. Hell, "Espresso: A stream cipher for 5G wireless communication systems," *Cryptogr. Commun.*, vol. 9, no. 2, pp. 273–289, Dec. 2015, doi: 10.1007/s12095-015-0173-2.

[4] M. Hell *et al.*, "Grain-128AEADv2 – A lightweight AEAD stream cipher." [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/grain-128aead-spec-final.pdf.

[5] I. Dinur and A. Shamir, "Cube attacks on tweakable black box polynomials," in *Proc. EUROCRYPT*, Cologne, Germany, 2009, pp. 278–299, doi: 10.1007/978-3-642-01001-9_16.

[6] A. Rukhin *et al.*, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," NIST Special Publication 800-22 Rev. 1a, Apr. 2010. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf

[7] S. Pugh, S. Raunak, D. Kuhn, and R. Kacker, "Systematic testing of lightweight cryptographic implementations [Extended Abstract]." [Online]. Available: https://csrc.nist.gov/CSRC/media/Events/lightweight-cryptography-workshop-2019/documents/papers/systematic-testing-of-lightweight-crypto-lwc2019.pdf.

[8] Synopsys, *Design Compiler*. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html.

[9] Cadence, *Innovus Implementation System*, 2020. [Online]. Available: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/soc-implementation-and-floorplanning/innovus-implementation-system.html.

[10] Cadence, *Virtuoso Layout Suite*. [Online]. Available: https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/layout-design/virtuoso-layout-suite.html.

[11] NC State EDA, "FreePDK45." [Online]. Available: https://eda.ncsu.edu/freepdk/freepdk45/.