IACR Communications in Cryptology
ISSN 3006-5496, Vol. 1, No. 4, 77 pages.

https://doi.org/10.62056/a3tx11zn4

Check for updates

Scalable Nonlinear Sequence Generation using Composite Mersenne Product Registers

David Gordon, Arman Allahverdi, Simon Abrelat, Anna Hemingway, Adil Farooq, Isabella Smith, Nitya Arora, Allen Ian Chang, Yongyu Qiang and Vincent John Mooney III

Georgia Institute of Technology, Atlanta, United States of America

Abstract. We introduce a novel composition method that combines linear feedback registers into larger nonlinear structures and generalizes earlier methods such as cascade connections. We prove a Chaining Period Theorem which provides the cycle structure of these register constructions. We then use this Chaining Period Theorem and a new construction we call a Product Register (PR) to introduce a flexible and scalable register family with desirable properties, which we term Composite Mersenne Product Registers (CMPRs). We provide an algorithm to estimate the linear complexity of a chosen CMPR and investigate the statistical properties and security of a CMPR-based pseudorandom generator. Finally, we propose a family of CMPR-based stream ciphers and provide comparisons with the TRIVIUM stream cipher in terms of hardware area and security.

Keywords: LFSR · Cascade Connection · Cycle Structure · CMPR · Nonlinear Sequence Generation · Stream Cipher

1 Introduction

1.1 Background

There has been a long line of work focusing on the generation of binary sequences and understanding feedback registers. Early work in the construction of feedback registers with full period focused on de Bruijn sequences [dBru46] and LFSRs, with influential sources such as [Gol82]. From these roots, a variety of approaches appeared, including filter and combination generators [Rue86; Can05; Can11], cascade products [GD70; MST79], cycle joining and cross-join pairs [Dub14; MG16], and single-cycle T-functions [KS04; ZW06], among many others. Across these different approaches, there is often a common goal of identifying methods for creating feedback registers with long period and complex nonlinear updates, as these often yield useful components for lightweight cryptographic primitives. For instance, one of the smallest (in terms of microchip area) unbroken stream cipher proposals to date is TRIVIUM [De 06; DP08]. In summary, this field of inquiry is of both theoretical and practical importance.

E-mail: dgordon48@gatech.edu (David Gordon), aallahverdi3@gatech.edu (Arman Allahverdi), simon.abrelat@gatech.edu (Simon Abrelat), ahemingway6@gatech.edu (Anna Hemingway), afarooq32@gatech.edu (Adil Farooq), ismith80@gatech.edu (Isabella Smith), narora70@gatech.edu (Nitya Arora), allen.chang@gatech.edu (Allen Ian Chang), yqiang7@gatech.edu (Yongyu Qiang), mooney@gatech.edu (Vincent John Mooney III)

This work is licensed under a “CC BY 4.0” license.

Received: 2024-10-09 Accepted: 2024-12-03

1.2 Contributions

- We generalize older results for analyzing cycle structure and introduce a more general method of composition for register structures similar to cascades, filter generators, or combination generators, i.e., prior methods that were used to explore enhancing the strength of LFSRs.
- We introduce a new more general method called chaining, introduce our Chaining Period Theorem which proves the exponential period of registers produced via chaining, and show its application to a couple of different existing classes of register families.
- We then introduce a new class of registers with nonlinear state sequences based on chaining and prove some important and desirable properties. These properties include an exponential expected value of the register period and high linear complexity. We name this new class Composite Mersenne Product Registers.
- In the course of this analysis, we introduce a new algorithm to determine the linear complexity of our new chaining construct.
- We perform statistical analysis and cryptanalysis on a CMPR-based PRNG, indicating potential use for chaining-based structures in cryptographic applications.
- We design a family of CMPR-based stream ciphers, discuss the design rationale behind CMPR-based stream cipher design, and perform a hardware implementation comparison with the TRIVIUM stream cipher.

1.3 Organization

Section 2 covers mathematical notation and preliminaries. Section 3 states and proves the *Chaining Period Theorem*, the main theorem about chaining. Section 4 introduces a new family of registers based on the concept of chaining and proves results about the period of any register from this new family. Section 5 discusses an approach to linear complexity analysis and presents an algorithm for estimating linear complexity for our proposed family. Section 6 presents an application in the form of a PRNG, along with statistical analysis and cryptanalysis, indicating potential usefulness in cryptography. Also presented are (i) a cube attack that may have success with generic chaining but is prevented by appropriately restricting the chaining function and (ii) a key-independent distinguisher that can distinguish the output of a non-permuted CMPR from truly random bitstream (but is prevented by swapping the high and low order bits half way through the rounds, thus permuting the internal state of the CMPR). Section 7 covers the design and implementation of a family of CMPR-based stream ciphers, along with a hardware implementation comparison to TRIVIUM, a lightweight, hardware-oriented stream cipher. We end the paper with a discussion in Section 8 and conclusion in Section 9.

2 Preliminaries

2.1 Mathematical Notation

This paper utilizes a mathematical background consisting mostly of linear and abstract algebra, with some elements from control theory, and we attempt to make our notation as standard to those fields as possible. We denote the finite field with p elements as $\mathbb{F}_p$, where p is a prime number. We denote by $\mathbb{F}_{p^n}$ the extension field of $\mathbb{F}_p$ which has a cardinality of p^n, and we represent its elements as an n-tuple of the elements of $\mathbb{F}_p$. For a field $\mathbb{F}$, we denote the additive and multiplicative groups as $\mathbb{F}^+$ and $\mathbb{F}^\times$, respectively. $\mathbb{F}[x]$ denotes

the ring of polynomials with coefficients in the field $\mathbb{F}$. Given a particular element $\alpha \in \mathbb{F}$, the subgroup α generates is denoted as $\langle\alpha\rangle = \{\alpha, \alpha^2, \alpha^3, \dots\}$. A degree n polynomial P in $\mathbb{F}_p[x]$ is primitive when it is an irreducible polynomial such that $\langle\alpha\rangle = \mathbb{F}_{p^n}^\times$ for some root $\alpha \in \mathbb{F}_{p^n}$ of P. The notation $\oplus$ and $\times$ will be used to denote field operations, while $+$ will signify arithmetic addition. We may also omit the symbol for multiplication when the usage is unambiguous. For a group G, we denote the identity element as 1_G. Additionally, we denote that p divides n with $p \mid n$. We will also use the vertical bar in set-builder notation of the form $\{2n \mid n \in \mathbb{Z}\}$, but the usage should be clear from context.

Note that the notation for the extension field $\mathbb{F}_{p^n}$ is distinct from the notation $\mathbb{F}_p^n$ which represents the n-dimensional vector space over $\mathbb{F}_p$. $\mathbb{GL}(n,p)$ represents the general linear group of order n over $\mathbb{F}_p$, which contains all invertible $n \times n$ matrices with entries in $\mathbb{F}_p$ [Rot95, p. 13]. $\mathbb{GA}(n,p)$ denotes the general affine group of order n over $\mathbb{F}_p$ which contains all affine functions of the form $f(x) = Mx + b$, with $M \in \mathbb{GL}(n,p)$ and $b \in \mathbb{F}_p^+$.

2.2 Registers, Feedback, Cycles and Systems

In this section, we will discuss a variety of historical constructions and some of their properties. Because we are giving an overview, we will start with more general definitions and then narrow down to more specific types. We will also approach this topic from a more mathematical perspective, with less of a focus on implementation details.

Definition 1 (System)**.** In this paper, we use the word system as a shorthand for a discrete-time finite-state dynamical system. Formally, we define this to be a pair $(\mathcal{S}, f)$, where $\mathcal{S}$ is a finite set of states for the system and $f : \mathcal{S} \to \mathcal{S}$ is a function which describes how the system evolves at each time step. If f is bijective, we may refer to either f, or the system as a whole, as *nonsingular*.

For a system S, we denote by $S[t]$ the state of the system at time t, with the initial state of the system being at $t = 0$. $\{S[t]\}$ denotes the sequence of states that the system goes through according to an update function f.

Definition 2 (Cycle Structure)**.** For a nonsingular system $(\mathcal{S}, f)$, f partitions $\mathcal{S}$ into cycles; we say that $s_1, s_2 \in \mathcal{S}$ are in the same cycle if $f^k(s_1) = s_2$ for some $k \in \mathbb{N}$. We define the cycle structure of this system as the list of the sizes of the cycles produced, counted with multiplicity.

Definition 3 (Period)**.** The period of a system S is the minimum value p such that $S[t+p] = S[t]$ for all t, given that the system S starts within the sequence defined by the p states with the specified period. Note that for a nonsingular system, S, the period of the system when started in state $S[0] = s \in \mathcal{S}$ is equal to the size of the cycle which contains s.

Definition 4 (Decimation)**.** A k-decimation of the state sequence $\{S[t] \mid t \geq 0\}$ is the sequence $\{S[kt] \mid t \geq 0\}$. When k is unknown, we will refer to this as a decimation. Notably, if the period of the system is p, then the k-decimation will create $\gcd(k,p)$ cycles of length $p \,/\, \gcd(k,p)$.

In practice, for systems implemented in hardware or software, the set of states usually corresponds to the states of a register and the update function to some combinational feedback logic.

Definition 5 (Feedback Register)**.** An n-bit feedback register (FR) is a register A such that on each clock cycle, the state is updated according to some fixed function $f : \{0,1\}^n \to \{0,1\}^n$:

$$A[t+1] = f(A[t])$$

This is a system with states $\mathcal{S} = \{0,1\}^n$ and update function f. We will refer to the update of the ith bit of the register as f_i.

For a register, we refer to the value of an individual bit at time t using a lowercase letter with a subscript. For instance, the value of the ith bit of register A at time t would be $a_i[t]$. The state of an n-bit register at any given point in time, $A[t]$, is defined as the entire sequence of values held in the bits $a_{n-1}[t] \ldots a_1[t]a_0[t]$. There are natural ways to associate this register state with several different algebraic objects. An n-bit binary state $A[t]$ can easily be identified with a vector in $\mathbb{F}_2^n$ or with a polynomial in $\mathbb{F}_2[x]$ of degree $n-1$ or less. One way to do this is for each bit $a_i[t]$ to correspond to the coefficient of x^i in the polynomial. For example, the state $A[t] = 101001$ could correspond to polynomial $x^5 + x^3 + 1$. Because $\mathbb{F}_{2^n} \cong \mathbb{F}_2[x]/P(x)$ for any primitive polynomial $P(x)$ of degree n, $A[t]$ can also be identified with an element of $\mathbb{F}_{2^n}$ if a primitive polynomial is chosen. It is useful to be able to take any of these views, depending on which lens is most convenient for a particular analysis, and we will swap between them freely.

Definition 6 (Linear Feedback Register)**.** A Linear Feedback Register is a feedback register, for which the update function f is linear, in the sense that it satisfies

$$f(a \oplus b) = f(a) \oplus f(b) \text{ for any } a, b \in \mathbb{F}_2^n$$

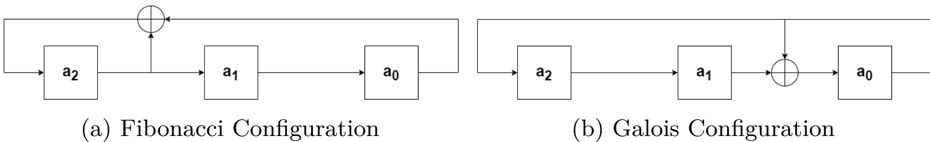
Notably, the feedback function for a linear feedback register can be implemented using only XOR gates. Additionally, the feedback of any nonsingular linear feedback register can also be represented by a matrix $U \in \mathbb{GL}(n, 2)$.

Definition 7 (Feedback Shift Register)**.** A feedback shift register (FSR) is a feedback register A such that for each $i \in \{1, \ldots, n-1\}$ the output of each bit a_i is connected to the input of the next bit a_{i-1} either directly or through an XOR gate.

Definition 8 (Linear Feedback Shift Register)**.** A Linear Feedback Shift Register (LFSR) is a register that is both a feedback shift register and a linear feedback register.

There are two main LFSR styles. In the Fibonacci or "external-XOR" configuration [Abr94, p. 434], $a_i[t+1] = a_{i+1}[t]$ for all $i \in \{0, \ldots, n-2\}$, while the most significant bit updates according to some XOR of bits in the state. This means only one new bit is generated per clock cycle while the remaining bits shift. Because only one new bit is generated on each clock cycle, the update functions are typically described as being $\mathbb{F}_2^n \to \mathbb{F}_2$ instead of $\mathbb{F}_2^n \to \mathbb{F}_2^n$. In the Galois or "internal-XOR" configuration [Abr94, p. 434], $a_0[t]$ is XORed with the feed-in to one or more bits in the state. There are sometimes implementation advantages to using a register in one configuration (e.g., Galois) over the other (e.g., Fibonacci). There is also a connection between the two configurations: for any LFSR A in one configuration which generates sequence $\{a_0[t]\}$, there is a dual LFSR B in the opposite configuration which generates the same output sequences $\{b_0[t]\}$. A technicality worth noting is that the full states $A[t]$ and $B[t]$ need not be equal, just the least significant bits $\{a_0[t]\}$ and $\{b_0[t]\}$.

Often LFSRs are designed using primitive polynomials over $\mathbb{F}_2$ to determine which bits are XORed in the feedback function. For an n-bit LFSR, this implies a period of $2^n - 1$; such an LFSR is called *full period.* Figure 1 shows a full period 3-bit LFSR in both the Fibonacci and Galois configurations.



(a) Fibonacci Configuration (b) Galois Configuration

Figure 1: LFSR in Both the Fibonacci and Galois Configuration.

LFSRs are often used in applications where security is not a concern or as parts of more complicated cryptographic functions [SM08; AHMN12]. The values are taken from

the least significant bit of an LFSR as it cycles form a sequence with good statistical properties. However, the linearity of LFSRs also leads to cryptographic vulnerabilities.

Definition 9 (Nonlinear Feedback Shift Register)**.** A Nonlinear Feedback Shift Register (NLFSR) is an FSR, for which the update function f is nonlinear.

Most research on NLFSRs has used the Fibonacci configuration, although there has been some work on converting existing Fibonacci NLFSRs to Galois-like configurations [Dub09]. In general, the properties and guarantees of NLFSRs are much harder to analyze, and finding constructions of NLFSRs with large periods is nontrivial.

2.3 De Bruijn Sequences

Definition 10 (De Bruijn Sequence)**.** A de Bruijn sequence of order n over an alphabet $\mathcal{A}$ is a sequence with period $|\mathcal{A}|^n$, such that every n-tuple of letters from $\mathcal{A}$ appears exactly once in each period of the sequence [dBru46].

In this paper, we are only concerned with binary de Bruijn sequences. These are the de Bruijn sequences over $\{0,1\}$ where n bits have period 2^n. These de Bruijn sequences are linked to Fibonacci-style shift registers; for example, an NLFSR with n bits and period 2^n outputs a de Bruijn sequence.

Definition 11 (Cascade Connection)**.** A cascade connection is a method for combining two Fibonacci-style feedback registers in which the output of the first register is XORed with the feedback which determines the new bit for the second Fibonacci-style feedback register.

Definition 12 (Cascade Product)**.** Let $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$ be two Fibonacci-style feedback functions. The cascade product of f and g is defined to be the function

$$(g * f)(x_1, \dots, x_{n+m-1}) = g(f(x_1, \dots, x_n), f(x_2, \dots, x_{n+1}), \dots, f(x_m, \dots, x_{n+m-1}))$$

This gives the feedback function of a Fibonacci FSR which generates the equivalent output as the cascade connection of f and g. This product is not commutative in general.

Due to the cascade product, in early papers such as [GD70], these constructions were called product shift registers. This construction is distinct from the concept of a "product register" later defined in this paper.

Property 1. Let A be an m-bit Fibonacci-style FSR with nonsingular feedback function g, and let B be an n-bit Fibonacci-style LFSR with a feedback function f corresponding to an irreducible characteristic polynomial. Denote the period of B as p_f. Let R denote the composite system formed via the cascade connection of A into B with feedback function $g * f$. Then the cycle structure of R can be determined as follows:

Let $\{a_0[t]\}$ be a cycle with period p_a, generated by the least significant bit of A. Then,

- if $p_f \mid p_a$ then the corresponding cycle is of length p_a or $2p_a$.
- if $p_f \nmid p_a$ then there are $2^n - 1$ cycles of length $\mathrm{lcm}(p_f, p_a)$, and one cycle of length p_a.

Additionally, Mykkeltveit et al. give a criterion to distinguish between cases with period p_a versus $2p_a$ [MST79, Theorem 3.15].

This property of cascades makes them useful to work with for extending existing constructions. As a modern example of this, [MG16] and [CGW20] both involve the use of the cascade product to extend existing NLFSRs. Theorem 1, the main theorem of this paper which is introduced in Section 3, will generalize Property 1.

2.4 T-Functions

Definition 13 (T-Function)**.** An n-bit T-Function is a function $f : \{0,1\}^n \to \{0,1\}^n$ in which the ith output depends only on the input bits with indices $\leq i$ [KS04].

Many common word-oriented operations (e.g., integer addition and multiplication mod a power of 2, as well as many Boolean operations) fall into this category of functions. Because of this, T-functions are usually utilized in software-focused environments that already have fast implementations of these operations. Thus, they are not usually considered in the same context as FSRs. However, they can be represented in a feedback register and thus form an interesting point of comparison in the context of this paper.

Property 2. Let f_n be an n-bit invertible T-function, and let $f_{n-1} : \{0,1\}^{n-1} \to \{0,1\}^{n-1}$ be the invertible T-function formed by the first $n-1$ bits of f_n. Then for each cycle under f_{n-1} of length p, there are either two cycles of length p or one cycle of length $2p$ under f_n [KS04, Lemma 3].

Note the similarity of this property to the first case of Property 1. However, Property 2 cannot be proved directly from Property 1 because of the structural differences between T-functions and Fibonacci LFSRs.

Property 3. Let $f : \{0,1\}^n \to \{0,1\}^n$ be an invertible T-function. Then f has a single cycle if and only its Algebraic Normal Form (ANF) has the form

$$f_0(x_0) = x_0 \oplus 1 \qquad\qquad f_k(x_0, \ldots, x_k) = x_k \oplus \prod_{i=0}^{k-1} x_i \oplus \psi(x_0, \ldots, x_{k-1})$$

where ψ is any boolean function with algebraic degree less than k [ZW06].

Property 3 gives a nice criterion for creating large-period nonlinear systems. T-functions form an extremely big class of large-period nonlinear structures and are interesting components from a theoretical point of view, although they have cryptographic weaknesses and some disadvantages for hardware implementation.

2.5 TRIVIUM

TRIVIUM is a lightweight, hardware-oriented synchronous stream cipher whose design was motivated by the exploration of the tradeoff between design simplification and cryptographic security [De 06; DP08]. Moreover, TRIVIUM is a shift-register-based stream cipher, with a 288-bit NLFSR at the heart of its design. While its use is discouraged in sensitive applications, TRIVIUM has yet to be fully compromised by cryptanalytic attacks. However, variants of TRIVIUM with lowered security parameters have been successfully attacked [LH24].

3 Chaining Period Theorem

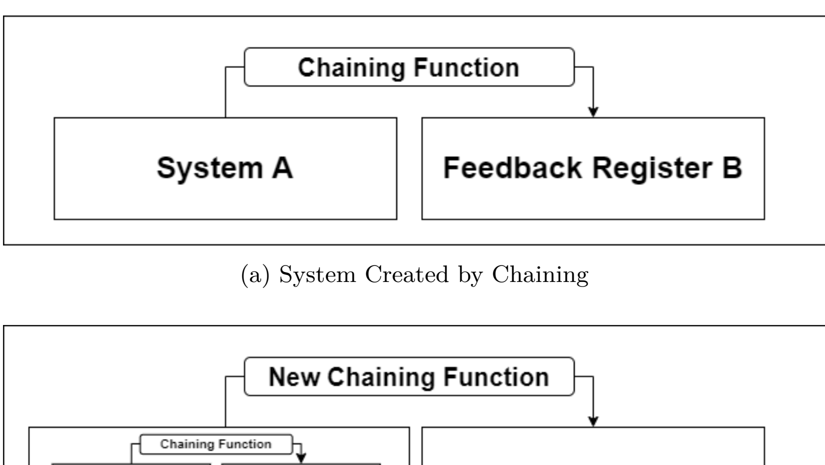
3.1 Chaining

Cascade constructions and invertible T-functions are similar in that there is a one-directional manner in which some components affect other components. Cascade constructions connect Fibonacci FSRs using only 1-bit linear connections. T-functions, on the other hand, have much more complicated connections in which the update of each bit may depend on nonlinear functions with multiple inputs. However, each of these functions only affects one bit, and thus the components themselves are simpler. We introduce the following concept to capture both notions and generalize the idea of a one-directional dependence on another system.

Definition 14 (Chaining Function)**.** Let A be a deterministic system with states $A[t] \in \mathcal{S}$. Let B be an n-bit feedback register that updates linearly according to the matrix $U \in \mathbb{GL}(n, 2)$. The chaining function $\mathcal{C} : \mathcal{S} \to \mathbb{F}_2^n$ describes a connection that allows A to change B via the following altered update equation:

$$B[t + 1] = (UB[t]) \oplus \mathcal{C}(A[t]) \tag{1}$$

When connected by a chaining function, A and B can be considered as part of a larger system with states in $\mathcal{S} \times \mathbb{F}_2^n$. Importantly, even if $\mathcal{C}$ is nonlinear, the altered update of register B, Equation 1, is an affine function and thus can be represented by an element of $\mathbb{GA}(2, n)$ acting on the state of B.



(a) System Created by Chaining

(b) Repeated Chaining

Figure 2: Abstract Diagram of Register Chaining

Figure 2a shows an abstracted diagram of chaining. The entire system created by chaining a register can then be reused, with a new chaining function, as shown in Figure 2b. There are a wide variety of constructions that are derivable from chaining together smaller systems in this fashion; cascade connections and T-functions represent well-known special cases of these. In this section, we will derive a result concerning the cycle structure of registers that have been chained together. This result generalizes Properties 1, 2 and 3 from Subsections 2.3 and 2.4. We will start by building up linear algebra and group theory necessary to state the result.

In what follows, when we refer to the characteristic polynomial of an invertible matrix, we mean the polynomial formed by taking the determinant of the matrix whose diagonal values are replaced by the original diagonal values minus a variable λ, and where the roots of this characteristic polynomial (the values of λ at which the characteristic polynomial evaluates to zero) correspond to the eigenvalues of the matrix [Leo09].

Lemma 1. *Let $U \in \mathbb{GL}(n, 2)$ be an invertible $n \times n$ matrix over $\mathbb{F}_2$, whose characteristic polynomial is irreducible. Then, for any $k \in \mathbb{N}$, U^k fixes either only the zero vector or the entire space $\mathbb{F}_2^n$.*

Proof. First, since the characteristic polynomial of U is irreducible, U has no nontrivial invariant subspaces (i.e., none other than $\{0\}$ and $\mathbb{F}_2^n$). Then let $E(1, U^k)$ be the eigenspace

of U^k corresponding to eigenvalue $\lambda = 1$. Commuting operators preserve each other's eigenspaces, so $E(1, U^k)$ is invariant under U and thus it can only be $\{0\}$ or $\mathbb{F}_2^n$. Since $E(1, U^k)$ is precisely the subspace of vectors fixed by U^k, this proves the claim. □

Definition 15 (Group Action)**.** A (left) group action is a function that takes as input a group G and a set X and has the following properties. Let 1_G be the identity element of G, then $\alpha : G \times X \to X$ is a (left) group action if

(i) $\alpha(1_G, x) = x$ for every $x \in X$, and

(ii) $\alpha(g, \alpha(h, x)) = \alpha(gh, x)$ for every $g, h \in G$ and $x \in X$.

Note 1. Since there are non-commutative groups, there are also right group actions. We only use left group actions and will omit the word left. For brevity, we will use the notation $\alpha(g, x) = g \cdot x$ when α is clear from context, with the understanding that $g \in G, x \in X$.

To define Frobenius groups, we briefly restate the definitions of transitive and faithful group actions. Transitive group actions are those for which for any two elements $x, y \in X$, there exists some element $g \in G$ where $y = g \cdot x$ [Rot95, p. 58]. Faithful group actions are those in which for $g \cdot x = x$ for all $x \in X$ only when $g = 1_G$ [Rot95, p. 248].

Definition 16 (Frobenius Group)**.** A group G is a Frobenius group if G has a faithful transitive group action on a set X with $|X| > 1$, where

(i) $\mathrm{Stab}(x) \neq \{1_G\}$ for all $x \in X$, and

(ii) $\mathrm{Stab}(x) \cap \mathrm{Stab}(y) = \{1_G\}$ for all $x, y \in X$.

Here, $\mathrm{Stab}(x) = \{g \in G \mid g \cdot x = x\}$ is the stabilizer subgroup of $x \in X$ [Gro10, p. 171].

Lemma 2. *Let $U \in \mathbb{GL}(n, 2)$ be an invertible $n \times n$ matrix over $\mathbb{F}_2$, whose characteristic polynomial is irreducible. Let G_U be the subgroup of $\mathbb{GA}(n, 2)$ containing functions of the form $g(x) = U^i x + c$ for any $i \in \mathbb{N}$ and $c \in \mathbb{F}_2^n$. Then, with the operation of function composition, G_U forms a Frobenius group with a Frobenius group action on $\mathbb{F}_2^n$.*

Proof. Let G_U act on $\mathbb{F}_2^n$ by $g \cdot v = g(v)$. This action is transitive because for any $v, w \in \mathbb{F}_2^n$, we can take $g(x) = x \oplus (v \oplus w) \in G_U$, which satisfies

$$g \cdot w = g(w) = w \oplus v \oplus w = v.$$

It is also faithful. Fix any $g(x) = U^i x \oplus c \neq 1_{G_U}$. Then either $c \neq 0$, or $U^i \neq I$. If $c \neq 0$, then $U^i c \neq 0$ since U is invertible, and thus $g \cdot c = U^i c \oplus c \neq c$. In particular, this says that g does not fix c. If instead $c = 0$, then $U^i \neq I$, and so U^i does not fix any nonzero points by Lemma 1. Thus, the only element that fixes all vectors is 1_G, i.e., the action is faithful.

For property (i) of Definition 16, note that any particular $v \in \mathbb{F}_2^n$ is fixed by the non-identity element $g = Ux \oplus (Uv \oplus v) \neq 1_{G_U}$. Now we demonstrate property (ii) of Definition 16. Let some group element $g = U^i x \oplus c$ stabilize two elements $u, v \in \mathbb{F}_2^n$, with $u \neq v$. Then

$$u \oplus v = (U^i u \oplus c) \oplus (U^i v \oplus c) = U^i u \oplus U^i v = U^i (u \oplus v)$$

since $g \cdot u = u$ and $g \cdot v = v$. Note that $u \oplus v \neq 0$ since $u \neq v$, so U^i fixes a nonzero vector. Thus by Lemma 1, U^i fixes all vectors, i.e., $U^i = I$. But $g(x) = U^i x \oplus c = Ix \oplus c$ fixes u, so we must have $c = 0$ since $u = g \cdot u = u \oplus c$. Thus $g = I$ must be the identity element of the group. Since g was arbitrary, the intersection of any two distinct stabilizer subgroups is only the identity. Thus this group is Frobenius, as claimed. □

Property 4. Within a Frobenius group, G, there are two subgroups H and N such that any element $g \in G$ can be expressed as $g = nh$ with $n \in N, h \in H$. H is called the *Frobenius complement* which is a subgroup that fixes some element $x \in X$ such that $H = \mathrm{Stab}(x)$. N is called the *Frobenius kernel*, which is the subgroup $N^* \cup \{1_G\}$ where N^* is the set of elements in G that have no fixed points. The Frobenius kernel is normal in G, and all Frobenius complements are in the same conjugacy class [Gro10, p. 180].

In the case of G_U, the Frobenius kernel is formed by the set of functions $N = \{Ix \oplus c \mid c \in \mathbb{F}_2^n\}$, which is isomorphic to the additive group of $\mathbb{F}_{2^n}$ and to $\mathbb{Z}_2^n$. The Frobenius complement is formed by the function $H = \{U^i x \oplus 0 \mid i \in \mathbb{N}\}$ and is isomorphic to $\mathbb{Z}_p$ where p is the period of U. For any element of G_U, $g = (U^i \oplus c) = (x \oplus c) \circ (U^i x) = nh$, where $n = (x \oplus c) \in N$ and $h = (U^i x) \in H$, as claimed. For our purposes, the important implication of this property is that for any U^i which is not equal to the identity matrix, $U^i \oplus c$ is conjugate to U^i.

Consider the update of an n-bit register by U, with influence from a chaining function, as described by Equation 1. Regardless of the value output by the chaining function, any update of this form is an element of G_U, which in turn is a subgroup of $\mathbb{GA}(n, 2)$. Thus, the behavior of the register can be modeled by a group action of G_U on $\mathbb{F}_2^n$ in which $g \cdot x = g(x)$. We are interested in observing how these effects on the state accumulate throughout the chaining, which motivates the following definition.

Definition 17 (Cumulative Update Function)**.** Consider an n-bit register R that updates by U, with influence from a chaining function. Accounting for chaining, the update function at time t is given by Equation 1. This update function is affine and is associated with the element in G_U, which we call g_t. We define the cumulative update function $f_t \in G_U$ to be the composition of all g_i where $i < t$. In other words, $f_t = g_{t-1} \circ g_{t-2} \circ \ldots \circ g_0$. The update of R by g_t and f_t are defined as follows:

$$R[t+1] = g_t \cdot R[t] = g_t(R[t]) \text{ and}$$
$$R[t] = f_t \cdot R[0] = f_t(R[0]).$$

Theorem 1 (Chaining Period Theorem)**.** *As shown in Figure 2a, let $A = (\mathcal{S}, f)$ be a nonsingular system, and let B be an n-bit register with linear feedback represented by the matrix $U \in \mathbb{GL}(n, 2)$. Let the characteristic polynomial of U be irreducible, and let B have period m. Let $\mathcal{C} : \mathcal{S} \to \mathbb{F}_2^n$ be some chaining function from A to B, and let R denote the composite system formed by A and B via chaining function $\mathcal{C}$. Then the cycle structure of R can be determined from the cycles of A and B as follows:*

For any cycle of length p in A:

1. *if $m|p$ and $f_p = 1_{G_U}$ then there are 2^n cycles of length p*
2. *if $m|p$ and $f_p \neq 1_{G_U}$ then there are 2^{n-1} cycles of length $2p$*
3. *if $m \nmid p$ then there are $\frac{2^n-1}{m}\gcd(m, p)$ cycles of length $\mathrm{lcm}(m, p)$, and one cycle of length p*

Proof. The state of A only repeats every p steps, by definition, and thus the state of R can also only repeat on multiples of p. Also note that the series of inputs from the chaining function $\mathcal{C}$ is periodic with period p, and thus $f_{kp} = f_p^k$ for any k and R has period kp if and only if $B[0]$ has period k under the iteration of f_p. Due to this, we are primarily concerned with understanding the behavior of register B under the iteration of the cumulative update function f_p.

In the first two cases, $m \mid p$. Thus, $f_p = U^p \oplus c = Ix \oplus c$ for some $c \in \mathbb{F}_{2^n}$, implying f_p is an XOR operation. These operations are part of the Frobenius kernel and are not conjugate to any of the Frobenius complements.

Case 1. If $f_p = Ix \oplus 0$ then f_p is the identity function, or equivalently an XOR with 0. Accordingly, $B[p] = 1_{G_U} \cdot B[0] = B[0]$. Because the period of A is also p we have that the entire state of R will repeat at time p, but not before, and thus R has period p. Note that there are 2^n initial states of B, each of which yields a unique cycle of length p.

Case 2. If $f_p = Ix \oplus c$, with $c \neq 0 \in \mathbb{F}_{2^n}$, then f_p is an XOR operation, or equivalently, vector addition in $\mathbb{F}_2^n$. Then

$$(f_p)^2 = I(Ix \oplus c) \oplus c) = Ix \oplus c \oplus c = 1_{G_U}$$

as desired. Accordingly, $B[2p] = 1_{G_U} \cdot B[0] = B[0]$, but $B[p] = f_p \cdot B[0] = B[0] \oplus c \neq B[0]$. Because the period of A is p we have that the entire state of R will repeat at time $2p$, but not before, and thus R has period $2p$. Note that for any initial state $R[0] = A[0] \mid\mid B[0]$, the state $f_p \cdot R[0] = A[0] \mid\mid (B[0] \oplus c)$ will be in the same cycle. The 2^n initial states of B can then be split into 2^{n-1} such pairs, so there are 2^{n-1} cycles, each of length $2p$.

Case 3. If $m \nmid p$ then $f_p = U^p \oplus c$ for some c, and some $U^p \neq I$. By Property 4, f_p is conjugate to U^p. This can be given even more explicitly as

$$f_p = U^p \oplus c = (Ix \oplus \hat{c}) \circ (U^p x \oplus 0) \circ (Ix \oplus \hat{c})$$

where $\hat{c} = (U^p \oplus I)^{-1} c$. By this similarity, the cycle structure of B under f_p is the same as the cycle structure of B under U^p, although the cycles themselves contain different elements. Since this is the same as a p-decimation of the normal operation of B, this is well understood.

Because U has an irreducible characteristic polynomial and period m, Lemma 1 applies; thus, the nonzero states of B are partitioned into cycles with length m. Thus, the cycles of B are as follows: there is 1 fixed point (the zero state) and $\frac{2^n-1}{m}$ cycles of length m. For each of these length m cycles, U^p creates $\gcd(m,p)$ cycles of length $\frac{m}{\gcd(m,p)}$, and thus under U^p (and equivalently under f_p) the cycle structure consists of one cycle of length 1 and $\frac{2^n-1}{m}\gcd(m,p)$ cycles of length $\frac{m}{\gcd(m,p)}$. Each step in one of these cycles under f_p represents the change after a full period of A. The length of each of these cycles represents the first multiple of p at which A and B repeat at the same time, which is the first time R repeats. Multiplying by p, corresponding to this cycle of length p in A, the cycle structure of R has $\frac{(2^n-1)}{m}\gcd(m,p)$ cycles of length $\frac{mp}{\gcd(m,p)} = \operatorname{lcm}(m,p)$, and one cycle of length p, as claimed. □

Theorem 1 may be applied to every cycle in A to determine the full cycle structure of R. Knowing this full cycle structure, the process may be repeated inductively to understand the cycle structure of any construction created via chaining.

Remark 1. There are ways to decompose the matrix into block forms that preserve the invariant factors of the characteristic polynomial (such as the Jordan normal form). We believe that this result could be extended to the case where the characteristic polynomial is not irreducible, but that this would be equivalent in power to the induction on the components using this version.

3.2 Examples and Applications

In this section, we wish to highlight part of how Theorem 1 generalizes previous results. The first point of note is that Theorem 1 applies to LFSRs with primitive polynomials in both the Fibonacci and Galois configurations or decimations thereof. There are no special constraints placed on either System A or the chaining function, thereby providing for extreme generality and applicability, particularly because nonlinear functions are allowed.

Example 1. In [GD70], the authors form a cascade from a 4-bit NLFSR that generates two cycles with lengths 1 and 15 into a 2-bit Fibonacci LFSR with period 3.

Applying Theorem 1 to the cycle of length 1, we see that $3 \nmid 1$, and this corresponds to $\frac{2^2-1}{3} \gcd(3,1) = 1$ cycle of length $\text{lcm}(3,1) = 3$ and one cycle of length $p = 1$.

Applying Theorem 1 to the cycle of length 15 we see that $3 \mid 15$, and this corresponds to either $2^2 = 4$ cycles of length $p = 15$ or $2^{2-1} = 2$ cycles of length $2p = 30$.

Thus, in total, we have that the resulting cycle structure of this process will always be either $\{30, 30, 3, 1\}$ or $\{15, 15, 15, 15, 3, 1\}$. This is confirmed in Green and Dimond's experiments. This observation can also be explained by Proposition 1.

However, Theorem 1 goes further. Imagine a more complex system formed by adding a new component via a more complicated chaining function: e.g., suppose that a nonlinear filter is applied to the 6 bits of both the NLFSR and the LFSR, and at each time step, the output of the filter is XORed with the most significant bit of a 5-bit LFSR in the Galois configuration. Property 1 would be unable to analyze the cycle structure of such a configuration, as the functions do not form a simple cascade. However, noting that 31 does not divide $1, 3, 15$, or 30, and is coprime to all, we can apply Theorem 1 to immediately determine that the only possible cycle structures for this 11-bit construction are $\{930, 930, 93, 31, 30, 30, 3, 1\}$ and $\{465, 465, 465, 465, 93, 31, 15, 15, 15, 15, 3, 1\}$, regardless of the filter function used, and depending only on the cycle structure of the first 6 bits.

The end of the previous example shows how we can use chaining functions to create nonlinear cascades. Applying this view to T-functions, we can consider an invertible T-function as being a series of chained 1-bit registers, each with irreducible characteristic polynomial $x \oplus 1$. In [MST79], Mykkeltveit et al. also analyze cascades using 1-bit registers well before the introduction of T-functions but due to the limitations of Property 1 are unable to connect them nonlinearly as T-functions do. Instead, they take the cascade product $f * (x \oplus 1)$ and apply cycle-joining techniques. The end result of this is the creation of a de Bruijn sequence of length 2^{n+1} from a sequence of length 2^n. Similarly, each additional bit in a single-cycle T-function creates a cycle of length 2^{n+1} from a cycle of length 2^n. Because T-functions do not maintain the Fibonacci structure, the sequences generated by the least significant bit are not de Bruijn sequences in general.

Example 2. Because Theorem 1 allows for nonlinear chaining functions, Theorem 1 can be used to simply re-derive Properties 2 and 3. Let us begin with Property 2.

In the base case, the first bit in a single cycle T-function must oscillate between 1 and 0. If it remains fixed, the T-function will obviously not generate all states.

For each additional bit we add after this, we encounter the same situation: the bit added has characteristic polynomial $x+1$ which is irreducible and has two cycles of period 1. 1 always divides the period of any cycle, so this addition either generates 2 cycles of size p or 1 cycle of size $2p$, regardless of the chaining function used. This verifies Property 2.

Now consider Property 3. The condition in Theorem 1, $f_p = 1_{G_U}$, allows us to use the structure of G_U to analyze the conditions for each case. In this case, $G_U \cong \mathbb{Z}_2$ (i.e., the cyclic group of order 2), consisting only of the elements $1x \oplus 0$ and $1x \oplus 1$. At any point in time, the constant c in the cumulative update function $f_t = 1x \oplus c$ represents the parity of the outputs of the chaining function. The period doubles if and only if this parity is odd. If the previous register has full 2^n period, then in one period the chaining function is evaluated on all possible 2^n inputs, and so this is equivalent to the truth table of the chaining function having odd weight. This only occurs if the ANF of the chaining function includes a term including the product of all previous bits, verifying Property 3.

Theorem 1 can sometimes be used to analyze the cycle structure when the rule given by Property 3 is not obeyed. Imagine a single-cycle T-function with n bits following the

rule and using chaining functions to add 1 bit that does not follow the rule and then a final bit that does follow the rule again. In total, this creates a $(n+2)$-bit T-function. The first n bits generate a single cycle of length 2^n. Because the next bit does not follow the rule, it generates 2 cycles of length 2^n. The final bit follows the rule again, and thus its truth table has an odd number of 1's. These cannot be split evenly between the two cycles, so one cycle must have an odd weight and the other even. Only the odd cycle will double. Thus, the final cycle structure has 3 cycles, with lengths $\{2^{n+1}, 2^n, 2^n\}$. This is a powerful result, given that this derived cycle structure is independent of the actual functions chosen, provided they meet the conditions.

In conclusion, concerning most of the prior work relating to cascade products, we prove a more general result because we do not impose any restriction on either the system generating the inputs or the complexity of the function connecting the two. This allows for highly nonlinear cascades, and the resulting family of constructions is very large due to the flexibility of the construction.

4 PRNGs Based on Chaining

4.1 Product Registers and Mersenne Primes

One of the major implications of Theorem 1 introduced in the previous section is that chaining functions can be used to extend an existing system repeatedly, with both highly varied nonlinear logic and predictable effects on the period and cycle structure. In the rest of this paper, we propose a large family of feedback registers (Definition 5 in Subsection 2.2) based on Theorem 1 and what we consider to be some of the more natural design choices. We will also prove some properties of this construction. While there are certainly other design choices that could be made and analyzed using similar methods or frameworks, we will only present the construction that appears most natural to us.

Definition 18 (Product Register)**.** An n-bit Product Register is constructed using a primitive polynomial, P, with degree n and an additional polynomial, U, with degree at most $n-1$, which is called the update polynomial. The register has feedback logic which, on each clock cycle, implements field multiplication by U in the $\mathbb{F}_{2^n}$ field defined modulo P. The update for the product register, A, is described algebraically as follows:

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

We do not consider $U = 0$ and $U = 1$ to be potential update polynomials because they are not useful. Field multiplication by 0 would result in clearing the register because 0 is the multiplicative absorbing element, and field multiplication by 1 would never change the state of the register because 1 is the multiplicative identity.

Note 2. Field addition and multiplication are by definition modulo the primitive polynomial P. When the choice of P does not affect the result, we will omit the modulus in this paper to reduce clutter.

Definition 19 (Mersenne Prime)**.** A Mersenne prime is a prime number that can be written in the form $2^n - 1$, where n is a positive integer. Additionally, the exponent n is called a Mersenne exponent.

Definition 20 (Mersenne Product Register)**.** An n-bit Mersenne Product Register (MPR) is a Product Register (PR) with n equal to a Mersenne exponent.

Although Product Registers are defined generally, we are primarily concerned with Mersenne Product Registers (MPRs) because they exhibit useful properties. For an MPR

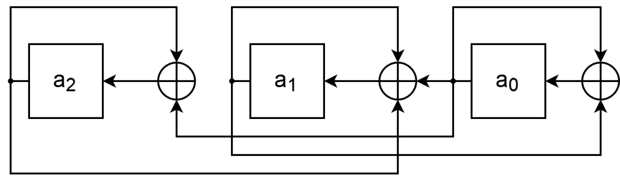


Figure 3: Example Product Register ($U = x^2 + x + 1$, $P = x^3 + x^2 + 1$)

with n bits, the multiplicative group of the corresponding Galois field, $\mathbb{F}_{2^n}^\times$, has no nontrivial subgroups by Lagrange's theorem [Rot95, p.24]. Consequently, $U \neq 0, 1$ implies $\langle U \rangle = \mathbb{F}_{2^n}^\times$. This, in turn, implies that any MPR with $U \neq 0, 1$ will have full period $2^n - 1$. This is illustrated for the 3-bit case in Table 1 which keeps P constant while varying U, to demonstrate the different state cycles a product register will go through for a given update polynomial, all of which are full period.

Table 1: State sequence excluding all zeros for every valid U for 3-bit PR
Primitive Polynomial: $x^3 + x^2 + 1$

Update	$t = 0$	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$t = 5$	$t = 6$
x	001	010	100	101	111	011	110
$x + 1$	001	011	101	010	110	111	100
x^2	001	100	111	110	010	101	011
$x^2 + 1$	001	101	110	100	011	010	111
$x^2 + x$	001	110	011	111	101	100	010
$x^2 + x + 1$	001	111	010	011	100	110	101

Because for an MPR every element of the field (other than $0, 1$) is a primitive element, all $(2^n - 2)/n$ irreducible polynomials are primitive [MvOV97]. Therefore, there are typically more primitive polynomials to choose from when using registers with Mersenne sizes; when combined with the choice of U from $2^n - 2$ elements of $\mathbb{F}_{2^n}$, there are a total of $(2^n - 2)^2/n$ different MPRs for any Mersenne exponent n. This means that MPRs form a varied and easy-to-construct class of building blocks which seem to be a natural choice for chaining together.

4.2 Composite Mersenne Product Registers

A Composite Mersenne Product Register (CMPR) is formed from the repeated concatenation of smaller individual MPRs to form a larger register whose size (i.e., number of bits) is equal to the sum of the sizes of its component MPRs, which are connected via chaining functions.

Definition 21 (Composite Mersenne Product Register)**.** A Composite Mersenne Product Register is a set of MPRs $\{M_1, M_2, \dots, M_k\}$, each with their respective U_i and P_i, and a set of chaining functions $\{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_{k-1}\}$. Each MPR now updates according to Equation 1 of Subsection 3.1 as follows:

$$
\begin{aligned}
M_1[t+1] &= (U_1 \times M_1[t]) \bmod P_1 \\
M_2[t+1] &= (U_2 \times M_2[t]) \oplus \mathcal{C}_1(M_1[t]) \bmod P_2 \\
&\vdots \\
M_k[t+1] &= (U_k \times M_k[t]) \oplus \mathcal{C}_{k-1}(M_1[t], \dots, M_{k-1}[t]) \bmod P_k
\end{aligned}
$$

Definition 22 (Composite Mersenne Product Register with Restricted Chaining)**.** A Composite Mersenne Product Register *with restricted chaining* is a CMPR with the chaining functions $\mathcal{C}_k$ restricted to take input only from the immediately preceding MPR M_k. In this setting, the update equations will be of the form

$$\begin{aligned} M_1[t+1] &= (U_1 \times M_1[t]) \bmod P_1 \\ M_2[t+1] &= (U_2 \times M_2[t]) \oplus \mathcal{C}_1(M_1[t]) \bmod P_2 \\ &\vdots \\ M_k[t+1] &= (U_k \times M_k[t]) \oplus \mathcal{C}_{k-1}(M_{k-1}[t]) \bmod P_k \end{aligned}$$

The motivation for this restriction will be justified in Section 6.3.3.3 (and further discussion will be deferred until then), but we note that all results derived for the general CMPR also apply to this restricted subset of CMPRs defined herein.

Figure 4 gives an example of a CMPR in which the logic shown in red (an XOR gate and an AND gate) acts as the chaining function $\mathcal{C}_1$. This logic connects the MPR shown in Figure 3 to a new, 2-bit MPR. By Theorem 1, we know that this construction has cycle structure $\{21, 7, 3, 1\}$. Specifically, Register A from Theorem 1 is Figure 3 and has two cycles, one of length 1 (for the case where all bits are zero) and a second of length 7. Register B is the right-hand side of Figure 4 and has, from the notation in Theorem 1, $n = 2$ and $m = 3$. Let us start with period 7 from Register A ($p = 7$). The value of $\gcd(m, p)$ is $\gcd(3, 7) = 1$ and the value of $\operatorname{lcm}(m, p)$ is $\operatorname{lcm}(3, 7) = 21$, so the result is that the period 7 cycle from Register A creates two cycles, one of length 21 and the other of length 7. The period 1 (all zeros) cycle from Register A by a similar set of calculations, when chained to Register B, results in a cycle of length 3 and a cycle of length 1. The result is a cycle structure $\{21, 7, 3, 1\}$.

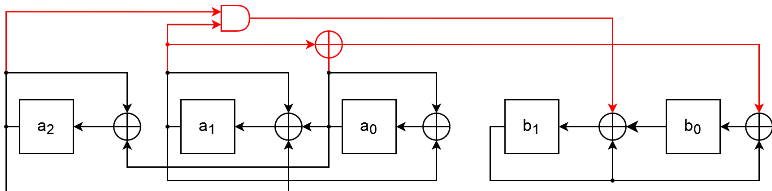


Figure 4: Example of CMPR with 2 and 3-bit MPRs

Theorem 2. *Let R be a CMPR composed of k MPRs, denoted $\{M_1, \ldots, M_k\}$, with sizes $S = \{m_1, \ldots, m_k\}$ where each m_i is a unique Mersenne exponent. Then R has a unique cycle of states for each subset $S \subseteq \{m_1, \ldots, m_k\}$, with that cycle having a length equal to $\prod_{m_i \in S} (2^{m_i} - 1)$.*

Proof. We will prove this by induction on the number of component MPRs, k.

In the base case, consider the first MPR, M_1, with size m_1 and no chaining function. Note that M_1 has one cycle of size 1 and one cycle of size $2^{m_1} - 1$, corresponding to the empty subset and $\{m_1\}$.

We will now prove the inductive step by showing that if the theorem holds for any CMPR with $k - 1$ component MPRs, then it holds for any CMPR with k. Consider the CMPR with one MPR of each size in $S' = \{m_1, \ldots, m_{k-1}\}$ formed by removing the final MPR, M_k, from any CMPR R. This CMPR still satisfies the conditions on MPR sizes, as each m_i is a Mersenne exponent and no two m_i are equal. Consider adding back MPR M_k, using the same chaining function, to recreate R, and the application of Theorem 1.

Consider any subset $S \subseteq \{m_1, \ldots, m_k\}$ and the modified subset $S' = S \cap \{m_1, \ldots, m_{k-1}\}$ formed by ignoring m_k. By the inductive hypothesis, there is a cycle in A with length $p = \prod_{m_i \in S'} (2^{m_i} - 1)$. Because all $m_i \in S'$ are distinct Mersenne primes, this is the prime factorization of the cycle length, and thus $2^{m_k} - 1$ does not divide p. This implies that cases one and two of Theorem 1 are not possible. Therefore, by Theorem 1, R has a cycle of length p (corresponding to the case when $m_k \notin S$) and a cycle of length $p(2^{m_k} - 1)$ (corresponding to the case when $m_k \in S$) proving the theorem also holds for R. This completes the induction. □

Example 3. Consider a 15-bit CMPR composed of three MPRs $\{M_1, M_2, M_3\}$ with sizes $\{3, 5, 7\}$ respectively. Table 2 shows the lengths of all state cycles in this CMPR.

Table 2: Cycles of a CMPR

Subset of MPRs	Cycle Length Formula	Cycle Length
$\{\}$	1	1
$\{M_1\}$	$(2^3 - 1)$	7
$\{M_2\}$	$(2^5 - 1)$	31
$\{M_3\}$	$(2^7 - 1)$	127
$\{M_1, M_2\}$	$(2^3 - 1)(2^5 - 1)$	217
$\{M_1, M_3\}$	$(2^3 - 1)(2^7 - 1)$	889
$\{M_2, M_3\}$	$(2^5 - 1)(2^7 - 1)$	3937
$\{M_1, M_2, M_3\}$	$(2^3 - 1)(2^5 - 1)(2^7 - 1)$	27559
Total:	2^{15}	32768

As can be seen in Table 2, the majority of the CMPR states are contained in the largest cycle regardless of the chaining functions used. This allows a very large degree of freedom in chaining function selection while still guaranteeing the cycle structure of the CMPR permutation. While the existence of short cycles is undesirable, at present, it is unclear how to efficiently identify short cycles, or how they could be used for an attack. Additionally, given a random seed, the odds that the register starts in a short cycle are very low because those cycles contain relatively few of the possible states. We formalize this intuition by analyzing the expected value of cycle length given a starting seed selected uniformly at random.

Definition 23 (Expected Period Ratio)**.** The Expected Period Ratio (EPR) of a CMPR of size n is given by $\frac{\mathbb{E}(X)}{2^n}$, where $\mathbb{E}(X)$ is the expected value of the period when the CMPR is initialized to a random state.

The EPR of any CMPR can be efficiently calculated using the following theorem.

Theorem 3. *Let C be a CMPR composed of k MPRs, denoted $\{M_1, \ldots, M_k\}$, with sizes $S = \{m_1, \ldots, m_k\}$ where each m_i is a unique Mersenne exponent. Let n be the total number of bits in register C. The total number of states in C is then $2^n = \prod_{m_i \in S} 2^{m_i}$.*

Let the initial state of C be chosen uniformly at random, and let the random variable X denote the length of the cycle before this initial state repeats (the period of the register, started from this state). Then,

$$\frac{\mathbb{E}(X)}{2^n} = \prod_{m_i \in S} \left(1 - \frac{(2^{(m_i+1)} - 2)}{2^{2m_i}}\right)$$

Proof. We proceed by induction, as in the proof for Theorem 2. In the base case of a single MPR with size m_1, there are only two cycles, one of length $(2^{m_1}-1)$ which occurs with probability $\frac{(2^{m_1}-1)}{2^{m_1}}$, and one of size 1 which occurs with probability $\frac{1}{2^{m_1}}$. Thus:

$$\mathbb{E}(X) = \frac{(2^{m_1}-1)^2+1}{2^{m_1}} = \frac{2^{2m_1}-2^{m_1+1}+2}{2^{m_1}}$$

In this case, $n = m_1$, thus:

$$\frac{\mathbb{E}(X)}{2^n} = \frac{2^{2m_1}-2^{m_1+1}+2}{2^{2m_1}} = 1 - \frac{(2^{m_1+1}-2)}{2^{2m_1}}$$

We will now prove the inductive step by showing that if the theorem holds for any CMPR with $k-1$ MPRs, then it holds for any CMPR with k MPRs, formed by chaining an additional MPR. Consider the CMPR C' with one MPR of each size in $S' = \{m_1, \ldots, m_{k-1}\}$ formed by removing the final MPR, M_k from C. This CMPR still satisfies the conditions on MPR sizes, as each m_i is a distinct Mersenne exponent. Consider adding back MPR M_k using the same chaining function to recreate C. By Theorem 1, for each possible cycle of length p in C' there is one cycle of length p in C and one cycle of length $p(2^{m_k}-1)$, depending on the initial state of M_k.

Let X' be the expected period of C' and let n' be its size. Because the initial state of M_k is chosen uniformly at random, the expected period stays the same (i.e., $\mathbb{E}(X) = \mathbb{E}(X')$), with only probability $1/2^{m_k}$. With probability $(2^{m_k}-1)/2^{m_k}$, every cycle in C' is multiplied by $2^{m_k}-1$, and by the linearity of expectation, $\mathbb{E}(X) = \mathbb{E}(X')(2^{m_k}-1)$. The inductive step follows from the Law of Total Expectation:

$$\begin{aligned}
\frac{\mathbb{E}(X)}{2^n} &= \left(\frac{\mathbb{E}(X')}{(2^{n'})(2^{m_k})}\right)\left(\frac{1}{2^{m_k}}\right) + \left(\frac{\mathbb{E}(X')(2^{m_k}-1)}{(2^{n'})(2^{m_k})}\right)\left(\frac{2^{m_k}-1}{2^{m_k}}\right) \\
&= \left(\frac{\mathbb{E}(X')}{2^{n'}}\right)\left(\frac{(2^{m_k}-1)^2+1}{2^{2m_k}}\right) \\
&= \left(\prod_{m_i \in S'}\left(1 - \frac{2^{m_i+1}-2}{2^{2m_i}}\right)\right)\left(1 - \frac{(2^{m_k+1}-2)}{2^{2m_k}}\right) \\
&= \prod_{m_i \in S}\left(1 - \frac{(2^{m_i+1}-2)}{2^{2m_i}}\right)
\end{aligned}$$

□

Since $1 - \frac{2^{(n+1)}-2}{2^{2n}} < 1$ for every $n \in \mathbb{N}$, this ratio decreases for each MPR added. However, this sequence can be can shown to converge using the following argument. The product evaluated over the set of all Mersenne exponents $\{2, 3, 5, \ldots\}$ is

$$\prod_{m \in 2,3,5,\ldots}\left(1 - \frac{(2^{m+1}-2)}{2^{2m}}\right) \geq \prod_{m=2}^{\infty}\left(1 - \frac{(2^{m+1}-2)}{2^{2m}}\right) \geq \prod_{m=2}^{\infty}\left(1 - \frac{(2^{m+1})}{2^{2m}}\right) = \prod_{m=1}^{\infty}\left(1 - \frac{1}{2^m}\right)$$

which converges to a constant, and thus the expected period is $O(2^n)$ for any n-bit CMPR. The product from Theorem 3 converges quickly, and we found that the product is ≈ 0.4514 when the set of sizes S consists of all Mersenne primes. This result implies that regardless of the construction sizes or chaining functions chosen, on average, the CMPR construction iterates through at least 45.14% of all bit-vectors that can be stored in a register of that size when initialized to a random seed. Even in the worst configurations, the expected cycle length of a CMPR grows linearly with the maximum period of the register and exponentially with the number of bits. In practice, the EPR is typically much higher than

45.14%. This is because the largest decreases in the expected period are caused by the use of small registers. For example, the use of a 2-bit MPR includes a factor of 0.625 in the product. This single term represents a bigger decrease in the EPR than all other Mersenne exponents combined, which have a product ≈ 0.7223. CMPR constructions that avoid even a few of the smaller registers have a much higher lower bound on their EPR. Table 3 shows lower bounds on the EPR, depending on the smallest MPR used, calculated using Theorem 3.

Table 3: Lower Bound on Expected Period Ratio, by Smallest Component Size

Smallest MPR Size	Approximate Lower Bound on EPR
2	0.4514
3	0.7223
5	0.9246
7	0.9842
13	0.9997

Even though only using Mersenne-sized blocks might seem to be a strong restriction, there are often many different constructions for desired sizes at practical scales. Table 4 shows some example constructions for common register sizes along with their associated expected period and ratio.

Table 4: Example Constructions for Common Sizes

CMPR Size	Mersenne Exponents	Expected Period	Expected Period Ratio
32	19,13	$\approx 4.2939 \times 10^9$	≈ 0.99975
64	61, 3	$\approx 1.4412 \times 10^{19}$	≈ 0.78125
128	61, 31, 19, 17	$\approx 3.4028 \times 10^{38}$	≈ 0.99998
256	127, 61, 31, 17, 13, 7	$\approx 1.1397 \times 10^{77}$	≈ 0.98424

These results demonstrate consistently large expected periods despite the incredible flexibility afforded by the chaining functions.

5 Linear Complexity Analysis

5.1 Preliminaries and Motivation

Definition 24 (Linear Complexity)**.** Let $u = \{u[t]\}$ be a periodic sequence of bits. The linear complexity $\Lambda(u)$ of u is the length of the smallest LFSR that can generate u.

One of the primary ways in which NLFSRs are superior to LFSRs is that LFSRs have an inherent weakness in their low linear complexity. To be secure, we require that CMPRs have sufficiently high linear complexity. However, analyzing the linear complexity of CMPRs is a nontrivial task because of the large differences between designs. Due to the freedom of selecting chaining functions, there is often significant variation between two different CMPRs of the same size. The linear complexity of a CMPR can range from linear in the number of bits (e.g., due to the use of trivial chaining functions which always output 1 or 0) to a significant portion of the full period, depending on the selection of sizes and chaining functions used.

This makes it necessary to algorithmically analyze each CMPR individually based on its chaining functions; it also makes it difficult to provide good bounds on linear complexity for the entire class. In addition, because the linear complexity of a CMPR can grow to be very large, empirical measurements such as the Berlekamp-Massey algorithm become unfeasible for large, reasonably complex CMPRs. For these reasons, we are unable to directly quantify

the linear complexity of large CMPRs. We instead introduce an algorithm that can be used to estimate the linear complexity of a CMPR based on its chaining functions. This allows for designers to assess the linear complexity of their selected MPR sizes and chaining functions, even when the resulting CMPR grows to an otherwise unapproachable size. Section 5 is devoted to presenting this algorithm and the results of the corresponding analysis. The remainder of Subsection 5.1 reviews prior work, defines new notation and explains the high-level approach of the algorithm. In Subsection 5.2 we discuss some of the mathematical approaches and introduce our new notation. In Subsection 5.3 we introduce the algorithm in full. Finally, in Subsection 5.4, we discuss a tweak to the algorithm that handles the lower bound of the estimate.

Definition 25 (Filter Generator)**.** Given an LFSR, L, of size n and a function, $f : \mathbb{F}_{2^n} \to \mathbb{F}_2$, a filter generator outputs the bit sequence $b = \{b[i]\}$ where $b[i] = f(L[i])$. Per [Key76], if d is the algebraic degree of f, then the linear complexity of the filter generator satisfies

$$\Lambda(b) \leq \sum_{i=0}^{d} \binom{n}{i}$$

.

Definition 26 (Combination Generator)**.** Given a set of LFSRs $\{L_1, \ldots, L_k\}$ with corresponding sizes $\{s_1, \ldots, s_k\}$, and a function $f : \mathbb{F}_2^k \to \mathbb{F}_2$, a combination generator outputs the bit sequence b where $b[i] = f(L_1[i], L_2[i], \ldots, L_k[i])$. Additionally, the linear complexity of the combination generator satisfies

$$\Lambda(b) \leq f(\Lambda(s_1), \Lambda(s_2), \ldots, \Lambda(s_n))$$

where the algebraic normal form of f is evaluated over the integers. If the sizes $\{s_1, \ldots, s_k\}$ are all distinct, then this holds with equality [Can11].

Filter generators, combination generators, and compositions of the two have historically served as a tool to build large-period, nonlinear structures from compositions of LFSRs. They are theoretically simple and have been well analyzed in terms of linear complexity. In a CMPR, the chaining functions bear similarities to both filter generators and combination generators. In particular, similar to filter generators, CMPRs allow for more than one bit to be used from each sub-register. On the other hand, similar to combination generators, CMPRs draw from multiple MPRs of different sizes. Despite these similarities, because CMPRs have nested composition with intermediate sequences on which the above bounds are not well-defined, we cannot naively compose the bounds to obtain an upper bound for CMPRs. Instead, the following section will describe a modification to the traditional methods of analysis for filter and combination generators which allows us to obtain tight bounds when analyzing CMPRs.

5.2 Root Counting

Historically, one of the methods used to analyze linear complexity is to bound the number of roots of certain polynomials. Notably, this technique was used by Key in [Key76] to bound the linear complexity of filter generators, and later the technique was extended to bound the linear complexity of combination generators. There exists a large body of research with several good resources such as [Rue86] and [LN94]; this subsection will briefly cover the technique and its application to CMPRs.

Definition 27 (Characteristic Polynomial)**.** For a sequence $\{s[t]\}$ over $\mathbb{F}_2$, if there exists a set of coefficients $c_1, \ldots, c_k$ such that

$$s[t+k] = \sum_{i=1}^{k} c[i]s[t+k-i],$$

then we say $\{s[t]\}$ satisfies a *linear recurrence* of degree k. We define the characteristic polynomial of the sequence to be

$$C(x) = x^k \oplus \sum_{i=0}^{k-1} c_{i+1} x^i$$

For a sequence generated by a full period LFSR or MPR, the characteristic polynomial coincides with the characteristic polynomial of the matrix representing the feedback function.

Property 5. The number of solutions to $C(x) = 0$, which are called *roots*, is equal to the linear complexity of the sequence. These roots will lie in some extension field $\mathbb{F}_{2^n}$. This makes roots useful for determining linear complexity [Key76].

Note 3. Often, authors will abuse notation by multiplying roots in different fields, e.g., multiplying $\alpha \in \mathbb{F}_{2^5}$ and $\beta \in \mathbb{F}_{2^3}$ and obtaining $\alpha\beta \in \mathbb{F}_{2^{15}}$; we will also adopt this abuse of notation. One way of resolving this formally is to view the roots as elements of the algebraic closure of $\mathbb{F}_2$ and understand that when we say that a root is in $\mathbb{F}_{2^n}$, it is a member of the subfield isomorphic to $\mathbb{F}_{2^n}$.

In [Key76], Key also showed that each term of the sequence can be expressed in terms of these roots. Let $\mathbb{F}_{2^n}$ be the splitting field of $C(x)$ (i.e., the smallest extension field which contains all of its roots), then the tth term of the sequence can be expressed as

$$a[t] = \sum_{i=1}^{2^n-1} A_i (\alpha^i)^t \tag{2}$$

where $A_i \in \mathbb{F}_{2^n}$ and α^i ranges over the entire multiplicative group of $\mathbb{F}_{2^n}$ (i.e., α is primitive). We will call this the *root representation* of the sequence.

Definition 28 (Cyclotomic coset)**.** A cyclotomic coset is a set of integers of the form $\{k2^i \mid i \geq 0\}$ for k a positive integer, when reduced modulo $2^n - 1$.

In $\mathbb{F}_{2^n}$, for any polynomial P, $P(x^2) = P(x)^2$. This implies that if $P(\alpha) = 0$ for some root α, then $P(\alpha^2) = 0^2 = 0$. This gives a natural representation of roots of any primitive polynomial as powers of any primitive element α: $\{\alpha^k, \alpha^{2k}, \alpha^{4k}, \ldots\}$ for some k. Thus, when n is prime, the cyclotomic cosets can be used to partition all elements $\alpha^i \in \mathbb{F}_{2^n}$ into sets of roots, each corresponding to one primitive polynomial.

Definition 29 (Coset weight)**.** We define the coset weight of a set of roots $\{\alpha^k, \alpha^{2k}, \alpha^{4k}, \ldots\}$ to be the Hamming weight of any exponent $\{k, 2k, 4k, \ldots\}$ when written in binary.

Note 4. In $\mathbb{F}_{2^n}$, $\alpha^{2^n-1} \equiv 1$, so squaring is equal to a cyclic left shift of the exponent. Therefore, the Hamming weight is well-defined and is the same for all elements of the set.

Example 4. If $\{m_i[t]\}$ is a binary sequence generated by one bit in a full-period LFSR or MPR with n bits, the linear recurrence also only has degree n. Thus, the characteristic polynomial has only one set of conjugate roots, and we have that, using Equation 2, the tth term can be written as

$$m_i[t] = \sum_{i=1}^{n} A_i (\alpha^{k2^i})^t$$

with all other coefficients equal to zero.

If a root is present (i.e., if there is a nonzero coefficient) in the root representation of a sequence, then its minimal polynomial divides the characteristic polynomial. Thus, if any member of a coset is present, the entire coset contributes to the linear complexity of the sequence. Key observed that term-wise operations on these sequences can be represented using their root representation. For example, termwise XOR (addition) of sequences can be represented as follows:

$$c[t] = a[t] \oplus b[t] = \sum_{i=1}^{2^n-1} A_i(\alpha^i)^t \oplus \sum_{i=1}^{2^n-1} B_i(\alpha^i)^t = \sum_{i=1}^{2^n-1} (A_i \oplus B_i)(\alpha^i)^t$$

When sequences are composed of roots from the same field (as is always the case when they are two sequences generated by the same LFSR), the exponents can be combined:

$$c[t] = a[t]\ b[t] = \left(\sum_{i=1}^{2^n-1} A_i(\alpha^i)^t\right)\left(\sum_{j=1}^{2^n-1} B_j(\alpha^j)^t\right) = \sum_{i=1}^{2^n-1}\sum_{j=1}^{2^n-1} A_i B_j (\alpha^{i+j})^t$$

After taking the termwise product, new exponents (and new cosets) may be generated, but none of these cosets have a weight greater than the sum of the weights of the two sequences added [Key76]. Key uses this concept of weight to tighten the upper bound on linear complexity for a filter and arrive at an upper bound for the linear complexity of a filter generator. If the degree of the filter is d, then the product of d terms, each with weight 1, must have weight less than or equal to d. This means that the linear complexity is upper bounded by the number of binary strings of length n with Hamming weight less than or equal to d. This leads to the well-known inequality cited earlier:

$$\Lambda(s) \leq \sum_{i=1}^{d} \binom{n}{i}$$

When the two sequences instead feature roots exclusively from two fields with coprime exponents n and m we instead have

$$c[t] = a[t]\ b[t] = \left(\sum_{i=1}^{2^n-1} A_i(\alpha^i)^t\right)\left(\sum_{i=1}^{2^m-1} B_i(\beta^i)^t\right) = \sum_{i=1}^{2^n-1}\sum_{j=1}^{2^m-1} A_i B_j (\alpha^i \beta^j)^t$$

This situation is also analyzed in a simple case by Key.

Lemma 3. *If $\alpha \in \mathbb{F}_{2^n}$ and $\beta \in \mathbb{F}_{2^m}$ are roots of irreducible polynomials with degree n and m respectively, and n and m are coprime, then their product $\alpha\beta$ is in $\mathbb{F}_{2^{nm}}$ and is not contained in any subfield. Moreover, the product of any conjugates of α and β is conjugate to $\alpha\beta$ as the root of an irreducible polynomial with degree nm [Key76].*

[Rue86] contains several generalizations of the result which allow the analysis of linear complexity for combination generators with LFSR sizes that are not coprime. For CMPRs, every MPR has a prime number of bits, and thus Lemma 3 is sufficient.

Our primary contribution in this area is introducing new notation and a method of computing these bounds in the case where each size is prime and is used only once. In this restricted case, which includes CMPRs, this new analysis unifies and generalizes several of the previous results. This in turn allows us to analyze the linear complexity of chaining functions in situations where we were unable to apply the known bounds previously. The assumption that field sizes will be prime and unique will be in place throughout the rest of this section.

Definition 30 (Coset class)**.** We define the coset class with exponent e and weight w to be the set of all roots in $\mathbb{F}_{2^e}$ with an exponent contained in a cyclotomic coset with weight at most w. To improve the readability of equations, we will denote them similar to a vector (either horizontal or vertical) with a dot in the middle to distinguish them from traditional vectors:

$$\text{Coset class with exponent } e \text{ and weight } w = \begin{bmatrix} e \\ \cdot \\ w \end{bmatrix} = \langle e \cdot w \rangle$$

In other words, if we fix $\alpha \in \mathbb{F}_{2^e}$ primitive, then $\beta \in \langle e \cdot w \rangle$ if $\beta = \alpha^k$ with $k \leq w$.

Definition 31 (Root expression)**.** To any periodic sequence S over $\mathbb{F}_2$ with roots only in fields with prime exponents, we can assign a formal algebraic expression of coset classes called its root expression. The root expression describes the set of roots that might have nonzero coefficients in the root representation of the sequence. We will also let the function $\mathcal{E}$ denote the mapping from sequences to their corresponding root expressions.

Property 6. By definition, no roots can contribute to the linear complexity of a sequence unless they are included in the root expression, and thus we have the following:

$$\Lambda(S) \leq |\mathcal{E}(S)|$$

This format helps maintain information about the different compositions of roots with varying extension fields and weights, which allows the results to be reused for analysis of future combiners and filters. These expressions provide a useful approximation to the true sets of roots. They do not fully describe which roots are present, but they do give us large-scale groups of roots that behave predictably with high probability. The expressions can be used to model the behavior of the underlying roots and provide tighter bounds. The AND (product) of two sequences is associated with the formal multiplication of their root expressions, and the XOR is associated with formal addition.

Example 5. Let sequence $\{a[t]\}$ have root expression $\mathcal{E}(\{a[t]\}) = \langle 13 \cdot 3 \rangle + \langle 3 \cdot 1 \rangle$ and sequence $\{b[t]\}$ have root expression $\mathcal{E}(\{b[t]\}) = \langle 7 \cdot 3 \rangle \langle 5 \cdot 1 \rangle + \langle 7 \cdot 6 \rangle$. Then the sequence formed by their termwise AND, $\{c[t]\} = \{a[t]\ b[t]\}$, has the following root expression:

$$\left(\begin{bmatrix} 13 \\ \cdot \\ 3 \end{bmatrix} + \begin{bmatrix} 3 \\ \cdot \\ 1 \end{bmatrix} \right) \left(\begin{bmatrix} 7 \\ \cdot \\ 3 \end{bmatrix} \begin{bmatrix} 5 \\ \cdot \\ 1 \end{bmatrix} + \begin{bmatrix} 7 \\ \cdot \\ 6 \end{bmatrix} \right) = \begin{bmatrix} 13 \\ \cdot \\ 3 \end{bmatrix} \begin{bmatrix} 7 \\ \cdot \\ 3 \end{bmatrix} \begin{bmatrix} 5 \\ \cdot \\ 1 \end{bmatrix} + \begin{bmatrix} 13 \\ \cdot \\ 3 \end{bmatrix} \begin{bmatrix} 7 \\ \cdot \\ 6 \end{bmatrix} + \begin{bmatrix} 3 \\ \cdot \\ 1 \end{bmatrix} \begin{bmatrix} 7 \\ \cdot \\ 3 \end{bmatrix} \begin{bmatrix} 5 \\ \cdot \\ 1 \end{bmatrix} + \begin{bmatrix} 3 \\ \cdot \\ 1 \end{bmatrix} \begin{bmatrix} 7 \\ \cdot \\ 6 \end{bmatrix}$$

The Algebraic Normal Form (ANF) for any Boolean filter function notation can be used to construct the appropriate root expression for the sequence formed when that filter is used on an LFSR or MPR. We will propose a set of rules that describe how to manipulate these root expressions and how to use them to derive an upper bound on linear complexity.

Proposition 1. *For any coset class, we have* $|\langle e \cdot w \rangle| = \sum_{i=1}^{w} \binom{e}{i}$

Proof. This argument can be found in [Key76]; each exponent can be written as an e-bit binary string, of which there are $\binom{e}{i}$ roots with with exponents with coset weight i. Summing over all weights less than w gives the number of roots in the coset class. □

Proposition 2. *Let* $\{\langle e \cdot w_1 \rangle, \langle e \cdot w_2 \rangle, \ldots, \langle e \cdot w_k \rangle\}$ *be a set of coset classes with common exponent* e*. Then we have the following:*

$$\prod_{i=1}^{k} \langle e_i \cdot w_i \rangle = \left\langle e \cdot \sum_{i=1}^{k} w_i \right\rangle$$

Proof. This proof can also be found in more detail in [Key76] and is the basis of the filter generator bound, expressed in our notation. □

Proposition 3. *Let $\{\langle e_1 \cdot w_1\rangle, \langle e_2 \cdot w_2\rangle, \ldots, \langle e_k \cdot w_k\rangle\}$ be a set of coset classes such that $e_i \neq e_j$ for any i, j. Then we have the following:*

$$\left|\prod_{i=1}^{k} \langle e_i \cdot w_i\rangle\right| = \prod_{i=1}^{k} |\langle e_i \cdot w_i\rangle|$$

Proof. This follows from induction on i, with the inductive step provided by Lemma 3. □

Proposition 4. *For any two root expressions E_1 and E_2:*

$$|E_1 + E_2| \leq |E_1 \cup E_2| = |E_1| + |E_2| - |E_1 \cap E_2|$$

Proof. Consider the termwise addition of expressions in the form of Equation 2:

$$c[t] = a[t] \oplus b[t] = \sum_{i=1}^{2^n-1} A_i(\alpha^i)^t \oplus \sum_{i=1}^{2^n-1} B_i(\alpha^i)^t = \sum_{i=1}^{2^n-1} (A_i \oplus B_i)(\alpha^i)^t$$

For any root α^i, the coefficient $(A_i \oplus B_i)$ is nonzero if at least one of A_i or B_i is nonzero and $A_i \neq B_i$. Thus, a root can only be present in $\mathcal{E}(\{c[t]\})$ if it already existed in one of the original sequences. This implies that for any two expressions, $|E_1 + E_2| \leq |E_1 \cup E_2|$. The second equality follows from the principle of inclusion-exclusion. □

Proposition 5. *Let E_1 and E_2 be two root expressions, each of which is a single product of coset classes. If they have different sets of exponents, then $E_1 \cap E_2$ is empty. Otherwise, if two products of coset classes have the same set of exponents $\{e_1, \ldots, e_k\}$, then*

$$E_1 \cap E_2 = \left(\prod_{i=1}^{k} \langle e_i \cdot w_{1,i}\rangle\right) \cap \left(\prod_{i=1}^{k} \langle e_i \cdot w_{2,i}\rangle\right) = \left(\prod_{i=1}^{k} \langle e_i \cdot \min(w_{1,i}, w_{2,i})\rangle\right)$$

Example 6.

$$(\langle 13 \cdot 3\rangle \langle 7 \cdot 6\rangle \langle 5 \cdot 4\rangle) \cap (\langle 13 \cdot 7\rangle \langle 7 \cdot 2\rangle \langle 5 \cdot 1\rangle) = \langle 13 \cdot 3\rangle \langle 7 \cdot 2\rangle \langle 5 \cdot 1\rangle$$
$$(\langle 19 \cdot 3\rangle \langle 7 \cdot 6\rangle \langle 5 \cdot 4\rangle) \cap (\langle 13 \cdot 7\rangle \langle 7 \cdot 2\rangle \langle 5 \cdot 1\rangle) = \emptyset$$

The difference between the two equations is that in the second equation, there is a coset $\langle 19 \cdot 3\rangle$ in the first root expression, with an exponent that is not present anywhere in the second root expression. Because the two root expressions do not have the same exact exponents, they have no intersection.

Proof. Because any MPR uses a prime number of bits, Lemma 3 implies that the roots represented by the term

$$\left(\prod_{i=1}^{k} \langle e_i \cdot w_i\rangle\right)$$

are contained in $\mathbb{F}_{2^n}$ where $n = \prod_{i=1}^{k} e_i$. Furthermore, the roots are not contained in any subfield. Thus, if the product terms do not have the same parameters e_i, then their roots reside in separate subfields and thus cannot overlap. We will proceed assuming that the two root expressions have the same exponents and thus nonzero intersection, and we further note that the weights introduced by Key give an ordering of the sets of roots: $\langle e_i \cdot w_1\rangle \subseteq \langle e \cdot w_2\rangle$ if and only if $w_1 \leq w_2$ [Key76]. Thus, for each parameter e_i shared by the two root expressions, $\langle e_i \cdot \min(w_{1,i}, w_{2,i})\rangle$, is the intersection of $\langle e_i \cdot w_{1,i}\rangle$ and $\langle e_i \cdot w_{2,i}\rangle$. □

These rules, in the case of LFSRs with distinct prime sizes, can be treated as a generalization and unification of the bounds for filter generators and combination generators. When allowing a nonlinear filter using multiple bits, but only from one LFSR, Propositions 1 and 2 give the filter generator bound directly. The combination generator bound is similarly derivable when allowing multiple LFSRs but restricting to only using the output sequence of each. Propositions 4 and 5 imply there is no nontrivial intersection of any of the product-terms, while Propositions 2 and 3 show that each product-term has root size equal to the product of the sizes of LFSRs used.

The primary disadvantage of this rule system is that it requires the LFSR sizes to be prime and distinct. This limitation might be overcome with more careful casework, but the current formulation is sufficient for the analysis of CMPRs. The primary advantage of this system is that it can handle complicated nonlinear combinations of inputs from multiple registers. Additionally, root expressions provide a compact intermediate representation, which can allow for easier algebraic manipulation, as shown in the following example.

Example 7. Let $\{a_1[t]\}, \{a_2[t]\}$ and $\{a_3[t]\}$ be three different sequences, each with root expression $\langle 7 \cdot 2 \rangle + \langle 5 \cdot 1 \rangle$. Such a sequence could, for example, be the output of a degree-two filter on a 7-bit LFSR XORed with the output of a 5-bit LFSR. Let $s = \{s[t]\}$ be the sequence formed by the termwise AND of all three sequences. In this example, we will use the root expression method to upper bound the linear complexity of $\{s[t]\}$.

First, we distribute the product to simplify the root expression for $\{s[t]\}$, which resembles ANF notation.

$$\begin{aligned}\Lambda(s) &\leq \left| \left(\begin{bmatrix} 7 \\ \cdot \\ 2 \end{bmatrix} + \begin{bmatrix} 5 \\ \cdot \\ 1 \end{bmatrix} \right)^3 \right| \\ &\leq \left| \left(\begin{bmatrix} 7 \\ \cdot \\ 2 \end{bmatrix} \begin{bmatrix} 7 \\ \cdot \\ 2 \end{bmatrix} \begin{bmatrix} 7 \\ \cdot \\ 2 \end{bmatrix} + \begin{bmatrix} 7 \\ \cdot \\ 2 \end{bmatrix} \begin{bmatrix} 7 \\ \cdot \\ 2 \end{bmatrix} \begin{bmatrix} 5 \\ \cdot \\ 1 \end{bmatrix} + \begin{bmatrix} 7 \\ \cdot \\ 2 \end{bmatrix} \begin{bmatrix} 5 \\ \cdot \\ 1 \end{bmatrix} \begin{bmatrix} 5 \\ \cdot \\ 1 \end{bmatrix} + \begin{bmatrix} 5 \\ \cdot \\ 1 \end{bmatrix} \begin{bmatrix} 5 \\ \cdot \\ 1 \end{bmatrix} \begin{bmatrix} 5 \\ \cdot \\ 1 \end{bmatrix} \right) \right|\end{aligned}$$

Using Proposition 3, we combine terms with roots in the same field:

$$\Lambda(s) \leq \left| \left(\begin{bmatrix} 7 \\ \cdot \\ 6 \end{bmatrix} + \begin{bmatrix} 7 \\ \cdot \\ 4 \end{bmatrix} \begin{bmatrix} 5 \\ \cdot \\ 1 \end{bmatrix} + \begin{bmatrix} 7 \\ \cdot \\ 2 \end{bmatrix} \begin{bmatrix} 5 \\ \cdot \\ 2 \end{bmatrix} + \begin{bmatrix} 5 \\ \cdot \\ 3 \end{bmatrix} \right) \right|$$

Using Proposition 4, we split the expression into an evaluation of terms.

$$\Lambda(s) \leq \left| \begin{bmatrix} 7 \\ \cdot \\ 6 \end{bmatrix} \right| + \left| \begin{bmatrix} 7 \\ \cdot \\ 4 \end{bmatrix} \begin{bmatrix} 5 \\ \cdot \\ 1 \end{bmatrix} \right| + \left| \begin{bmatrix} 7 \\ \cdot \\ 2 \end{bmatrix} \begin{bmatrix} 5 \\ \cdot \\ 2 \end{bmatrix} \right| - \left| \left(\begin{bmatrix} 7 \\ \cdot \\ 4 \end{bmatrix} \begin{bmatrix} 5 \\ \cdot \\ 1 \end{bmatrix} \right) \cap \left(\begin{bmatrix} 7 \\ \cdot \\ 2 \end{bmatrix} \begin{bmatrix} 5 \\ \cdot \\ 2 \end{bmatrix} \right) \right| + \left| \begin{bmatrix} 5 \\ \cdot \\ 3 \end{bmatrix} \right|$$

Using Proposition 5, we substitute $|\langle 7 \cdot 2 \rangle \langle 5 \cdot 1 \rangle)|$ in place of $|(\langle 7 \cdot 4 \rangle \langle 5 \cdot 1 \rangle) \cap (\langle 7 \cdot 2 \rangle \langle 5 \cdot 2 \rangle)|$

$$\Lambda(s) \leq \left| \begin{bmatrix} 7 \\ \cdot \\ 6 \end{bmatrix} \right| + \left| \begin{bmatrix} 7 \\ \cdot \\ 4 \end{bmatrix} \begin{bmatrix} 5 \\ \cdot \\ 1 \end{bmatrix} \right| + \left| \begin{bmatrix} 7 \\ \cdot \\ 2 \end{bmatrix} \begin{bmatrix} 5 \\ \cdot \\ 2 \end{bmatrix} \right| - \left| \begin{bmatrix} 7 \\ \cdot \\ 2 \end{bmatrix} \begin{bmatrix} 5 \\ \cdot \\ 1 \end{bmatrix} \right| + \left| \begin{bmatrix} 5 \\ \cdot \\ 3 \end{bmatrix} \right|$$

Using Propositions 1 and 2, we separate each term into the product of its different coset classes and evaluate:

$$\begin{aligned}\Lambda(s) &\leq \sum_{i=1}^{6} \binom{7}{i} + \sum_{i=1}^{4} \binom{7}{i} \sum_{i=1}^{1} \binom{5}{i} + \sum_{i=1}^{2} \binom{7}{i} \sum_{i=1}^{2} \binom{5}{i} - \sum_{i=1}^{2} \binom{7}{i} \sum_{i=1}^{1} \binom{5}{i} + \sum_{i=1}^{3} \binom{5}{i} \\ &\leq 126 + (98 \cdot 5) + (28 \cdot 15) - (28 \cdot 5) + 25 \\ &\leq 921\end{aligned}$$

Thus, we know that the linear complexity of s is at most 921.

The number of roots represented by each coset class can become quite large. One of the motivations for this approach is that working with these expressions is computationally easier than trying to model individual roots. Consider, for example, a 127-bit MPR: there are $2^{127} - 2$ primitive roots in the multiplicative group, but only 127 different coset classes to manipulate.

5.3 Root Propagation through an MPR

In total, we have that coset classes behave predictably as they propagate through chaining functions, and we will show in this subsection that the representation propagates efficiently through registers as well. However, because root expressions are only approximations, their efficiency comes at the cost of imperfect information.

Definition 32 (Degeneracy)**.** When a set of roots is included in a root expression but does not contribute to the linear complexity of the corresponding sequence, we say that the set of roots has degenerated. A degeneracy is evident when the linear complexity of a system is lower than the upper bound predicted by root counting.

Although root expressions only yield an upper bound, the linear complexity is typically very close to this bound as observed by [Key76] and [Rue86]. We will further discuss statistical models for degeneracy in Subsection 5.4. In this subsection, we will examine another of the major advantages of using root expressions. Because each periodic sequence has a specific root expression, regardless of the starting point of that sequence, the root expression is the same. Therefore, root expressions are invariant to shifts of the sequence in time. Moreover, because a root expression is unchanged by an XOR with itself (by Property 4), root expressions are invariant to both phase shifts and XORs with phase shifts of the same sequence. We will show that this property means that the representation can be propagated through an MPR easily.

There are many different ways to analyze linear systems (e.g., LFSRs and MPRs). The following approach uses formal power series with coefficients drawn from $\mathbb{F}_2$ and is explored in more detail in resources such as [Rue86] and [LN94]. In this paper, we will use the notation of a Z-transform (i.e., a formal power series in the variable z^{-1}). Following this path leads to the following result.

Theorem 4. *Let $\mathcal{C}(A[t])$ be any periodic sequence of n-bit vectors in $\mathbb{F}_2^n$, and let its Z-transform be denoted $C(z)$. Let B be an n-bit linear feedback register, with n prime and its characteristic polynomial primitive, and which updates according to Equation 1, with $\mathcal{C}(A[t])$ as input on each cycle. Then the Z-transform of $B[t]$ is*

$$B(z) = (zI \oplus U)^{-1}(C(z) \oplus zB[0]) = \frac{1}{\chi_U(z)}\text{Adj}(zI \oplus U)(C(z) \oplus zB[0]))$$

where $\chi_U(z)$ is the characteristic polynomial of U, the matrix by which B updates.

Proof. We begin by writing the update equation $B[t+1] = UB[t] \oplus \mathcal{C}(A[t])$. Then taking the Z-transform of each side, we obtain

$$z(B(z) \oplus B[0]) = UB(z) \oplus C(z)$$

After collecting the $B(z)$ terms to one side and factoring, we find that

$$(zI \oplus U)B(z) = C(z) \oplus zB[0]$$

From here, we can write

$$\begin{aligned} B(z) &= (zI \oplus U)^{-1}(C(z) \oplus zB[0]) \\ &= \frac{1}{\det(zI \oplus U)} \mathrm{Adj}(zI \oplus U)(C(z) \oplus zB[0]) \\ &= \frac{1}{\chi_U(z)} \mathrm{Adj}(zI \oplus U)(C(z) \oplus zB[0]) \end{aligned}$$

where we use the formula $M^{-1} = (\det M)^{-1}\mathrm{Adj}(M)$ for the inverse of a matrix. □

Example 8. We will consider the example from Figure 4 in Subsection 4.2. Note that for this example, we can explicitly write out the update equation as follows:

$$B[t+1] = \begin{bmatrix} b_0[t+1] \\ b_1[t+1] \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} b_0[t] \\ b_1[t] \end{bmatrix} \oplus \mathcal{C}(A[t])$$

Although it is not practical in general to fully compute $C(z)$, we will do so for this example using the Berlekamp-Massey algorithm. Consider the sequence $\mathcal{C}_0(A[t]) = a_1[t] \oplus a_0[t]$; one period of this sequence is given by $(1, 0, 0, 0, 1, 0, 0)$. From the Berlekamp-Massey algorithm, we can determine

$$C_0(z) = \frac{N(z)}{1 \oplus z^{-1} \oplus z^{-3}}$$

where the numerator $N(z)$ is dependent on the initial values of the sequence. Rearranging, we can recover

$$N(z) = C_0(z)(1 \oplus z^{-1} \oplus z^{-3}) = z^{-1} \oplus z^{-2}$$

Next we substitute to obtain

$$C_0(z) = \frac{z^{-1} \oplus z^{-2}}{1 \oplus z^{-1} \oplus z^{-3}} = \frac{z^2 \oplus z^1}{z^3 \oplus z^2 \oplus 1}$$

We can repeat this process to obtain the slightly more complex

$$C_1(z) = \frac{1 \oplus z^{-1} \oplus z^{-2} \oplus z^{-3}}{(1 \oplus z^{-1} \oplus z^{-3})(1 \oplus z^{-2} \oplus z^{-3})} = \frac{z^6 \oplus z^5 \oplus z^4 \oplus z^3}{(z^3 \oplus z^2 \oplus 1)(z^3 \oplus z^1 \oplus 1)}$$

This is more complex because this bit, C_1, is nonlinear (C_0 is linear). From the denominator, this sequence has linear complexity 6. This also could have been noted by observing that the root expression for this sequence is $\langle 3 \cdot 2 \rangle$, which contains 6 roots. Because we know U, we can solve for the resolvent as follows:

$$U = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \implies zI \oplus U = \begin{bmatrix} z & 1 \\ 1 & z \oplus 1 \end{bmatrix} \implies (zI \oplus U)^{-1} = \frac{1}{z^2 \oplus z \oplus 1} \begin{bmatrix} z \oplus 1 & 1 \\ 1 & z \end{bmatrix}$$

In total,

$$
\begin{aligned}
B(z) = \begin{bmatrix} b_0(z) \\ b_1(z) \end{bmatrix} &= \frac{1}{z^2 \oplus z \oplus 1} \begin{bmatrix} z \oplus 1 & 1 \\ & \\ 1 & z \end{bmatrix} \left(\begin{bmatrix} \dfrac{z^2 \oplus z^1}{z^3 \oplus z^2 \oplus 1} \\ \dfrac{z^6 \oplus z^5 \oplus z^4 \oplus z^3}{(z^3 \oplus z^2 \oplus 1)(z^3 \oplus z^1 \oplus 1)} \end{bmatrix} \oplus \begin{bmatrix} z \\ \\ z \end{bmatrix} \right) \\
&= \begin{bmatrix} \dfrac{z^8 \oplus z^7 \oplus z^6 \oplus z^3 \oplus z}{(z^2 \oplus z \oplus 1)(z^3 \oplus z^2 \oplus 1)(z^3 \oplus z^1 \oplus 1)} \\ \dfrac{z^5 \oplus z^3}{(z^2 \oplus z \oplus 1)(z^3 \oplus z^1 \oplus 1)} \end{bmatrix} \\
&= \begin{bmatrix} \dfrac{1 \oplus z^{-1} \oplus z^{-2} \oplus z^5 \oplus z^{-7}}{(1 \oplus z^{-1} \oplus z^{-2})(1 \oplus z^{-1} \oplus z^{-3})(1 \oplus z^{-2} \oplus z^{-3})} \\ \dfrac{1 \oplus z^{-2}}{(1 \oplus z^{-1} \oplus z^{-2})(1 \oplus z^{-2} \oplus z^{-3})} \end{bmatrix}
\end{aligned}
$$

These can be converted back into sequences using a division algorithm to find the initial values and then using the recurrence to extend it. However, an easier forward check is to compute $B[t]$ by running the shown register, computing its Z-transform as described earlier, and verifying that this is indeed the correct sequence.

For most MPR sizes, the matrices $\mathrm{Adj}(zI \oplus U)$ are feasible, if slow, to calculate; however, the Z-transform of the inputs, $C(z)$, becomes infeasible to compute, as the degrees of the polynomials involved are proportional to the linear complexity of the sequences involved. However, if the goal is only to estimate linear complexity, we do not need to know the exact Z-transform of the sequence, only its root expression. The adjugate matrix consists of polynomials in z, which means any output signal can be interpreted as the XOR of several shifts applied the inputs. It is worth noting that the adjugate matrices tend to be dense; none of the matrices we tested had any entries equal to zero, although we do not know if this always holds true. The adjugate matrices being dense implies that this results in a strong mixing; the root expression for every bit in the MPR over time is equal to the sum of many different delays of all root expressions coming from the chaining. Because root expressions are invariant to shifts, after matrix multiplication by $\mathrm{Adj}(zI \oplus U)$, this means that each output root expression is equal to the sum of the root expressions of the all the inputs. There is also an additional primitive polynomial in the denominator, visible in the $1/\chi_U(z)$ term, derived from the characteristic polynomial of the current MPR. This term contributes the roots $\langle m \cdot 1 \rangle$. This makes it possible to easily calculate the root expression of the output sequences using the root expressions of the chaining functions.

Example 9. We will repeat the analysis in Example 8, using root expressions instead. By the rules established in Subsection 5.2, we have that $\mathcal{E}(C_0) = \langle 3 \cdot 1 \rangle$ and $\mathcal{E}(C_1) = \langle 3 \cdot 2 \rangle$. The sequence corresponding to $1/\chi(z)$ has the same characteristic polynomial as U (which has characteristic polynomial $\chi_U(z)$), by definition, and thus has root expression $\langle 2 \cdot 1 \rangle$. In total we have

$$\mathcal{E}(b_0[t]) = \mathcal{E}(b_1[t]) = \langle 3 \cdot 2 \rangle \langle 2 \cdot 1 \rangle$$

which places an upper bound on the linear complexity at 8, which we can see in the example is achieved. Note that in this case, we could not detect the degeneration of $(1 \oplus z^{-1} \oplus z^{-3})$ from the denominator of $C_1(z)$, also demonstrating a degeneracy we were unable to detect.

The estimation algorithm is given in Algorithm 1. It can be summarized informally as follows. First, we initialize an associated root expression for each bit in the first MPR. Next, using the algebraic manipulations given in Propositions 1 through 5, we use the ANF of the chaining function to obtain the root expressions of the output of each chaining function (this process is captured in line 10 of the algorithm). Using Theorem 4, we propagate these expressions to each bit of the next MPR, adding in a new coset class

Algorithm 1 CMPR Root Expression Algorithm

input: A CMPR composed of q MPRs $\{M_1, \ldots, M_q\}$, with distinct sizes $\{s_1, \ldots, s_q\}$. $\mathcal{C}_1 \ldots \mathcal{C}_{q-1}$ denotes the chaining functions of the CMPR.
output: An upper bound on the linear complexity of the output stream, Λ

```
1:  procedure UpperBound
2:      RootExpressions is a table containing the root expression for each bit.
3:      Evaluate(R) determines the size of root expression R
4:      bits(M_i) returns the bits associated with M_i
5:      for b ∈bits(M_1) do
6:          RootExpressions [b] ← ⟨s_1 · 1⟩                    ▷ Initialize table
7:      end for
8:      for i ← 2 . . . q do
9:          composition ← C_{i−1}(RootExpressions)             ▷ Combine table values
10:         for b ∈bits(M_i) do
11:             RootExpressions [b] ← (composition + ⟨s_i · 1⟩)    ▷ Table update
12:         end for
13:     end for
14:     Λ ← Evaluate(RootExpressions[0])
15: end procedure
```

and updating our table, as shown in line 11. We then repeat this process inductively to calculate the root expressions for any bit in the CMPR chain. Finally, after all root expressions have been generated, we evaluate the root expression of the output stream to upper bound linear complexity, again using Propositions 1 through 5. This process allows for more efficient analysis of CMPRs with large linear complexity.

Example 10. We will analyze the approximate linear complexity of a simple example by hand to demonstrate the process. Let C be a 15-bit CMPR made from MPRs with sizes $\{7, 5, 3\}$, depicted in Figure 5.

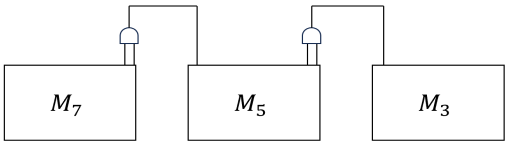


Figure 5: Example 15-bit CMPR

Each chaining function is an AND from the two least significant bits of one MPR to the most significant bit of the next. The ANF is as follows:

$$
\begin{aligned}
c_{14}[t+1] &= c_{13}[t] \oplus c_{14}[t] \\
c_{13}[t+1] &= c_{12}[t] \\
c_{12}[t+1] &= c_{11}[t] \oplus c_{14}[t] \\
c_{11}[t+1] &= c_{10}[t] \\
c_{10}[t+1] &= c_{9}[t] \\
c_{9}[t+1] &= c_{8}[t] \oplus c_{14}[t] \\
c_{8}[t+1] &= c_{14}[t]
\end{aligned}
$$

$$
\begin{aligned}
c_{7}[t+1] &= c_{6}[t] \oplus c_{8}[t]c_{9}[t] \\
c_{6}[t+1] &= c_{5}[t] \oplus c_{7}[t] \\
c_{5}[t+1] &= c_{4}[t] \\
c_{4}[t+1] &= c_{3}[t] \\
c_{3}[t+1] &= c_{7}[t]
\end{aligned}
$$

$$
\begin{aligned}
c_{2}[t+1] &= c_{1}[t] \oplus c_{2}[t] \oplus c_{3}[t]c_{4}[t] \\
c_{1}[t+1] &= c_{0}[t] \\
c_{0}[t+1] &= c_{2}[t]
\end{aligned}
$$

We start by noting that the sequence generated by each bit c_8 through c_{14} has root expression $\langle 7 \cdot 1 \rangle$, as this is an ordinary MPR of size 7. The output of the first chaining function, which consists of the AND of c_8 and c_9, correspondingly has root expression

$$(\langle 7 \cdot 1 \rangle)^2 = \langle 7 \cdot 2 \rangle$$

By Theorem 4 we can conclude that the sequence generated by each bit c_3 through c_7 has root expression $\langle 7 \cdot 2 \rangle \oplus \langle 5 \cdot 1 \rangle$. The next chaining function is the AND of two of these output sequences, c_3 and c_4, and thus has root expression

$$\left(\begin{bmatrix} 7 \\ \cdot \\ 2 \end{bmatrix} \oplus \begin{bmatrix} 5 \\ \cdot \\ 1 \end{bmatrix} \right)^2 = \left(\begin{bmatrix} 7 \\ \cdot \\ 4 \end{bmatrix} \oplus \begin{bmatrix} 7 \\ \cdot \\ 2 \end{bmatrix} \begin{bmatrix} 5 \\ \cdot \\ 1 \end{bmatrix} \oplus \begin{bmatrix} 5 \\ \cdot \\ 2 \end{bmatrix} \right)$$

Therefore, c_0 through c_2 has root expression $(\langle 7 \cdot 4 \rangle \oplus \langle 7 \cdot 2 \rangle \langle 5 \cdot 1 \rangle \oplus \langle 5 \cdot 2 \rangle \oplus \langle 3 \cdot 1 \rangle)$. Because our output sequence is taken from bit zero, we can upper bound the linear complexity by evaluating the size of this root expression. This yields the estimate $98+(28)(5)+15+3 = 256$. The actual linear complexity of this CMPR, starting from any initial state where none of the MPRs are locked in the zero state, measured using the Berlekamp-Massey algorithm [Mas69] is 253, which amounts to a degeneration of one root set from the 3-bit MPR.

We also note that Algorithm 1 is polynomial with respect to the number of bits of the CMPR and exponential with respect to the number of MPRs used to construct the CMPR. To demonstrate this, we will compute a pessimistic upper bound on the time complexity of the algorithm. Let n denote the number of bits in the CMPR, k be the number of constituent MPRs, and $S = \{s_0, \ldots, s_k\}$ be the set of Mersenne exponents. Any root expression will have at most as many terms as the products of the elements in S, excluding the largest Mersenne exponent. Thus, the maximum number of terms in a root expression is given by

$$R \leq \frac{\Pi_{s_i \in S} s_i}{max(S)}$$

For each addition of root expressions, we must loop through each of their terms in sequence, giving an intermediate list of size $O(R)$. To compute the product of the root expressions, we must loop over every pair of terms, giving an intermediate list of size $O(R^2)$. Producing each term in the intermediate list has a cost of at most $O(k)$. Thus, producing the intermediate lists for addition and multiplication has complexity $O(kR)$ and $O(kn^2)$, respectively.

This intermediate list is then pruned by incrementally building a list of maximal elements. At any given point, the maximal list is less than R, and we must loop through the maximal list once for each element we add from the intermediate list (to check if it is maximal). Similar to the cost of building the intermediate terms, the cost of each comparison is $O(k)$. In total, this gives a cost of at most $O(kR^3)$ operations per multiplication and $O(kR^2)$ operations per addition. This dominates the cost of the addition and multiplication, so overall, the complexities for addition and multiplication are $O(kR^2)$ and $O(kR^3)$, respectively.

There is a pruning step in which we calculate only maximal terms (terms where the set of variables are not a subset of any other terms in the ANF of the function) of the ANF of the chaining function, to avoid computing smaller terms which will never be maximal. Due to this pruning step, this ANF can also have only up to R terms, each representing a monomial of at most n variables. Thus, there are at most $O(R)$ additions and $O(nR)$ multiplications, each with cost $O(kR^3)$, for a total cost of $O(knR^4)$. This is repeated for each block (at most k blocks), yielding $O(nk^2R^4)$. Recalling the expression for R, we can obtain a simpler upper bound in terms of n and k:

$$R \leq \frac{\Pi_{s_i \in S} s_i}{max(S)} \leq \frac{\Pi_{s_i \in S} \frac{n}{k}}{max(S)} = \frac{1}{max(S)} \left(\frac{n}{k}\right)^k$$

Substituting this upper bound for R into the expression $O(nk^2R^4)$ and noting that $max(S)$ is always at least $\frac{n}{k}$ gives a total runtime complexity of

$$O\left(\frac{k^6}{n^3}\left(\frac{n}{k}\right)^{4k}\right)$$

which is indeed polynomial with respect to n, the number of bits in the CMPR, and exponential with respect to k, the number of MPRs. With additional careful analysis, we believe that this upper bound can be tightened.

In practice and especially for large CMPRs, the CMPR Root Expression Algorithm outperforms the Berlekamp-Massey algorithm. This difference in performance is demonstrated in Figure 6 where each algorithm was applied to every possible CMPR configuration up to 300 bits and using degree 3 chaining functions. Using higher-degree chaining functions (higher than degree 3) for this experiment was not feasible, as the generated sequences would have much larger linear complexity, and we would have been unable to obtain enough data points for the Berlekamp-Massey algorithm to generate the figures. We note that for visual clarity, the Berlekamp-Massey results are shown on the plot only for linear complexity lower than 10000, as the Berlekamp-Massey algorithm runtime increased drastically. However, an extrapolated comparison is also given to compare our algorithm to the expected behavior of Berlekamp-Massey.

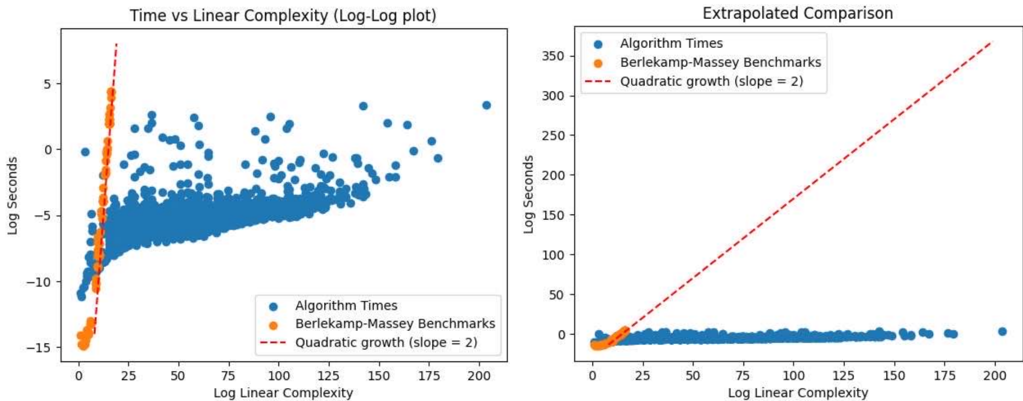


Figure 6: Runtime and Linear Complexity Comparison Between the Berlekamp-Massey Algorithm and CMPR Root Expression Algorithm

Remark 2. We can draw several structural insights about chaining as a composition strategy from Algorithm 1. First, from Theorem 4 and the density of the adjugate matrices, chaining with the current state of the linear feedback register appears to create complex relationships between the every output of the register and every input to the register (i.e., the outputs of the chaining function). Due to the polynomial entries in the adjugate matrices, the output stream of the MPR is the XOR of many different shifts of each of the inputs, creating different complex relationships between each output and input pair. Additionally, by observing the propagation of root expressions (e.g., in Example 10), we can observe that the nonlinear effects of each chaining function are amplified as they pass through subsequent chaining functions. This yields some insight into how CMPRs are able to generate high linear complexity, even with relatively simple chaining functions.

5.4 Degenerate Cosets and a Pessimistic Estimate

In the previous section, we mentioned degeneracy and observed an example, but did not explore the concept further. Historically, while prior works such as [Key76] and [Rue86] acknowledge the risk of degeneracies, they also claim that they are relatively insignificant and that the upper bound is a good estimate of the true linear complexity. However, in our case, we have multiple filters and opportunities for degeneracy. In addition, certain types of degeneracies pose a larger threat to our estimates than they did to Key [Key76]. In this section, we will explore modifications to the algorithm which help to account for these types of degeneracies, and give an additional estimate that aims to be reasonably pessimistic, to aid designers trying to ensure a given linear complexity.

In [Rue86], Rueppel analyzes the probability of degeneracy in a filter generator under a simplifying model which assumes each irreducible polynomial of degree n independently degenerates with probability $1/2^n$. We will work under the similar assumption that in the output of the chaining function each primitive polynomial degenerates independently with probability $1/2^n$. However, even if this set of roots is nondegenerate in the output of the chaining function, there is an additional $1/2^n$ chance it degenerates while propagating through the next MPR under the same simplifying assumptions. We can use this to estimate that each primitive polynomial with size in the final answer n does not degenerate with probability $(2^n - 1)^2/2^{2n}$.

There is a different edge-case with a more dramatic effect. Specifically, consider the case of an n-bit filter generator: it is unlikely a designer would include an AND of all n bits, and therefore such a designer would never encounter the full coset class $\langle e \cdot e \rangle$. This coset class is special since it is the only one that contains the element 1 from the base field, $\mathbb{F}_2$. Lemma 3 and the analysis in [Rue86] explicitly apply to elements outside the base field because multiplying a sequence by the identity can result in cancellations of whole sequences. These cancellations produce degeneracies which are far more dramatic than those which happen by chance. For CMPRs, as coset classes pass through multiple nonlinear chaining functions, they mix and grow multiplicatively. Even with low-degree chaining, it is possible to encounter these large coset classes when designing CMPRs. The analysis of CMPRs is unable to ignore the phenomenon, even though it did not pose a problem in the analysis of filter or combination generators.

When calculating the lower estimate, we can reuse the same root expression calculated for the upper bound. We simply modify the algorithm to evaluate the root expressions in a way that disregards any coset classes that have weight equal to their exponent, or any term containing such a coset class. This is equivalent to pessimistically assuming that all possible large cancellations did in fact occur and remove a large number of roots. We then also evaluate each root term weighted by the probability of normal degeneracy, as discussed in the previous paragraph. Both of these modifications create a very pessimistic estimate, which is almost always far lower than tests for the actual linear complexity. Additionally, the degeneracies these modifications account for become less likely as n becomes larger. For larger MPRs and bigger CMPRs the need to account for these edge cases behavior decreases and we can expect the upper bound to be an increasingly good estimate of the true linear complexity. These modifications mostly help to avoid coincidental degeneracies that occur when small sizes such as 2 or 3 are used and help to give bounds that contain the true linear complexity with high probability.

To demonstrate the accuracy of this method, we constructed 6400 CMPRs with randomly generated chaining and analyzed them both with the estimation and Berlekamp-Massey algorithms. The CMPRs used were of size less than 20 due to the inefficiency of running the Berlekamp-Massey algorithm [Mas69] on large CMPRs. However, the estimation algorithm is scalable and extends to larger sizes. When generating Berlekamp-Massey data points for comparison, the CMPRs were initialized to random initial states. However, due to the small sizes of the MPRs used, the Berlekamp-Massey trials were more

likely to start in smaller cycles. To avoid skewed results caused by the shorter period, the test was rerun. The 6400 CMPRs used were divided into the 16 possible combinations of Mersenne primes summing to size less than 20. 400 tests were run on each Mersenne prime combination. The Berlekamp-Massey algorithm and the estimation algorithm were run on each construction to verify that the true linear complexity falls within the bounds the estimation algorithm provides. Of the 6400 tests, the algorithm was correct 99.547% of the time, despite linear complexities ranging over several orders of magnitude, depending on the chaining used. The results are summarized in Table 5, where "Sizes Used" denotes the sizes of the component MPRs in bits, and "Accuracy" denotes the proportion of tests for which the actual linear complexity fell inside the predicted range. Additionally, for the larger sizes where this algorithm is necessary, we expect it to be even more accurate than demonstrated here.

Table 5: Estimation Algorithm Tests

Sizes Used	(3,2)	(5,2)	(7,2)	(13,2)	(17,2)	(5,3)	(7,3)	(13,3)
Accuracy	1	0.995	1	1	1	0.9975	1	1

Sizes Used	(7,5)	(13,5)	(5,3,2)	(7,3,2)	(13,3,2)	(7,5,2)	(7,5,3)	(7,5,3,2)
Accuracy	1	1	0.995	0.9925	0.9875	0.97	0.995	0.995

5.5 Examples and Comparisons

5.5.1 Linear Complexity of a Simple 128-bit CMPR

As shown in Table 4, we use the Mersenne exponents $\{61, 31, 19, 17\}$ to construct a 128-bit CMPR, where we chain the four MPRs by descending size. We choose primitive polynomials as shown in Table 6.

Table 6: Chosen MPRs

Size	Primitive Polynomial	Update Polynomial
61	$x^{61} + x^{58} + x^{54} + x^{49} + 1$	x
31	$x^{31} + x^{30} + x^{28} + x^{22} + 1$	x
19	$x^{19} + x^{17} + x^{14} + x^{10} + 1$	x
17	$x^{17} + x^{14} + 1$	x

To achieve the most direct comparison possible to existing constructions, we chose $U = x$ for all MPRs, meaning these MPRs are also Galois LFSRs. For the chaining functions, we choose to connect the two least significant bits of each MPR to an AND gate, the output of which is XORed with the feed-in to the most significant bit of the next MPR, similar to the structure shown in Figure 7b. We consider the sequence produced by the least significant bit of the 17-bit MPR to be the output sequence of the register; this is what we measure for linear complexity. For this construction, the estimation algorithm gives a lower estimate $\approx 2.032 \times 10^{19}$ and an upper bound of $\approx 2.923 \times 10^{19}$, both of which are between 2^{64} and 2^{65} We also verified this to the best of our ability using the Berlekamp-Massey algorithm to establish a true lower bound on the linear complexity. Due to computational constraints, we terminated the algorithm after processing 2,060,000 bits of input and observed a linear complexity growing approximately half as fast as the number of bits analyzed to 1,030,000 at the termination of the algorithm. This behavior is expected, as in a truly random sequence, the linear complexity should grow approximately half as fast as the number of bits processed. Note that the scale of what was able to be processed with the Berlekamp-Massey algorithm is very small relative to the estimate, demonstrating the need for the estimation algorithm. The linear complexity is also large despite very sparse

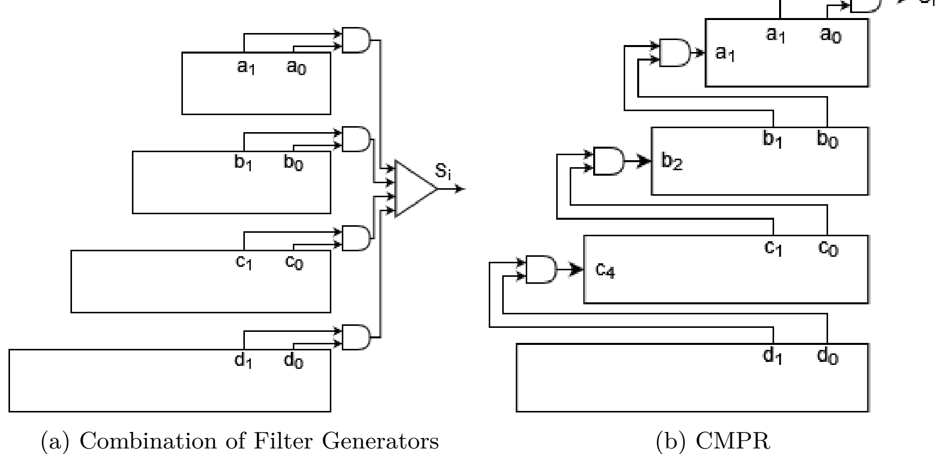
chaining and simple choices for update polynomials; in fact, this construction uses only slightly more hardware than a 128-bit LFSR. The linear complexity can be increased as needed with more complicated designs or by adding more MPRs.

5.5.2 Comparison of CMPRs with Filter and Combination Generators

Filter generators, combination generators, and combinations thereof have historically served as tools to build large-period, nonlinear structures from LFSRs. In this experiment, we will compare the linear complexities of two similar constructions: one using a traditional filter and combination structure, and the other using a similar CMPR structure. We will show that the CMPR can have a much higher linear complexity while using very similar hardware building blocks. We chose to use a CMPR constructed from the four smallest Mersenne exponents, $\{7, 5, 3, 2\}$, to simplify the necessary computations, such as the Berlekamp-Massey algorithm. Again, we opt to use the same Galois LFSRs in both constructions for a more direct comparison. The different primitive polynomials we chose are shown in Table 7.

Table 7: Chosen MPRs

MPR Name	Size	Primitive Polynomial	Update Polynomial
A	2	$x^2 + x + 1$	x
B	3	$x^3 + x^2 + 1$	x
C	5	$x^5 + x^3 + 1$	x
D	7	$x^7 + x^6 + x^4 + x + 1$	x



(a) Combination of Filter Generators (b) CMPR

Figure 7: Generator Structure

Figure 7 highlights the high-level similarities and differences between the two different constructions. The figure excludes the feedback functions for the LFSRs since their structure is already understood. In both diagrams, an arrow originating from a box represents the signal output by the labeled bit. Similar to Example 10 and Subsection 5.5.1, the chaining for the CMPR connects the AND of the two least significant bits of each MPR to the most significant bit of the next, which is denoted by the arrow into each MPR in Figure 7b. The outward arrow S_i denotes the output stream of values which we measure to calculate linear complexity. In an attempt to make an even comparison, the combination of filter generators also uses the AND of the two least significant bits of each MPR. However, instead of feeding into the next MPR, these sequences are combined again using a nonlinear

combining function, denoted by the triangle in Figure 7a. This combining function can be selected from the space of 4-input boolean functions. For example, in the setup of Figure 7a, using a 4-input NAND gate as the combining function yields a sequence with linear complexity of 7561. This setup ensures that the MPRs and AND gates used in Figures 7a and 7b are as similar as possible; the only difference is what is done with the outputs of the AND gates.

For the combination of filter generators in Figure 7a, using the upper bounds established in Definitions 25 and 26 and naively composing the upper bounds yields the following optimistic bound on the linear complexity of the output sequence:

$$\Lambda(S_i) \leq \prod_{i\in\{7,5,3,2\}} \left(\left(\sum_{j=1}^{2} \binom{i}{j} \right) + 1 \right) = 12992$$

Using the root expression techniques discussed earlier, this upper bound can be lowered to 9744, which we confirmed was achievable using the Berlekamp-Massey algorithm. We also experimentally determined that the linear complexity of the CMPR shown is 27090 which is a significant improvement over even the optimistic bound. Thus, use of the CMPR construction of Figure 7b can compose LFSRs with significantly higher linear complexity compared to traditional methods like filter and combination generators. Although linear complexity is far from a direct measure of security, this is a promising result.

6 PRNG Analysis

6.1 Pseudorandom Number Generator Specification

We now apply CMPR theory to a practical use case: pseudorandom number generator (PRNG) design. We consider a family of 9 different CMPR sizes formed by first using the 4 smallest Mersenne exponents, then using the 5 smallest, etc., up to the 12 smallest. Therefore, the largest CMPR size in our family uses MPRs of sizes $2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107$, and 127. The result is that the 9 CMPRs in this family have sizes of $17, 30, 47, 66, 97, 158, 247, 354$ and 481 bits. These MPRs are chained in descending order by size (e.g., the chaining function of each MPR only takes input from larger MPRs). Furthermore, we restrict our chaining functions to consist of 2-, 3- or 4-input AND gates feeding 2- or 3-input XOR gates. We could have used 4-input XOR gates, but using 3-input XOR gates allows for a more simple representation of the ANF of the 17-bit CMPR, which is useful for some of the cryptanalytic examples in this section. Figure 8 shows our smallest CMPR PRNG, which has 17 state bits, and the output is taken to be the least significant bit of the 17-bit state.

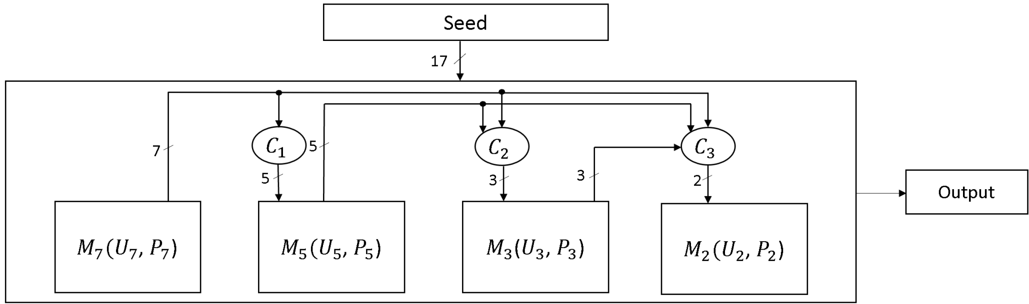


Figure 8: 17-bit CMPR PRNG

Throughout this section, we use the 17-bit CMPR as a recurring example. The small size of this CMPR makes it possible to include descriptions in reasonable amounts of space and detail as well as making certain analyses computationally feasible. The update and primitive polynomials are given in Table 8, and the randomly generated chaining logic is described in Table 9.

Table 8: 17-bit CMPR Update and Primitive Polynomials

MPR	Size	Update Polynomial	Primitive Polynomial	State Bits
M_7	7	$U_7 = x^5 + 1$	$P_7 = x^7 + x + 1$	$a_{16}a_{15}a_{14}a_{13}a_{12}a_{11}a_{10}$
M_5	5	$U_5 = x^4 + x + 1$	$P_5 = x^5 + x^2 + 1$	$a_9a_8a_7a_6a_5$
M_3	3	$U_3 = x^2 + 1$	$P_3 = x^3 + x + 1$	$a_4a_3a_2$
M_2	2	$U_2 = x + 1$	$P_2 = x^2 + x + 1$	a_1a_0

Table 9: 17-bit CMPR Chaining Logic

Chaining Function and Bit	Affected State Bit	Chaining Logic
Bit 0 of $\mathcal{C}_3$	a_0	$1 \oplus a_6 \oplus (a_2a_3a_7a_{13})$
Bit 1 of $\mathcal{C}_3$	a_1	$a_2 \oplus a_3 \oplus (a_4a_9a_{11}a_{14})$
Bit 0 of $\mathcal{C}_2$	a_2	$1 \oplus a_{10} \oplus (a_5a_7a_{11}a_{15})$
Bit 1 of $\mathcal{C}_2$	a_3	$a_5 \oplus a_7 \oplus (a_8a_9a_{14}a_{15})$
Bit 2 of $\mathcal{C}_2$	a_4	$a_{11} \oplus a_{13} \oplus (a_6a_7a_{10}a_{16})$
Bit 0 of $\mathcal{C}_1$	a_5	$1 \oplus a_{14} \oplus (a_{10}a_{11}a_{12}a_{13})$
Bit 1 of $\mathcal{C}_1$	a_6	$a_{10} \oplus a_{15} \oplus (a_{11}a_{12}a_{13}a_{14})$
Bit 2 of $\mathcal{C}_1$	a_7	No Chaining
Bit 3 of $\mathcal{C}_1$	a_8	$1 \oplus a_{10} \oplus (a_{11}a_{12}a_{14}a_{16})$
Bit 4 of $\mathcal{C}_1$	a_9	$a_{11} \oplus a_{12} \oplus (a_{13}a_{14}a_{15}a_{16})$

6.1.1 Pseudorandom Number Generator Initialization

Each of our CMPR-based PRNGs takes as input a binary seed k. The length of k cannot exceed n, where n refers to the CMPR size. Once the seed is provided, the bits of the initial state of the CMPR are loaded with the seed such that the most significant bits of the seed correspond to the most significant bits of the initial state. If the length of k does not equal the CMPR size, the remaining least significant bits of the initial state are set to a fixed value of 1. We recommend that the number of fixed bits be at least the size of the smallest MPR (e.g., the final MPR with bit a_0). Thus, for this family, we require that at least the two least significant bits be part of the fixed segment.

The CMPR is then clocked a number of times as initialization to randomize the state of the CMPR. Each clocking of the CMPR is termed an initialization round. Upon completion of the initialization rounds, the generation of the pseudorandom output z begins. The number of necessary initialization rounds is not fully understood, but we will discuss a heuristic approach to the problem. If the chaining functions are approximately balanced (in that their truth tables contain roughly the same number of ones and zeroes), we expect it to take approximately two clock cycles for a change to propagate through a chaining function and cause a change further down the chain of MPRs. For this reason, we recommend at least $2q$ initialization rounds, where q is the number of MPRs. Thus, for the 17-bit CMPR shown in Figure 8, $q = 4$, and we recommend at least $4 * 2 = 8$ initialization rounds. Each bit of the pseudorandom output is generated by taking the least significant bit of the CMPR per clock cycle, such that $z_i = a_0[i]$.

6.1.2 CMPR Pseudorandom Number Generator Design Considerations

To better explain the application of CMPR theory presented in the context of pseudorandom number generation, it is important to outline the reasoning behind the CMPR PRNG design choices. Observe that the MPRs in Figure 8 are chained from largest to smallest; that is, any chaining function from $M_A \to M_B$ is chosen such that $\text{length}(M_A) > \text{length}(M_B)$. This design choice is made to maximize the linear complexity of the CMPR and is consistent with the results presented in Section 5. Additionally, the CMPR is clocked to sufficiently randomize the state of the CMPR. While the specific number of initialization rounds is a rather flexible design choice, initialization to mask the initial state is necessary. When choosing a bit or combination of bits to generate an output stream from the CMPR state, bits closest to the least significant bit of the CMPR are preferred in the output generation since these bits tend to best display the nonlinear effects from the chaining functions from higher-order MPRs. In our PRNG design, we simply clock the CMPR and do not permute the internal state or introduce any additional inputs to the internal state besides the seed. Thus, the bits in the largest MPR evolve linearly since the largest MPR does not intake any chaining functions from other registers. Bits from the largest MPR should not be used in output generation since their linearity presents an exploitable vulnerability and increases the attack surface. Additionally, in our PRNG design, we chose to simply use the value of the least significant bit over time to generate the output, but we alternatively could have used a balanced filter function, taking into consideration which bits we used.

6.2 Data Generation and Statistical Analysis

6.2.1 NIST Statistical Test Suite

To verify the random number generation properties and cryptographic security of the CMPR PRNG output, we applied the NIST Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications [BRS+10] to bitstreams of length 10^6 generated by the CMPR PRNG from a set of random seeds. 10^6 is the minimum bitstream length which ensures that all of the tests in the suite can be applied. The NIST test suite was applied to the 9 CMPR PRNG sizes from Subsection 6.1, with the modification that all MPRs use the update function $U(x) = x$. For statistical testing, we opted to use the simplest update function $U(x) = x$ to demonstrate that even with the simplest possible MPR instantiation, the outputs still exhibit desirable statistical properties. The 17-bit CMPR PRNG consistently failed the FFT test (while passing all other tests) because the period of the CMPR was too small for the FFT test, which evaluates the periodicity of the input sequence. However, all other CMPRs passed every test. Although satisfying the requirements set forth by the NIST test suite alone is not enough to guarantee security, this is a good indication that reasonably sized CMPR PRNGs (30 bits and above) produce output streams with good statistical properties.

6.2.2 Bit Contribution Statistical Tests

To further assess the cryptographic strength of the CMPR PRNG, we ran several additional statistical tests on various versions of the CMPR PRNG. The family of bit contribution tests includes tests such as the bit contribution for key test, bit contribution for nonce test, bit contribution for plaintext test [PRKK19], and the similar alternative test for the strict avalanche criterion [WT86]. These statistical tests are used to measure properties such as confusion and diffusion [Sha45], completeness [KD79], the avalanche effect [Fei73], and the strict avalanche criterion [WT86]. The result of this bit contribution for seed test is a dependence matrix D with the number of rows equal to the length of the seed and the number of columns equal to the length of bitstream generated. The value $D(i, j)$ in the

dependence matrix indicates how frequently bit j in the output flips when bit i in the seed is flipped, across T randomly generated seeds.

For each test, we set the number of trials T to 10,000 as recommended by [PRKK19]. To pass the test, every value in the dependence matrix should be approximately $T/2 = 5000$, as we expect a change in any bit of the seed to cause a change in every bit of the output with probability 1/2. We can determine if a PRNG passes the bit contribution test through analysis of the dependence matrix. Statistically, we expect to see each element of the matrix resemble the binomial distribution Binomial$(T, 1/2)$. This can also be approximated by the easier-to-calculate normal distribution, Normal$(\mu = T/2, \sigma^2 = T/4)$. According to NIST, a recommended $10,000$ trials yields an interval of $4750 < D(i,j) < 5250$ that we expect the matrix values to fall within with high confidence [PRKK19]. Specifically, for 10,000 trials, we expect each value to fall within that range with a 99.999943% probability.

A preliminary test of the 17-bit CMPR PRNG with no initialization rounds, 500 trials, and an output of 500 bits revealed that the first few values of the output rarely changed when we perturbed bits of the seed as part of the bit contribution test. Essentially, without initialization rounds, the changes made to the bits of the CMPR's initial state by the bit contribution test do not have sufficient time to propagate through the entire CMPR and thus the first few bits of the output are often similar to the first few bits of the output of a very similar seed, which is undesirable behavior. We determined that in the case of this particular 17-bit CMPR PRNG, as few as 8 initialization rounds would be sufficient to yield an output with statistically desirable properties. This led to further investigation, resulting in the development of the heuristic in Subsection 6.1.1. However, to be conservative, and ensure that the changes to the initial states of the CMPR in the bit contribution tests are sufficiently propagated, we use 100 initialization rounds for all future tests.

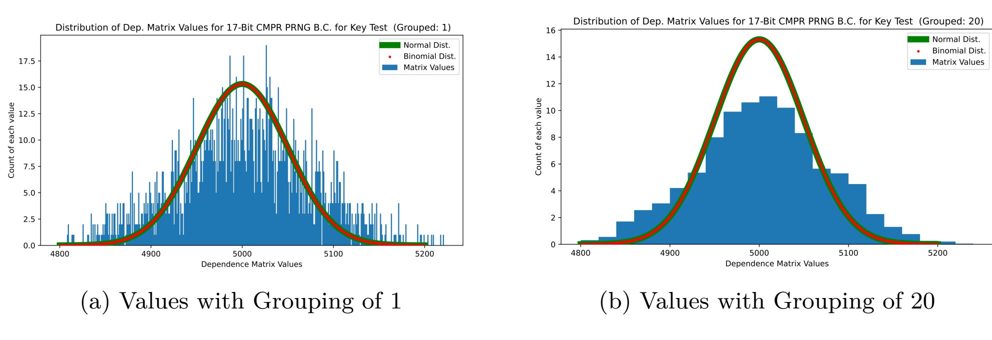
A second observation from the preliminary test results is that the final MPR always contributes linearly to the output because it is never filtered through a chaining function. Thus, for these CMPR PRNGs (which have a least significant MPR of size two), any changes made to the least significant two bits cause a predictable pattern of changes in the output. This is the reason why the CMPR PRNG specification in Subsection 6.1.1 states that the fixed segment of the initial state of the CMPR must be at least the size of the last MPR. Thus, for all our tests on the 17-bit, 30-bit, and 47-bit CMPR PRNGs, we have seeds of length 15, 28, and 45 respectively, no initialization vectors, and the two least significant bits fixed. Subsequently, when we ran the bit contribution for seed test on the 17-bit, 30-bit, and 47-bit CMPR PRNGs, we used 10,000 trials, 100 initialization rounds, and generated a pseudorandom output of 128 bits. The resulting dependence matrices were analyzed using the NIST recommended bound of $[4750, 5250]$; the results are displayed in Table 10.

Table 10: Ranges of different bit contribution tests

CMPR size	$D(i,j)$ minimum value	$D(i,j)$ maximum value	Passes NIST bound
17	4808	5220	yes
30	4830	5181	yes
47	4820	5169	yes

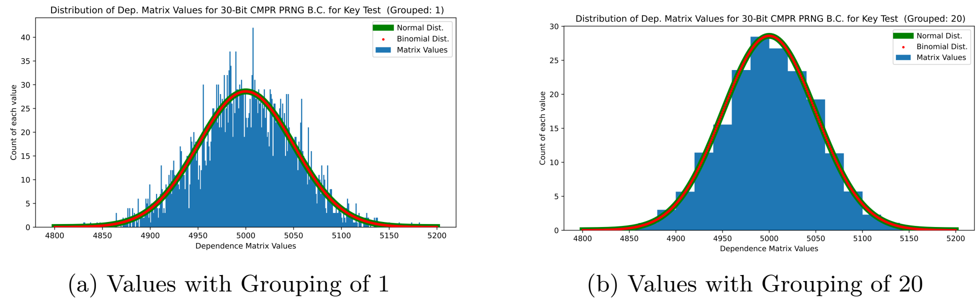
According to the range given by NIST, all tested CMPR PRNGs pass. However, visually inspecting the data offers additional insights. Figures 9, 10 and 11 show histograms of the data for the 17-bit, 30-bit, and 47-bit tests respectively, with the expected normal and binomial distributions overlaid. In each figure, subfigure (a) shows the actual data, while subfigure (b) shows histogram groupings of 20, which allows the distribution to be perceived more easily. The values in the 17-bit dependence matrix are concentrated more heavily in the tails of the distribution and have larger variance than expected, indicating a slight statistical weakness. However, this effect seems to disappear in larger tests, as both

the 30-bit and 47-bit tests seem to closely follow the expected distribution. We suspect that the 17-bit CMPR is too small and thus displays non-random behavior that vanishes as the size is increased, similar to the results observed in Subsection 6.2.1. However, again, the distributions for the larger CMPRs we tested appear to follow the expected distribution. Overall, the results of the bit contribution for seed test constitute strong experimental evidence that for reasonable sizes, even simple CMPR PRNGs exhibit a complicated relationship between seed and output stream; a change of a single bit of the seed changes each bit in the output with 50% probability.



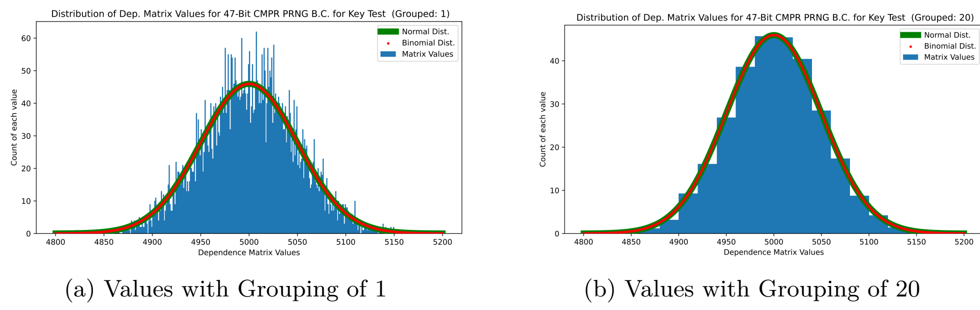
(a) Values with Grouping of 1 (b) Values with Grouping of 20

Figure 9: 17-Bit CMPR PRNG: Distribution of Bit Contribution Test Matrix Values



(a) Values with Grouping of 1 (b) Values with Grouping of 20

Figure 10: 30-Bit CMPR PRNG: Distribution of Bit Contribution Test Matrix Values



(a) Values with Grouping of 1 (b) Values with Grouping of 20

Figure 11: 47-Bit CMPR PRNG: Distribution of Bit Contribution Test Matrix Values

6.3 Cryptanalysis

6.3.1 Setup

To analyze the cryptographic weaknesses of a CMPR-based PRNG, the authors of this paper were divided into two teams, offensive and defensive, to model a real-world attack scenario. The defensive team generated a small PRNG which the offensive team then tried to break by either finding the initial state or predicting the future output. Similar to Subsections 6.1 and 6.2, we use the 17-bit example CMPR. Although there are obvious weaknesses due to the small size, using this PRNG simplified the attack process. This allowed us to obtain a better understanding of the process and help determine which types of attacks are likely to work on larger CMPRs.

The design the defensive team chose was a simple PRNG scheme where the least significant bit of a 17-bit CMPR was used directly as pseudorandom output. Although this is insecure due to the small size of the CMPR, we wanted to start by finding successful attacks on smaller and weaker variants as a proof-of-concept. Full knowledge of the CMPR, including all primitive polynomials, update polynomials, and chaining functions were provided to the offensive team, as well as a working model of the system. The CMPR consists of 4 MPRs with sizes 7, 5, 3, and 2, chained in descending order (7 feeds 5, etc.). Table 8 shows the update and primitive polynomials chosen for these MPRs. Using the update polynomial, primitive polynomial, and chaining, the full ANF used for the 17-bit CMPR is as follows:

$$
\begin{aligned}
c_{16}[t+1] &= c_{11}[t] \oplus c_{16}[t] \\
c_{15}[t+1] &= c_{10}[t] \oplus c_{15}[t] \oplus c_{16}[t] \\
c_{14}[t+1] &= c_{15}[t] \oplus c_{16}[t] \oplus c_{14}[t] \\
c_{13}[t+1] &= c_{15}[t] \oplus c_{14}[t] \oplus c_{13}[t] \\
c_{12}[t+1] &= c_{12}[t] \oplus c_{14}[t] \oplus c_{13}[t] \\
c_{11}[t+1] &= c_{11}[t] \oplus c_{12}[t] \oplus c_{13}[t] \\
c_{10}[t+1] &= c_{10}[t] \oplus c_{12}[t]
\end{aligned}
$$

$$
\begin{aligned}
c_{9}[t+1] &= c_{9}[t] \oplus c_{5}[t] \oplus c_{11}[t] \oplus c_{12}[t] \oplus (c_{13}[t]c_{14}[t]c_{15}[t]c_{16}[t]) \\
c_{8}[t+1] &= 1 \oplus c_{9}[t] \oplus c_{8}[t] \oplus c_{10}[t] \oplus (c_{11}[t]c_{12}[t]c_{14}[t]c_{16}[t]) \\
c_{7}[t+1] &= c_{8}[t] \oplus c_{7}[t] \oplus c_{10}[t] \oplus c_{15}[t] \oplus (c_{11}[t]c_{12}[t]c_{13}[t]c_{14}[t]) \\
c_{6}[t+1] &= c_{5}[t] \oplus c_{6}[t] \oplus c_{7}[t] \\
c_{5}[t+1] &= 1 \oplus c_{6}[t] \oplus c_{5}[t] \oplus c_{14}[t] \oplus (c_{10}[t]c_{11}[t]c_{12}[t]c_{13}[t])
\end{aligned}
$$

$$
\begin{aligned}
c_{4}[t+1] &= c_{2}[t] \oplus c_{11}[t] \oplus c_{13}[t] \oplus (c_{6}[t]c_{7}[t]c_{10}[t]c_{16}[t]) \\
c_{3}[t+1] &= c_{5}[t] \oplus c_{7}[t] \oplus c_{4}[t] \oplus (c_{8}[t]c_{9}[t]c_{14}[t]c_{15}[t]) \\
c_{2}[t+1] &= 1 \oplus c_{2}[t] \oplus c_{3}[t] \oplus c_{10}[t] \oplus (c_{5}[t]c_{7}[t]c_{11}[t]c_{15}[t])
\end{aligned}
$$

$$
\begin{aligned}
c_{1}[t+1] &= c_{3}[t] \oplus c_{2}[t] \oplus c_{0}[t] \oplus (c_{4}[t]c_{9}[t]c_{11}[t]c_{14}[t]) \\
c_{0}[t+1] &= 1 \oplus c_{1}[t] \oplus c_{6}[t] \oplus c_{0}[t] \oplus (c_{2}[t]c_{3}[t]c_{7}[t]c_{13}[t])
\end{aligned}
$$

For a baseline, the offensive team implemented a brute force attack running through the $(2^2-1)(2^3-1)(2^5-1)(2^7-1) = 82{,}677$ states in the main cycle of the CMPR. In the sections to follow, various approaches and attempts by the offensive team to break the PRNG are described.

6.3.2 Algebraic Attacks

Algebraic attacks, as described by Courtois and Meier [CM03], analyze a system by constructing a set of multivariate polynomial equations that relate the known variables to the desired variables. Given enough equations, the system is over-constrained, and various techniques exist for solving the desired variables. In the given system, the only known output is the pseudorandom output streams, and the desired variables are the seed bits. Equations for output bits in terms of initial state can be obtained simply by composing the ANF of the CMPR over multiple cycles. Example 11 gives the first and second equations.

Example 11.

$$\begin{aligned} c_0[1] &= 1 \oplus c_1[0] \oplus c_6[0] \oplus c_0[0] \oplus (c_2[0]c_3[0]c_7[0]c_{13}[0]) \\ c_0[2] &= 1 \oplus c_1[1] \oplus c_6[1] \oplus c_0[1] \oplus (c_2[1]c_3[1]c_7[1]c_{13}[1]) \\ &= c_3[0] \oplus c_2[0] \oplus c_5[0] \oplus c_7[0] \oplus c_1[0] \oplus (c_4[0]c_9[0]c_{11}[0]c_{14}[0]) \\ &\quad \oplus (c_2[0]c_3[0]c_7[0]c_{13}[0]) \oplus (c_2[1]c_3[1]c_7[1]c_{13}[1]) \end{aligned}$$

The second equation leaves four bits in terms of the first clock cycle instead of the initial state for readability – after multiplying the composed polynomials, the second equation has 157 terms.

The straightforward approach would be to solve the system of equations once it is over-constrained. Resistance to algebraic attacks is rooted in the difficulty of solving multivariate polynomial equations, a problem which is NP-complete even with quadratic equations modulo 2 [CP03] [DS09]. Due to the non-linearity of the chaining functions, new high-degree terms are introduced each clock cycle and generally persist from cycle to cycle until a bound is reached. The attack team generated equations for the first 17 output bits of the PRNG. The precise number of monomials and degree of these equations are shown in Figure 12.

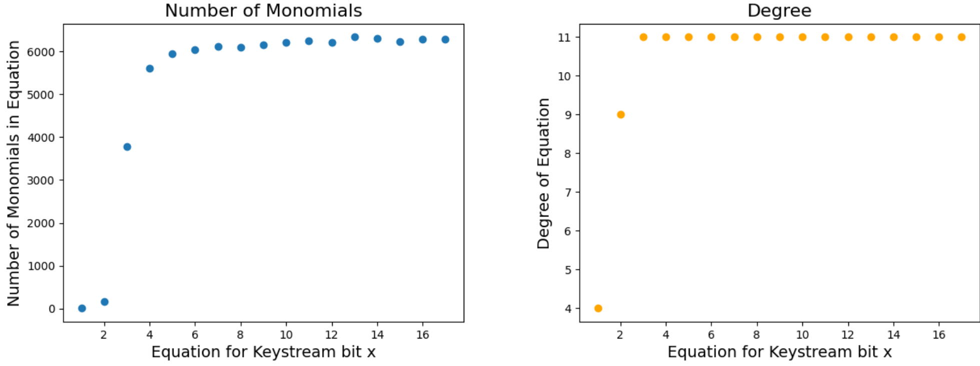


Figure 12: Number of Monomials and Degree of Composed Equations

Even for a small, obviously insecure, CMPR such as this, carrying out the algebraic attack involves solving a system of polynomials of degree 11 with over 6000 terms in each polynomial equation. In addition to the difficulty of solving these equations, generating them was a nontrivial task. Part of our reason for using an example as small as 17 bits is that we could not feasibly generate this data for larger examples. However, this small example allows us to demonstrate the reasons that the attack is difficult to apply, especially as the system scales. In general, the number of possible monomials in a CMPR equation is slightly greater than the linear complexity. In a random equation of this form, we would expect roughly $\frac{1}{2}$ of these monomials to be present. The linear complexity of this 17-bit

CMPR is 12,387, which is the number of possible monomials. Thus, we would expect the number of monomials to plateau just above 6000; the graph confirms this behavior. CMPRs can achieve high linear complexity (and thus high degree and number of monomials in their equations). We believe an algebraic attack would require solving an infeasibly large system of equations, and thus the straightforward approach to algebraic attacks is slower than brute force in all but exceptionally sparse and low-degree constructions.

Linearization is a simple technique for solving the system of equations by substituting nonlinear terms with new independent variables to solve the resulting linear system with methods such as Gaussian elimination [DS09]. Concerning the equations for the resulting PRNG found using composition (see Example 11), linearization is extremely inefficient given the number of nonlinear terms in each equation that would have to be replaced; the third equation has 3777 alone. One idea the attack team had to cope with this was to avoid composing variables of nonlinear terms (such as the second equation in Example 11); however, the resulting system still has 151 unique variables. Even with sub-cubic methods to solve the system of equations, this approach is infeasible due to the number of nonlinear terms in the equations.

Alternate algorithms exist with the intent of reducing the complexity of finding solutions to the system of equations. One example is with the use of a Gröbner Basis algorithm such as Buchberger's algorithm [AFI$^+$04] or the F_4 algorithm [Fau99]. Buchberger's algorithm has exponential or worse runtime [CKPS00]. The F_4 algorithm is at least an order of magnitude faster but does not improve the worst-case complexity [Fau99]. Another example is the XL algorithm [CKPS00]. In [AFI$^+$04], however, it was shown to be a redundant version of F_4. Due to the expected inefficiency of these algorithms, in conjunction with the difficulty of generating the many large equations in the first place, the attack team did not consider these alternate algorithms to be a fruitful area of investigation.

Fast Algebraic Attacks, see [Cou03] and [Arm04], use boolean functions describing the system in the form:

$$F(L^t(K), \ldots, L^{t+\delta}(K), k[0], \ldots, k[t+\delta]) = 0$$

where $k[t]$ is the output bit at time t, K is the seed, and L^t is a linear boolean function at time t describing the update of a component in the system. Thus, Fast Algebraic Attacks are not immediately applicable to CMPRs because the internal state does not update linearly.

Correlation Attacks explored the vulnerability of the CMPR to a variety of approaches, including fast correlation attacks. These are divide-and-conquer-based techniques that attempt to recover the initial state of every constituent MPR by exploiting observed relationships between these components with knowledge of some output bits, along with linear relations to compute implied probabilities that output bits match the state of the CMPR at any time t based on constructed and observed parity relationships between MPR bits.

To implement a fast correlation attack on the 17-bit CMPR PRNG, the PRNG was clocked to produce output bits, aiming to exploit statistical correlations between the output bits and the states of the MPRs. For this 17-bit CMPR composed of 2-bit, 3-bit, 5-bit, and 7-bit MPRs chained together, around 60 clock cycles were utilized to generate a sufficient number of output bits for the attack. Using these output bits, 32 relations were constructed by identifying patterns between these output bits and the internal states of each MPR. However, the attack failed primarily due to under-constraining in the early clock cycles, resulting in a system of equations that was rank-deficient. Another key reason for failure was that the relations constructed were negligible probabilistically in the correlations between keystream bits and internal state bits they provided, such that information gain about an internal MPR bit from knowing a keystream bit for the relations was not greater than that provided by a random guess of the MPR bit state.

In the early stages, the limited output bits led to fewer relations than needed to constrain the internal states. With the number of equations m (the relationships formed) being less than the number of variables n (the internal states), the system became under-constrained: $\text{rank}(A) < n$ (which implies infinitely many solutions) where $\text{rank}(A)$ is the rank of the matrix A representing these relations. This rank deficiency created significant ambiguity, compounded by weak correlations represented as follows: $C(X, Y) \approx 0$ where the metric $C(X, Y)$ used for the attack was Pearson's correlation coefficient between output bits X (the generated output bits) and internal states Y (the states of the MPRs), indicating that the output provided little information about the internal states. While additional clock cycles produced more output bits, the added complexity did not resolve the uncertainties from the early stages. The probabilities associated with the MPR states remained largely unchanged $P(S|K) \approx P(S)$ where $P(S|K)$ is the conditional probability of the MPR states S (the specific states to be recovered) given the observed output bits K (the bits produced during the attack). This suggests that knowing the output and MPR construction (across different correlational statistics) did not significantly alter the probabilities of the internal states, further obfuscating internal state retrieval. As such, the combination of early under-constraining, the rank deficiency of the relations, and weak correlations between output and internal states ultimately led to the failure of the correlation attack on the CMPR.

6.3.3 Cube Attacks and Cube Testers

The **Cube Attack**, defined in [DS09], treats the system as a black box polynomial p in terms of the seed bits $x[0], \ldots, x[n-1]$ and tries to solve for those bits. $I \subset \{0, \ldots, n-1\}$ represents a subset of variables, and t_I is the product of those variables. For example, if $I = \{0, 1, 3\}$ then $t_I = x[0]x[1]x[3]$. The paper [DS09] describes how the black box polynomial may be rewritten as follows:

$$p(x[0], \ldots, x[n-1]) = t_I p_{S(I)} \oplus q(x[0], \ldots, x[n-1])$$

where $p_{S(I)}$ is referred to as the *superpoly* of I in p and q is the *remainder*. When the superpoly is linear, then t_I is referred to as a *maxterm*. A *cube* is a set of all possible assignments of 0/1 to the variables in t_I. For a vector v in the cube, $p_{|v}$ is the derived variable for v and p_I is the sum of derived polynomials over the cube. The main observation of the paper (Theorem 1 in [DS09]) is the following: for any polynomial p and index set I, $p_I \equiv p_{S(I)} \bmod 2$. This allows superpolys to be computed by summing over the cube. Thus, the idea behind cube attacks is to search for maxterms in an offline phase, use them to obtain linear superpolys (using Theorem 2 in [DS09]), and then solve the resulting system of linear equations combined with brute force if the seed bits are not entirely covered.

In our initial attempts, we searched for maxterms by computing a uniformly random index subset I to determine if $p_{S(I)}$ is linear (i.e., a maxterm). We were unable to find sufficiently many maxterms to perform a successful cube attack. In the following sections, we introduce the concept of monomial profiles to exploit a regularity in the CMPR, and then we describe how this vulnerability can be fixed.

6.3.3.1 Monomial Profiles: Notation and Definitions

To evaluate the susceptibility of the CMPR construction to cube attacks, we introduce the concept of *monomial profiles*, which capture the distribution of monomials that may appear in the ANF of the CMPR at different indices.

Definition 33 (Monomial Profile)**.** We assign a formal expression to the sequence of polynomials produced by a feedback register, where this expression represents the set of monomials that can appear in any of the polynomials. In the context of CMPRs, this

formal expression, called a monomial profile, consists of pairs $\langle e; w \rangle$ which correspond to sets of all monomials where:

- the variables are chosen from the constituent MPR of size e
- the number of variables chosen from the constituent MPR is between 1 and w, inclusive.

The product of any number of pairs corresponds to the set of monomials that can be formed by taking the product of monomials where exactly one monomial comes from each pair in the product. The addition of any number of expressions corresponds to the union of their corresponding sets of monomials.

Example 12. Consider a 17-bit CMPR constructed from a 7-bit MPR (bits 10-16), 5-bit MPR (bits 5-9), 3-bit MPR (bits 2-4), and 2-bit MPR (bits 0-1). Consider the monomial profile

$$\langle 7; 4 \rangle \langle 5; 2 \rangle + \langle 3; 2 \rangle \langle 2; 1 \rangle$$

Monomials which satisfy this profile will have either:

- between 1 and 4 variables from the 7-bit MPR multiplied with between 1 and 2 variables from the 5-bit MPR (e.g., $x_{16}x_{13}x_7x_6$), or
- between 1 and 2 variables from the 3-bit MPR multiplied with 1 variable from the 2-bit MPR (e.g., $x_3x_2x_0$).

The monomial profile of a CMPR allows us to describe the upper bounds on the number and degree of monomials within a function (in this case the polynomial representation of the bit relationships in each MPR). e and w were chosen to be reminiscent of the root expressions introduced in Section 5.2, as monomial profiles and root expressions exhibit very similar properties.

Proposition 1. Propositions 1-5 (Section 5.2) for root expressions also hold for monomial profiles.

Proof. Below, we reason about how Propositions 1-5 from Section 5.2 also apply to monomial profiles.

1. For a monomial that is the product of d variables from an $e-$bit MPR, there are $\binom{e}{d}$ such monomials. Then, summing over all possible monomial degrees between 1 and w gives the total number of possible monomials of degree 1 to w. The equation for this is the following:

$$|\langle e; w \rangle| = \sum_{i=1}^{w} \binom{e}{i}$$

2. If all variables used to form a monomial come from the same $e-$bit MPR, then the number of variables in the product (degree) of a set of such monomials must be at least 1 and less than or equal to the sum of the number of variables in each monomial. The equation for this is the following:

$$\prod_{i=1}^{k} \langle e; w_i \rangle = \left\langle e; \sum_{i=1}^{k} w_i \right\rangle$$

3. If monomials are selected from different MPRs, then the total number of ways to pick monomials is equivalent to the product of the number of ways to pick monomials from each MPR. The product of each combination of monomials yields a different monomial in the product. The equation for this is the following:

$$\left| \prod_{i=1}^{k} \langle e_i; w_i \rangle \right| = \prod_{i=1}^{k} |\langle e_i; w_i \rangle|$$

4. Consider two monomial profiles E_1 and E_2. $|E_1 + E_2| = |E_1 \cup E_2|$, which follows immediately from Definition 33, and $|E_1 \cup E_2| = |E_1| + |E_2| - |E_1 \cap E_2|$, from the principle of inclusion-exclusion. In total, the result is the following:

$$|E_1 + E_2| = |E_1 \cup E_2| = |E_1| + |E_2| - |E_1 \cap E_2|$$

5. Let E_1 and E_2 be two monomial profiles. If the sets of MPRs used to form the monomial profiles are not identical, then $E_1 \cap E_2$ is empty because if the two sets are not identical, then there must be at least one pair $\langle e; w \rangle$ that appears in one product, but not the other. Consider an arbitrary monomial m in the intersection of E_1 and E_2. Because m is in the monomial profile that contains $\langle e; w \rangle$, m must have at least one of the variables in the MPR corresponding to e. However, because m is also in the monomial profile which does not contain $\langle e; w \rangle$, m cannot have any variable coming from the MPR corresponding to e. This is a contradiction which shows that the intersection of E_1 and E_2 is empty, or

$$E_1 \cap E_2 = \emptyset$$

Otherwise, if two products of coset classes have the same set of MPRs $\{e_1, \ldots, e_k\}$, then any monomial which has less between 1 and the minimum number of variables allowed for each MPR, is in both products. Thus,

$$E_1 \cap E_2 = \left(\prod_{i=1}^{k} \langle e_i; w_{1,i} \rangle \right) \cap \left(\prod_{i=1}^{k} \langle e_i; w_{2,i} \rangle \right) = \left(\prod_{i=1}^{k} \langle e_i; \min{(w_{1,i}, w_{2,i})} \rangle \right)$$

□

6.3.3.2 A Successful Cube Attack Implementation

In this section, we introduce the strategy of decrementing the monomial profile of a CMPR in order to efficiently search for cube candidates. When decrementing, for each product of pairs T in the monomial profile and for each pair $\langle e; w \rangle$ in T, we include the similar product T', which contains $\langle e; w-1 \rangle$ in place of $\langle e; w \rangle$. If $w = 1$, then the pair is omitted from T'. Lastly, if there exists a second product $\hat{T}$ in the monomial profile such that $\hat{T} \neq T$ and $\hat{T} \cap T' = T'$, then we do not include T' in the set of decremented profiles.

Example 13. Consider the the monomial profile used in Example 12:

$$\langle 7; 4 \rangle \langle 5; 2 \rangle + \langle 3; 2 \rangle \langle 2; 1 \rangle$$

Decrementing yields the possible monomial profiles $\langle 7; 3 \rangle \langle 5; 2 \rangle$, $\langle 7; 4 \rangle \langle 5; 1 \rangle$, $\langle 3; 1 \rangle \langle 2; 1 \rangle$, or $\langle 3; 2 \rangle$. In the case of a CMPR, this would suggest that any cube would have variables corresponding to these resulting profiles:

- 3 variables from the 7-bit MPR multiplied with 2 variables from the 5-bit MPR
- 4 variables from the 7-bit MPR multiplied with 1 variable from the 5-bit MPR
- 1 variable from the 3-bit MPR multiplied with 1 variable from the 2-bit MPR
- 2 variables from the 3-bit MPR

The key insight of our cube attack strategy is that for any monomial which has variables exactly according to one of these decremented profiles has a superpoly with degree at most one. This can be observed from the fact that if the superpoly were of degree greater than one, then by the equation in Section 6.3.3, there would be a term in $p(x[0], \ldots, x[n-1])$ with more variables than allowed by the monomial profile. If we assume that every monomial allowed by the monomial profile appears in $p(x[0], \ldots, x[n-1])$ with probability

$\frac{1}{2}$ (which is in line with our empirical observations in Section 6.3.2), then this means that any monomial chosen according to the decremented profiles is highly likely to be a cube. In fact, this decrementing strategy can be seen as a generalization of the idea of $d-$random polynomials in [DS09].

Returning to our 17-bit CMPR from Section 6.1, the monomial profile is more complicated, but can be computed using Algorithm 1, which yields the monomial profile

$$\langle 2;1\rangle + \langle 3;1\rangle + \langle 7;7\rangle\langle 5;2\rangle\langle 3;1\rangle + \langle 7;3\rangle\langle 5;3\rangle\langle 3;1\rangle + \langle 7;7\rangle\langle 3;1\rangle + \langle 7;1\rangle\langle 5;1\rangle\langle 3;2\rangle$$
$$+ \langle 7;5\rangle\langle 3;2\rangle + \langle 5;1\rangle + \langle 7;7\rangle, \langle 5;4\rangle + \langle 7;5\rangle\langle 5;5\rangle + \langle 7;7\rangle$$

Decrementing this monomial profile yields the cube candidates in Table 11.

Table 11: Cube Profile Summary

Cube Profile	**Status**
<5;1><3;2>	Valid Cube
<7;2><5;3><3;1>	Cube Requires Key Bits
<7;4><3;2>	Cube Requires Key Bits
<7;6><5;2><3;1>	Cube Requires Key Bits
<7;7><5;1><3;1>	Cube Requires Key Bits
<7;4><5;5>	Cube Requires Key Bits
<7;6><5;4>	Cube Requires Key Bits
<7;7><5;3>	Cube Requires Key Bits

The monomial profiling in Table 11 was used for cube selection for our cube attack of the 17-bit CMPR. Due to saturation, all monomial profiles eventually become identical after a certain point. Moreover, cubes that require key bits, or bits of the CMPR that are initialized with the nontweakable variables, cannot be used for the cube attack. The generic cube attack model only allows for tweaking public variables such as an IV. As a result, only one cube was found. The cube used was $\{x_2, x_3, x_5\}$, where x_2 and x_3 (bits 2 and 3) both belong to the 3-bit MPR, and x_5 represents the least significant bit of the 5-bit MPR.

6.3.3.3 Modified CMPR Construction

To resolve the vulnerability found with the monomial profile-based cube attack, we present a modified construction of the CMPR, where each block M_i chains only from the previous block M_{i-1} (see Definition 22 for more details). Thus, we can restrict the monomial expression (ME) to the following representation:

$$\text{ME}_{i+1} = (\text{ME}_i)^d + \langle s_{i+1}; 1\rangle$$

where d is the degree of chaining, and $\langle s_{i+1}; 1\rangle$ introduces one variable from the $(i+1)$-th MPR.

We also introduce a concept of degree redistribution to describe how terms from earlier MPRs can trade degrees with terms in a current MPR block.

Definition 34. Degree Redistribution: Consider two consecutive blocks M_i and M_{i+1} with respective monomial profiles $\langle s_i; d_i\rangle$ and $\langle s_{i+1}; d_{i+1}\rangle$. The degree redistribution property implies that for each term $\langle s_{i+1}; d_{i+1}\rangle$ in the profile of M_{i+1}, it is possible to

express that term as a combination of variables from the earlier block M_i. Specifically, reducing the number of variables selected from the $(i+1)$-th block by 1 (i.e., decrementing d_{i+1} by 1) can be compensated by selecting additional variables from the i-th block. Formally, the total degree constraint for such terms can be expressed as follows:

$$d_i + d_{i+1} = D$$

where D is the total degree assigned to the combined expression involving both M_i and M_{i+1}.

For any block M_i, redistributing the degree between blocks means that reducing the number of selected variables from the later block M_{i+1} results in an increase in the number of selected variables from the earlier block M_i, ensuring that the total degree remains constant.

Using the concept of degree distribution, we demonstrate that the proposed chaining structure in this section, where M_i only receives chaining functions from the prior MPR M_{i-1}, is a sufficient condition for preventing cube attacks.

We now analyze the root expressions for each block, in order to verify that the root expression for each MPR (excluding the largest MPR) contains nonlinear terms even when the cube attack from Subsection 6.3.3.2 is applied. Each root expression RE_i is built recursively by combining terms from the previous root expression RE_{i-1} raised to a power d, along with a new term from the current block:

$$\mathrm{RE}_i = (\mathrm{RE}_{i-1})^d + \langle s_i; 1\rangle$$

This chaining ensures that every new block's root expression depends on the degree d and previous terms. This creates a structure where degree redistribution can occur across blocks. With this redistribution, decrementing a term in a monomial profile (e.g., reducing d_{i+1} in $\langle s_{i+1}; d_{i+1}\rangle$) results in the introduction of terms with additional variables from earlier blocks (e.g., an increase in d_i in $\langle s_i; d_i\rangle$). This process leads to the emergence of nonlinear terms in the resulting expression, even when attempting to decrement variables from later blocks which prevents the cube attack from simplifying the overall expression to a linear or constant form (maxterm).

Given this property of redistribution, there are two cases to consider when the attacker performs a cube attack (while exploiting the monomial profile) and decrements terms in the monomial profile to isolate cube candidates. Consider a monomial profile of the form:

$$\mathrm{MP}_i = \langle s_1; d_1\rangle\langle s_2; d_2\rangle \dots \langle s_i; d_i\rangle$$

where $\langle s_j; d_j\rangle$ represents d_j variables selected from the s_j-bit register and i denotes the number of MPRs that appear in the monomial profile.

Suppose the attacker attempts to decrement $\langle s_j; d_j\rangle$ for some $j < i$. By degree redistribution, the resulting monomial profile still contains nonlinear terms, preventing the cube attack from reducing the polynomial to a linear or constant form and formation of maxterms.

Otherwise, consider the reduction of the monomial profile $\langle s_i; d_i\rangle$ for decrementing to exploit the monomial profile computed during the offline phase, such an attack is constrained by the fact that, in our PRNG design, the top block corresponds to the key bits, which cannot be flipped during the cube attack. Therefore, attempts to decrement $\langle s_i; d_i\rangle$ are not feasible, as the attacker lacks control over these key bits.

Chaining each block in the CMPR system from only the previous block is hence a sufficient condition for preventing cube attacks, and formalizes a strong construction for the CMPR.

6.3.3.4 Key-Independent Distinguishers and Cube Testers

Key-independent distinguisher attacks distinguish the output of a cryptographic scheme from random data, under the assumption that the distinguishing attack can be performed without knowledge of the key and only knowledge of public variables, such as an IV, can be modified [COOP23]. To analyze the susceptibility of CMPR-based designs to key-independent distinguishers, we refer to cube testers [ADMS09], which apply a similar methodology to cube attacks but aim to detect nonrandom behavior in the output of a cryptographic scheme by tweaking public variables and holding the key constant. Cube testers, which we will also refer to as cube distinguishers, are highly applicable to cryptographic schemes where the input-output relation can be represented by polynomials of known degree, which is the case for CMPRs, since the polynomial representation of a CMPR is captured by its monomial profile. The main observation we use when applying cube testers to CMPR-based designs is that for a tweakable black box polynomial (in this case, the polynomial relation between the initial CMPR state and output, which we will call the output polynomial) of degree at most d in the tweakable variables, evaluating the polynomial over any cube of size $d+1$ or greater yields the zero polynomial, meaning the computational cost of a cube tester is 2^{d+1}.

Generally, to find the lowest-cost cube tester that can be applied to a CMPR-based design, we must determine which tweakable variables in the output polynomial have the lowest degree d, which depends heavily on the specifics of the design, namely the number of constituent MPRs and the number of AND gate inputs to the chaining functions. In order for a cube distinguisher to be infeasible, $d+1$ must be large enough so that it is infeasible for an attacker to perform the required 2^{d+1} computations. That is, if a CMPR-based design is to to resist cube testers, the tweakable variables in the output polynomial must be of sufficiently high degree such that it is infeasible to perform 2^{d+1} computations.

6.3.3.5 A Successful Cube Tester Implementation

Consider the case of our PRNG in Figure 8, where the output is extracted from the least significant bit of the 2-bit MPR. Let the 7-bit MPR be initialized with nontweakable variables, and the 5 and 3-bit MPRs be initialized with tweakable variables. The 2-bit MPR is also initialized with nontweakable variables. So, the internal state includes 8 tweakable bits, initially stored in the 5- and 3-bit MPRs. We will also assume that the modified CMPR construction from Subsection 6.3.3.3 is being used; thus, the chaining functions in the CMPR construction can only connect from one MPR to the next, and a chaining function from a given MPR cannot connect to multiple MPRs.

The tweakable variables in the 3-bit MPR only go through the chaining functions between the 3-bit MPR and 2-bit MPR, and these chaining functions use AND gates with at most 4 inputs. Thus, we can initially state that the chaining functions are of at most degree 4. However, due the use of a 3-bit MPR as the 2nd-to-last MPR in the construction, this PRNG is a special case where an optimization can be made: when designing the chaining functions between the 3- and 2-bit MPRs, we can only use the 3 state variables from the 3-bit MPR as AND gate inputs, since we are using the modified CMPR construction. If we can only select from 3 state variables for the AND gate inputs, then the actual degree of the chaining functions between the 3- and 2-bit MPRs is 3. In the output polynomial, which is obtained from the least-significant bit of the 2-bit MPR, the tweakable variables from the 3-bit MPR have at most degree 3. We have found that $d=3$, so the cube tester requires $2^{d+1}=16$ computations. For the computations, the 8-bit vector of tweakable inputs to the PRNG is constructed such that the input varies in the 4 least significant bits, taking on all possible values of 4 bits. The rest of the input vector remains constant. For the 16 inputs, we obtain 16 outputs of equal length. Subsequently, we take the XOR of all 16 outputs, which is equivalent to summing over a $4-$dimensional cube. Thus, the cube tester has been successfully implemented, and the PRNG outputs

have been distinguished from true random values.

6.3.3.6 Resisting Cube Testers

In the case where the output is extracted from the smallest MPR, resisting cube testers requires that in the output polynomial, the degree d of the variables from the last tweakable MPR in the construction must be large enough such that it is infeasible for an attacker to perform 2^{d+1} computations. For example, consider a CMPR designed using our modified CMPR construction from Subsection 6.3.3.3, where the chaining functions are only from one MPR to the next. In this section, the degree of the chaining (maximum number of AND gate inputs) is k and the number of nontweakable MPRs between the last tweakable MPR and the output extraction MPR is n. The variables from the last tweakable MPR pass through $n+1$ stages of $k^{th}-$degree chaining functions, with each stage contributing a multiplying factor of k to the degree of these variables in the output polynomial. Then, the degree d of the tweakable variables in the output polynomial is $d = k^{n+1}$. Thus, the cube tester requires at least $2^{d+1} = 2^{k^{n+1}+1}$ computations. This number grows rapidly with respect to n. For example, if $k = 4$ (as in the case of our PRNG design), then $n = 2$ additional fixed MPRs results in over 2^{64} computations and $n = 3$ additional fixed MPRs results in over 2^{256} computations being required for the cube tester.

Generally, the degree of the tweakable variables from the last tweakable MPR in the output polynomial can be increased by (i) including additional nontweakable MPRs (MPRs initialized to constant values that are not tweakable by an attacker) between the last tweakable MPR and the MPR from which the output is extracted or (ii) permuting the state of the CMPR during initialization (for example, swapping the upper and lower halves of the state halfway through the initialization rounds). Solutions (i) and (ii) both ensure that all bits in the initial state, including tweakable variables, pass through several stages of chaining functions between MPRs, resulting in the output polynomial terms that include tweakable variables having a sufficiently high degree to resist cube distinguishers. However, applying solution (i) requires paying careful attention to the Mersenne exponents used. For example, when including additional nontweakable MPRs, it is important that the Mersenne exponents of the additional MPRs be greater than the number of AND gate inputs to the chaining functions, so that the optimization made in Subsection 6.3.3.5 cannot be applied. In addition, we note that applying solution (ii) means that the assumption that state bits are not permuted as explained in Subsection 6.1.2 is no longer true; however, this is not a problem at all as permuting the internal state with a single swap only enhances cryptographic properties such as confusion and diffusion.

From a design perspective, solutions (i) and (ii) both incur additional hardware implementation cost. The hardware implementation cost of solution (i) depends largely on the Mersenne exponents used in the CMPR before the solution is applied. These Mersenne exponents will limit the new MPRs that can be added. For example, if adding MPRs to resist cube testers, one must be careful when using MPRs of identical size in a CMPR construction, since our theory in Subsection 3 assumes coprimality of the MPR periods for analyzing the period of a CMPR. Solution (i) also results in additional latency, since the size of the CMPR increases and more time is needed to clock a larger register. However, solution (ii) is more efficient to implement in hardware as no new MPRs or chaining functions are needed. In our implementations, only one clock cycle of latency is introduced, with the one additional clock cycle being used to permute the internal state of the CMPR. In general, however, we believe that it is possible to implement solution (ii) without introducing a clock cycle of latency by applying optimization techniques such as implementing a custom circular shift function in the hardware description code for the CMPR.

6.3.4 Linear Complexity Analysis Algorithms

The Berlekamp-Massey Algorithm [Mas69] finds the smallest LFSR that could produce the pseudorandom output of the CMPR. This would be considered an attack if it could be done faster than brute force. As mentioned in Subsection 6.3.4, the linear complexity of our 17-bit CMPR PRNG output was empirically determined to be 12,387. Based on the algorithm runtime in conjunction with the huge number of output bits needed to implement it, we believe that the Berlekamp-Massey Algorithm is not considered an attack unless the construction is sparse and of low degree. Otherwise, if the chaining functions use degree 4 and chaining functions are applied to nearly every state bit (as is the case in our PRNG design), the linear complexity of the CMPR output quickly grows as the CMPR is clocked and output bits are generated.
The Estimation Algorithm described in Section 5 should also not be viewed as an attack, as it does not necessarily find the specific LFSR recurrence like the Berlekamp-Massey algorithm, but rather finds the degrees of possible characteristic polynomials. We believe this information is of little use to an attacker when the CMPR avoids sparse or low-degree constructions.

6.3.5 Linear Cryptanalysis

Presented in [Mat94] by Matsui, linear cryptanalysis tries to obtain a linear approximation for a given cryptographic algorithm. Let P_i denote plaintext bits at index i, C_j denote pseudorandom output bits at index j, and K_l denote seed bits at register l. These linear approximate equations are in the form:

$$P_1 \oplus P_2 \oplus \dots C_1 \oplus C_2 \oplus \dots = K_1 \oplus K_2 \dots$$

One method of generating linearized equations is to reuse the equations obtained from the algebraic attack in Section 6.3.2 and remove nonlinear terms. For example, we could linearly approximate the first and second equations for the least-significant bit our 17-bit CMPR construction as follows:

$$C_0(1) = 1 \oplus C_1(0) \oplus C_6(0) \oplus C_0(0)$$
$$C_0(2) = C_3(0) \oplus C_2(0) \oplus C_5(0) \oplus C_7(0) \oplus C_1(0)$$

Figure 13 shows the probability that the linearized equation is accurate with respect to clock cycles. Although this strategy gives equations that hold with high probability for the first couple of clock cycles, after a few additional initialization rounds, the probabilities are approximately random. We were unable to derive any other strategy for finding meaningful linearizations. Thus, we were unable to derive a meaningful attack using this approach.

6.3.6 Cryptanalysis Conclusion

After modifying the chaining structure of our CMPR construction and adding a state swap in order to resist cube attacks and cube testers, the offensive team was unable to find an attack on the 17-bit PRNG with everything known but the initial state of the CMPR. Nonetheless, this PRNG could be made more cryptographically secure by employing several different techniques, including increasing the size of the CMPR, withholding information about the CMPR specifications such as the chaining functions, using reconfigurable logic to make the update polynomials variable increasing the search space by size proportional to the order of the field, and introducing additional initialization rounds before the pseudorandom output is generated.

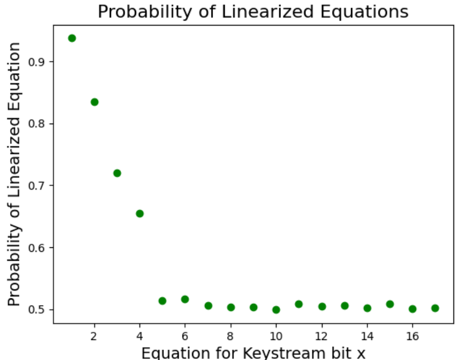


Figure 13: Probability of Linear Approximate Equations Holding with Respect to Initialization Rounds

7 Stream Cipher Design and Comparison

In this section, we propose a new class of CMPR-based stream ciphers and provide detailed comparisons with TRIVIUM [De 06; DP08]. Due to being a lightweight, NLFSR-based stream cipher, we consider TRIVIUM to be a viable target for comparison with our new CMPR-based stream ciphers.

We begin in Subsection 7.1 by proposing three CMPR-based stream ciphers. Then, in Subsection 7.2, we continue with a discussion of the design choices involved in the design of CMPR-based stream ciphers. Next, in Subsection 7.3 we discuss the security analysis and statistical testing of the CMPR stream ciphers including appropriate comparisons with TRIVIUM. Hardware implementations and stream cipher comparisons occur in Subsection 7.4, including ASIC and FPGA synthesis results for both TRIVIUM and the CMPR stream ciphers. Finally, we conclude this section with Subsection 7.5 where we summarize our comparison and discuss the advantages of CMPRs in stream cipher design.

7.1 Three CMPR-Based Stream Ciphers

In this section we describe three CMPR-based stream ciphers we have designed and which we claim exemplify the possible tradeoffs between hardware implementation area and security margins. These stream ciphers are specified and named in Table 12 to simplify any references to them throughout the remainder of this section. For reference, the TRIVIUM stream cipher is included in the same table.

Table 12: Specifications for TRIVIUM and the Family of CMPR-Based Stream Ciphers

Cipher Name	Internal State Size and Type	Key Size	IV Size
TRIVIUM	288 bits (NLFSR)	80 bits	80 bits
CMPR stream cipher v_1	288 bits (CMPR)	128 bits	128 bits
CMPR stream cipher v_2	162 bits (CMPR)	80 bits	80 bits
CMPR stream cipher v_3	170 bits (CMPR)	84 bits	84 bits

For each CMPR-based stream cipher, the MPRs, update polynomials, and primitive polynomials used to construct the CMPRs are shown in Tables 13, 14, and 15. Architecture diagrams for each CMPR are also included. For all CMPRs, the chaining functions are 2-, 3- or 4-input AND gates cascaded with 2-, 3- or 4-input XOR gates, with the restriction that a chaining function can only contain input bits from the MPR immediately before the MPR with which the chaining function is being combined (by use of XOR). In Figures 14,

15, and 16, the C_i represent chaining functions.

Table 13: CMPR Specification for CMPR stream cipher v_1

MPR Size	Update Polynomial	Primitive Polynomial
107 bits	$U_{107}(x) = x$	$P_{107}(x) = x^{107} + x^{89} + x^{84} + x^{40} + x^{29} + x^{23} + 1$
89 bits	$U_{89}(x) = x$	$P_{89}(x) = x^{89} + x^{81} + x^{68} + x^{31} + x^{21} + x^{18} + 1$
61 bits	$U_{61}(x) = x$	$P_{61}(x) = x^{61} + x^{44} + x^{19} + x^{15} + 1$
31 bits	$U_{31}(x) = x$	$P_{31}(x) = x^{31} + x^3 + x^2 + x + 1$

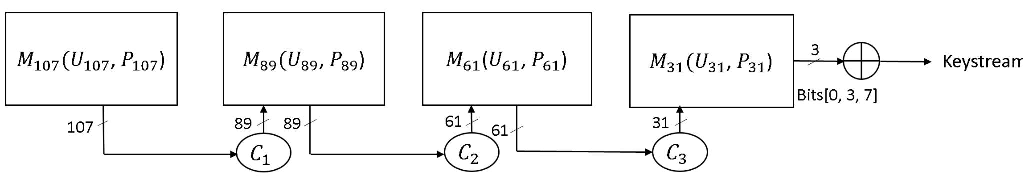


Figure 14: Architecture Diagram for CMPR stream cipher v_1

Table 14: CMPR Specification for CMPR stream cipher v_2

MPR Size	Update Polynomial	Primitive Polynomial
107 bits	$U_{107}(x) = x$	$P_{107}(x) = x^{107} + x^{89} + x^{84} + x^{40} + x^{29} + x^{23} + 1$
31 bits	$U_{31}(x) = x$	$P_{31}(x) = x^{31} + x^3 + x^2 + x + 1$
17 bits	$U_{17}(x) = x$	$P_{17}(x) = x^{17} + x^8 + x^7 + x^6 + x^4 + x^3 + 1$
5 bits	$U_5(x) = x$	$P_5(x) = x^5 + x^4 + x^3 + x^2 + 1$
2 bits	$U_2(x) = x$	$P_2(x) = x^2 + x + 1$

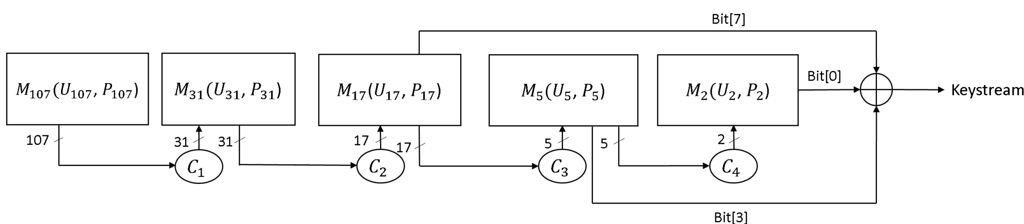


Figure 15: Architecture Diagram for CMPR stream cipher v_2

Table 15: CMPR Specification for CMPR stream cipher v_3

MPR Size	Update Polynomial	Primitive Polynomial
127 bits	$U_{127}(x) = x$	$P_{127}(x) = x^{127} + x^{54} + x^{45} + x^{13} + 1$
19 bits	$U_{19}(x) = x$	$P_{19}(x) = x^{19} + x^5 + x^4 + x^3 + x^2 + x + 1$
17 bits	$U_{17}(x) = x$	$P_{17}(x) = x^{17} + x^8 + x^7 + x^6 + x^4 + x^3 + 1$
5 bits	$U_5(x) = x$	$P_5(x) = x^5 + x^4 + x^3 + x^2 + 1$
2 bits	$U_2(x) = x$	$P_2(x) = x^2 + x + 1$

Each CMPR-based stream cipher operates according to Algorithm 2. The Algorithm consists of two phases. Phase (i) is the initialization phase, during which the CMPR

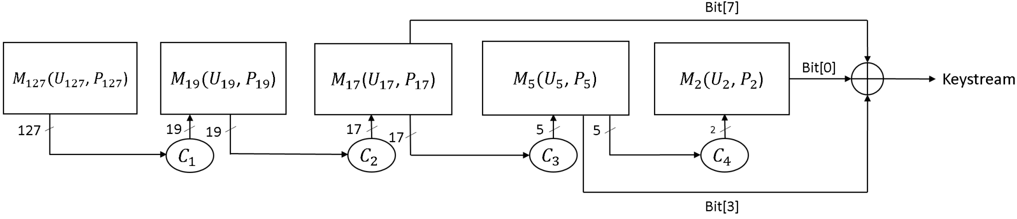


Figure 16: Architecture Diagram for CMPR stream cipher v_3

Algorithm 2 CMPR-Based Stream Cipher Algorithm

inputs: An $m-$bit key K, $m-$bit IV IV, and $n-$bit plaintext message PT
output: $n-$bit ciphertext c
The subscript i after a variable represents indexing the variable at the ith bit

1: **procedure** INITIALIZE
2: $CMPR$ is the $j-$bit CMPR used in the stream cipher
3: $nextstate()$ clocks the CMPR and advances to the next state
4: $swap()$ swaps the upper (most significant) and lower (least significant) halves of the CMPR state (assuming the CMPR state is of even size), preserving the original bit ordering of the upper and lower halves
5: $ones()$ represents a list of 1's
6: z is the $n-$bit keystream used for encryption
7: $(CMPR_{j-1}, \ldots, CMPR_{j-m}) \leftarrow (K_{m-1}, \ldots, K_0)$ ▷ Initialize the most significant bits of $CMPR$ with the key
8: $(CMPR_{j-m-1}, \ldots, CMPR_{j-2m}) \leftarrow (IV_{m-1}, \ldots, IV_0)$ ▷ Initialize the next most significant bits of $CMPR$ with the IV
9: $(CMPR_{j-2m-1}, \ldots, CMPR_0) \leftarrow ones()$ ▷ Initialize the remaining bits of $CMPR$ with ones
10: **for** $i \leftarrow 0 \ldots 50$ **do**
11: $CMPR.nextstate()$ ▷ Clock the CMPR
12: **end for**
13: $CMPR.swap()$ ▷ Swap the upper and lower halves of the CMPR state
14: **for** $i \leftarrow 0 \ldots 50$ **do**
15: $CMPR.nextstate()$ ▷ Clock the CMPR
16: **end for**
17: **end procedure**
18: **procedure** GENERATEKEYSTREAM
19: **for** $i \leftarrow 1 \ldots n$ **do**
20: $z_i \leftarrow CMPR_0 \oplus CMPR_3 \oplus CMPR_7$ ▷ Generate keystream bit by XORing three internal state bits
21: $CMPR.nextstate()$ ▷ Clock the CMPR
22: **end for**
23: $z \leftarrow (z_1, \ldots, z_n)$
24: **end procedure**
25: $c \leftarrow PT \oplus z$ ▷ Compute ciphertext using XOR of plaintext and keystream

stream cipher is initialized with the key and IV and is clocked 100 times, where clocking the CMPR is considered an initialization round. The upper and lower halves of the internal state are swapped midway through the 100 initialization rounds, and the overall goal of the initialization phase is to to randomize the internal state before the keystream generation phase. Phase (ii) is the keystream generation phase, during which the keystream bit is

derived by XORing the three internal state bits 0, 3, and 7.

In Algorithm 2, the number of initialization rounds is always 100, and state bits 0, 3, and 7 are always used to generate the keystream. CMPR stream ciphers v_1, v_2, and v_3 all operate according to Algorithm 2.

7.2 Design Rationale

In the previous subsection, Subsection 7.1, three specific CMPR stream ciphers were introduced, but in fact we believe that a large class (or "family") of CMPR stream ciphers can be specified. In designing this family of CMPR-based stream ciphers, we formulated and adhered to several important design principles. Some of these design principles are generally accepted practices in stream cipher design, whereas others are unique to CMPRs and were discovered throughout the design and testing process. The latter class of principles is the primary focus of this section since violating CMPR-specific design principles can result in the entire construction being vulnerable to cryptanalytic attacks or poorly performing hardware implementations.

Some generic stream cipher design principles that are reflected in our family of CMPR-based stream ciphers are the use of a balanced boolean function (a boolean function with an equal amount of 0s and 1s in its truth table), specifically the XOR operation, when deriving the keystream bit from the internal state of the cipher, the use of initialization rounds prior to keystream generation, and the use of an XOR to combine the keystream with plaintext for encryption [PP97].

7.2.1 Keystream Generation, Initialization, and Key Input

In this subsection we explore keystream generation, the initialization phase of the ciphers and how to input the key. We assume that the number of bits in the CMPR is chosen to be large enough to accommodate both the key and the IV in separate parts of the CMPR during cipher initialization.

Keystream Generation: We extract the keystream from an XOR of bits from the lowest-order MPRs. The use of the smallest MPRs ensures that the keystream is being generated from the portion of the CMPR state that evolves in the most nonlinear manner. The lowest-order MPRs are situated at the very end of the cascade of chained MPRs and are therefore the smallest MPRs in a CMPR. As a CMPR is clocked, the bits in the smallest MPRs evolve under the influence of all prior nonlinear chaining functions in the CMPR, and thereby have desirable statistical properties.

We empirically made the choice to generate the keystream by performing an XOR between three internal state bits. This choice ensures that the keystream is a function of several bits of the internal state and also ensures that the keystream is balanced (due to XOR being a balanced operation). For the CMPR-based stream ciphers synthesized in our investigation, the three state bits $0, 3$, and 7 were XORed to generate the keystream. TRIVIUM uses 6 internal state bits to generate its keystream. Our empirical decision to use three internal state bits was based on the fact that each bit of the CMPR already depends on the state of the higher-order MPRs due to chaining, although we stress the importance of the keystream generation bits coming from the lowest-order MPRs.

Initialization Rounds: In Section 6.1.1, it was empirically observed during statistical testing of our CMPR-based stream ciphers that a number of initialization rounds equal to two times the number of MPRs used to construct the CMPR obscured the statistical relationship between the initial state of the CMPR and its output. While this observation provides a good lower bound for initialization rounds, in practice, we choose to use 100 initialization rounds.

Our choice of 100 initialization rounds is motivated by the behavior of CMPRs in the context of monomial counts and degrees. Figure 17 compares the degree and count

of monomials in the polynomial representation of the least-significant bit of the 17-bit CMPR of Section 6 alongside the polynomial representation of the TRIVIUM output as a function of initialization rounds, across 320 initialization rounds. In this comparison, we used the 17-bit CMPR because precise monomial analysis of larger CMPRs quickly becomes computationally infeasible.

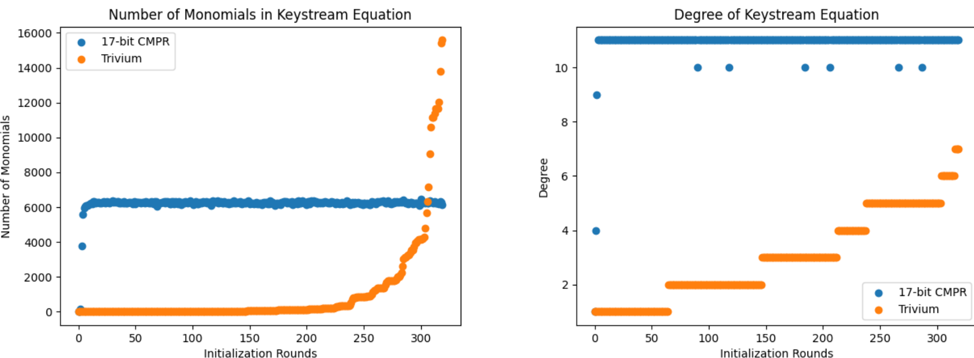


Figure 17: Monomial Count and Degree Comparison Between TRIVIUM and 17-bit CMPR

For a CMPR, the degree and number of monomials quickly saturates, as opposed to TRIVIUM where the degree and number of monomials grow more slowly. This rapid growth of monomial count and degree can be explained by the aggregated effect of the chaining functions. Every initialization round, the CMPR is clocked and the chaining functions between MPRs add monomials to the MPRs that are being chained into. Since the chaining functions of our designs apply to every state bit that can be chained into (all state bits excluding those in the largest MPR), many monomials are added to the MPRs that receive chaining functions. Moreover, across initialization rounds, the degree of the chaining functions influences the degree of the monomials. Since we use degree 4 chaining functions (4 inputs to the AND gates of the chaining functions), we observe that the monomials also saturate quickly with respect to degree. The tendency of the polynomial representation of the least-significant bit of the CMPR to quickly saturate with respect to monomial count and degree showcases why we can comfortably employ 100 initialization rounds.

Swapping During Initialization: In Section 6.3.3.4, we concluded that permuting the internal state of a CMPR (e.g., by swapping the upper and lower halves) provides resistance against cube distinguishers. For example, consider the scenario where the internal state of our CMPR-based stream ciphers is not permuted. To find the lowest-cost cube tester that can be applied to the ciphers, we must determine which of the tweakable variables (in the case of stream ciphers, the IV bits) in the keystream polynomial have the lowest degree. CMPR stream ciphers v_1, v_2, and v_3 are shown in 14, 15, and 15 and operate according to Algorithm 2. For these stream ciphers, we observe that the second-to-last MPR is initialized with the least significant bits of the IV; thus, the second-to-last MPR includes public variables that can be tweaked when applying cube testers. The IV bits in the second-to-last MPR only pass through the chaining functions connecting the second-to-last and last MPRs. As these chaining functions use AND gates with at most 4 inputs, we conclude that in the keystream polynomial, the terms including IV bits from the second-to-last MPR are at most degree 4. Thus, we have $d = 4$, so the lowest-cost cube tester would require $2^{4+1} = 32$ computations using a fixed key and IVs that vary in the 5 least significant bits. That is, the 32 IVs must take on all possible 5-bit values in their 5 least significant bits, while the rest of the IV remains constant. In order to provide resistance against this cube tester, we swap the upper and lower halves of the CMPR state halfway

through the 100 initialization rounds. That is, the CMPR is clocked 50 times, the internal state is swapped, and the CMPR is clocked another 50 times. When performing the swap, we preserve the original bit ordering of each half of the state. The final results is that all of the key and IV variables in the initial state will eventually propagate through multiple stages of chaining functions and thereby have higher degree in the keystream polynomial.
Key Input: Initializing the most significant bits of the CMPR with the secret key ensures that the key bits immediately propagate through as many of the nonlinear chaining functions as possible in the design as the CMPR is clocked 100 times. As a CMPR is clocked, the bits that are initially in the largest MPRs (the most significant bits) are fed through chaining functions that connect to smaller MPRs. The chaining functions feed forward from larger to smaller MPRs. Thus, the key bits initially stored in the most significant bits will propagate through nonlinear chaining functions between the MPRs. This design choice further obscures any relation between the secret key and the keystream since the keystream is extracted from the smallest MPRs. We place the IV in the next highest order bits in the CMPR after the key bits to ensure that the IV bits also propagate through nonlinear chaining functions, although we prioritize initializing the most significant bits of the CMPR with the key.

One limitation discussed in Subsection 6.2.2 concerns adversary control of the smallest MPR in a CMPR-based design. If the adversary can control the initial state of the lowest-order MPR, e.g., through choosing the IV, then non-uniform statistics may emerge; in summary, the bit-contribution tests on the smallest MPR indicate a problem. Therefore, as suggested in Subsection 6.1, we always set the initial state of the lowest order MPR to all ones. For example, if the smallest MPR in a CMPR stream cipher has size 31, then the last 31 bits are always initially set to all ones and hence cannot receive a key or an IV. Similarly, if the lowest-order MPR is a two-bit MPR, then the last two bits are always set to one and cannot receive key bits or IV bits.

7.2.2 MPR Selection

The selection of MPRs to construct a CMPR involves several design options, including Mersenne exponents (the size of each MPR), update polynomials, and primitive polynomials. There are several tradeoffs to consider, as the size, update polynomials, and primitive polynomials of the MPRs have a significant impact on linear complexity, expected period ratio, hardware area, and energy consumption.
Mersenne Exponent Selection: When selecting Mersenne exponents for the MPRs that are chained together to form a CMPR and considering how many MPRs are used to construct a CMPR, we look for a tradeoff among the linear complexity, the expected period ratio of the resulting CMPR, and the hardware area of the construction. For two CMPRs of the same size, the CMPR that is constructed using more MPRs will have higher linear complexity, following the results of Section 5.3. However, it is important to remain cognizant of the number of MPRs used because there are chaining functions between MPRs, and the chaining functions contribute to increased hardware area and energy consumption. It is also important not to use very few MPRs (for example, a two-MPR CMPR construction) because the resulting design will likely be vulnerable to cryptanalytic attacks (see Subsection 6.3.3). For CMPRs with less than 300 register bits, we prefer constructing a CMPR using 4 to 6 MPRs, which provides a reasonable middle ground between increased usage of chaining (and hence increased linear complexity) and increases in hardware area.
Small MPRs: The presence of small MPRs also presents an interesting tradeoff. The presence of small MPRs (such as 2-, 3-, or 5-bit MPRs) in a CMPR construction increases the linear complexity of the construction, and small MPRs contribute relatively little hardware area. However, the use of small MPRs in a CMPR construction also decreases the expected period ratio of the construction, as demonstrated in Table 3 from the very end

of Section 4. Thus, appending a small MPR to a CMPR is a good strategy for bolstering linear complexity for little hardware cost, at the expense of a lower expected period ratio. Additionally, following the statistical testing results of Section 6.2, we determined that flipping a bit in the 2-bit MPR used in the PRNG resulted in a predictable change in the PRNG output. Therefore, in CMPR stream ciphers v_2 and v_3, both of which use 2-bit MPRs, we initialize the contents of the 2-bit MPRs to 1s so that these bits cannot be altered during key and IV initialization. In the case of CMPR stream cipher v_1, the 256 most significant bits of the 288-bit internal state are initialized with the 128−bit key and 128−bit IV, and the 32 least-significant bits of the state are initialized to all 1's. So for cipher v_1, there was no need to address the statistical vulnerability discovered in Section 6.2.

To illustrate how a larger number of constituent MPRs and the presence of small MPRs increase linear complexity, we applied Algorithm 1 to the CMPRs used in CMPR stream ciphers v_1, v_2, and v_3. As a result, we obtained lower estimates and upper bounds for the linear complexity of each construction in Table 16.

Table 16: Linear Comlexity Lower Estimates and Upper Bounds for CMPR Stream Ciphers v_1, v_2, and v_3

Cipher	CMPR Size	Lower Estimate	Upper Bound
CMPR stream cipher v_1	288 bits	$2^{130.30}$	$2^{130.31}$
CMPR stream cipher v_2	162 bits	$2^{154.66}$	$2^{154.67}$
CMPR stream cipher v_3	170 bits	$2^{163.23}$	$2^{163.26}$

An interesting observation is that the CMPRs of ciphers v_2 and v_3, despite being over 100 bits smaller than the CMPR of cipher v_1, have higher linear complexity estimates and bounds due to using more MPRs in their respective CMPR constructions.

Update Polynomials: As showcased in Table 1, varying the update polynomial of an MPR affects the state sequence of an MPR, or more specifically, the order in which the possible states of an MPR are traversed when starting from a known initial state. For this change of state order to be possible and to implement update functions of a higher degree, usage of larger update functions requires additional XOR gates to be introduced into the hardware design. Thus, using $U(x) = x$ as the update polynomial of the MPRs lowers the hardware implementation area of the CMPR-based stream ciphers, and also means that the MPR is equivalent to a Galois LFSR.

Primitive Polynomials: In selecting the primitive polynomial of an MPR, we seek primitive polynomials with enough terms to resist correlation attacks, but few enough terms to maintain an efficient hardware implementation. Similar to update polynomials, larger primitive polynomials correspond to using more XOR gates in hardware. Empirically, we found that we prefer primitive polynomials with 5 to 7 terms in order to provide a comfortable middle ground between correlation attack resistance and implementation cost, although polynomials with even more terms may be suitable for larger MPRs. We also want to point out, as explained at the end of Subsection 4.1, that for MPRs the use of a Mersenne exponent results in the fact that every irreducible polynomial is primitive (thus simplifying the search for primitive polynomials).

7.2.3 Chaining Function Selection

When analyzing algebraic attacks in Subsection 6.3.2, we observed that chaining functions with product terms consisting of 4 variables resulted in a rapid increase in the number of variables which must be solved for in order for the attack to succeed. The number of variables in the product terms of the chaining function is analogous to the number of inputs to the AND gates used in the chaining function. Thus, the number of inputs to the AND gates of the chaining functions contributes to algebraic attack resistance,

as well as the linear complexity of the periodic state sequences generated by a CMPR. However, chaining functions also contribute to the area and energy consumption of the hardware implementation of a CMPR since each chaining function requires several logic gates when being implemented in hardware. By using 4-input AND gates, we intend to achieve reasonably lightweight hardware implementations and high linear complexity while also resisting algebraic and cube attacks. The chaining functions applied to the CMPRs used in the CMPR stream ciphers consist of AND gates with at most 4 inputs, and the outputs of the AND gates are connected to XOR gates with at most 4 inputs.

We also ensured that the chaining functions are balanced, in that the truth table of each chaining function contains an equal number of 0s and 1s. Balanced chaining is important when considering the statistical properties of the CMPR states as the CMPR is clocked. If unbalanced chaining functions are used, the states of the CMPR will be biased with 1s or 0s (depending on the particular chaining functions used). This is a statistical vulnerability that can potentially reveal information about the initial state of the CMPR.

For each CMPR used in the CMPR ciphers, the chaining functions were applied to every bit of the CMPR excluding the bits in the largest MPR. This measure of applying the maximum possible number of chaining functions is a precaution to ensure that each construction incorporates as many nonlinear chaining functions as our mathematical formulation of chaining in Section 3 allows. It is possible that fewer chaining functions can be applied in favor of increased initialization rounds; we believe that chaining function density forms a power-area tradeoff. The ANF of the CMPRs used in the stream ciphers v_1, v_2 and v_3 (which includes the specific chaining functions used) is provided in the Appendices.

The chaining functions were applied with the restriction that a chaining function can only contain bits from the MPR immediately prior to the MPR that is being chained into (following Definition 22). That is, the chaining functions only connect from one MPR to the next in descending Mersenne exponent order and cannot skip any MPRs. This is an important limit on the chaining function design space and follows from the results of our cryptanalysis of cube attacks against CMPRs in Section 6.3.3. The bits used in the chaining functions were determined by uniformly sampling the bits of the prior MPR.

7.3 Security Analysis and Statistical Comparison to TRIVIUM

We divide our security analysis into two steps, namely cryptanalysis and statistical analysis.

7.3.1 Cryptanalysis

In Section 6.3, a variety of security analyses were applied to a proof-of-concept PRNG design utilizing a 17-bit CMPR, with the assumption that the attacker possesses full knowledge of the CMPR construction including chaining functions, update polynomials, and primitive polynomials. The analyses were performed with generality and applicability to CMPR constructions as a whole in mind. This is because the attack methodologies and the overall structure of CMPRs remain consistent even as the size of a CMPR varies. We claim the same results for restricted chaining (i.e., following Definition 22), namely the results for algebraic attacks, cube attacks, and linear cryptanalysis, for our CMPR-based stream cipher designs. More specifically, for the CMPR stream cipher designs using four or more MPRs and with restricted chaining (Definition 22) and swapping, over the past three years of effort the authors of this paper have not been able to find any cryptanalytic attack technique that would have any reasonable chance of success.

We also applied cube attacks to our stream ciphers using the monomial profile cube search methodology described in Section 6.3.3. The cube attacks were ran on a high-end research server with an Intel® Xeon® E5-2699 v4 CPU running at 2.20GHz, 44 cores, and 256GB of RAM. For all of the stream ciphers, the attacks were unsuccessful.

7.3.2 Statistical Analysis of Keystreams

Strictly speaking, the statistical analyses of Section 6.2 performed on CMPR-based PRNGs do not extend automatically to the keystreams generated by our CMPR-based stream ciphers. Therefore, we apply the NIST Statistical Test Suite and the bit contribution statistical tests described in Section 6.2 to large volumes of keystreams generated by the CMPR-based stream ciphers and analyze the test results for any failures indicative of statistical weaknesses. For comparison, we also perform the same statistical analyses on keystream data from TRIVIUM. Moreover, we apply the bit contribution tests (abbreviated in Table 17 as B.C.) from Section 6.2.2 to the keystreams generated by the stream ciphers and, for comparison, also apply the bit contribution tests to keystreams generated by TRIVIUM. Subsequently, we analyze the resultant dependence matrices for any statistical relations between the key or IV and the keystream.

For each stream cipher (TRIVIUM and CMPR-based stream ciphers v_1, v_2 and v_3), a keystream dataset for the NIST Statistical Test Suite was generated from pseudorandom (key, IV) pairs. Each dataset consisted of 10 keystreams, with each keystream being (10^6) bits long, following the minimum dataset size required to apply all of the tests in the NIST Statistical Test Suite, as mentioned in Section 6.2.1. For all four stream ciphers (TRIVIUM and the three CMPR-based stream ciphers), the datasets passed all 15 of the tests in the NIST Statistical Test Suite. This result means that the keystreams generated by TRIVIUM and CMPR-based stream ciphers are pseudorandom and suitable for applications requiring pseudorandom numbers, at least from the perspective of the NIST Statistical Test Suite. It is important to note that the NIST Statistical Test Suite does not address whatsoever the security of the underlying design used to generate the test set, but rather whether the tested data on its own can be considered pseudorandom.

We also generated datasets to which the bit contribution tests (abbreviated in Table 17 as B.C.) from Section 6.2.2 were applied. For each stream cipher, the bit contribution for key and bit contribution for IV tests were applied. For each test, the number of trials was set to 10,000, and subsequently the dependence matrix was analyzed to determine any statistical anomalies between the inputs (keys and IVs) and outputs (keystreams). The results from Section 6.2.2, including the histograms shown in Figures 9, 10 and 11, are identical. Thus, all results are the same for our CMPR-based stream cipher keystreams as was found earlier for CMPR-based PRNGs. Table 17 summarizes this result.

Table 17: Statistical Comparison Between TRIVIUM and CMPR-based Stream Ciphers

Cipher Name	Passes NIST Tests?	Passes B.C. Statistical Tests?
TRIVIUM	Yes	Yes
CMPR stream cipher v_1	Yes	Yes
CMPR stream cipher v_2	Yes	Yes
CMPR stream cipher v_3	Yes	Yes

7.4 Hardware Implementations and Comparison to TRIVIUM

Since CMPRs are formed by an interconnected cascade of MPRs (and MPRs are a form of feedback register) and TRIVIUM is a hardware-oriented and feedback-shift-register-based stream cipher, we believe that TRIVIUM is a useful choice for comparison with CMPR-based stream ciphers. In this section, we explore some of the tradeoffs between register size, security, and hardware implementations.

Each CMPR stream cipher in Table 12 is designed to outperform TRIVIUM in one dimension, while remaining roughly equal in the others. For example, CMPR stream cipher v_1 uses the same internal state size as TRIVIUM, while increasing the key and IV lengths to 128 bits and thus increasing the level of security at the expense of increased

hardware area and energy consumption. CMPR stream cipher v_2 uses the same key and IV lengths as TRIVIUM and thus has the same level of security, but decreases the CMPR size with the resulting benefit of lower hardware area and energy consumption. CMPR stream cipher v_3 slightly increases the key and IV lengths to 84 bits but is designed with the intention of maintaining energy consumption approximately equivalent to TRIVIUM.

7.4.1 ASIC Implementation Hardware Analysis

To determine how the hardware for our family of CMPR-based stream ciphers compares to TRIVIUM, we synthesize VHDL implementations of TRIVIUM and the CMPR-based stream ciphers to an ASIC target and compare results for hardware area and energy consumption. Synthesis was performed using the Synopsys DesignVision tool [Syn03]. Subsequently, we performed place and route using the Cadence Innovus Implementation System[Cad15] and layout with the Cadence Virtuoso Layout Suite [Cad91]. All of the aforementioned tools were configured with the FreePDK45 45nm standard cell library [Uni11]. The fastest clock frequency we could reliably obtain was 300MHz for all designs. The results are shown in Tables 18 and 19.

Table 18: ASIC Hardware Implementation Area and Security Comparison Between TRIVIUM and CMPR-based Stream Ciphers

Cipher	Area (μm^2)	Power (mW)	Key Security	Init. Rounds
TRIVIUM	5627	3.0226	2^{80}	1152
CMPR stream cipher v_1	12909	5.4405	2^{128}	100
CMPR stream cipher v_2	7057	2.8894	2^{80}	100
CMPR stream cipher v_3	7292	2.8993	2^{84}	100

Table 19: Relative ASIC Hardware Comparison Between TRIVIUM and CMPR-based Stream Ciphers

Cipher	Area Relative to TRIVIUM	Power Relative to TRIVIUM
CMPR stream cipher v_1	+129.4%	+80.0%
CMPR stream cipher v_2	+25.4%	-4.41%
CMPR stream cipher v_3	+29.6%	-4.08%

CMPR stream cipher v_1, which has an internal state size of 288 bits and offers improved security with a 128-bit key, results in a 129.4% larger hardware implementation with 80.0% more energy consumption in comparison to TRIVIUM. The increased area and power of the CMPR stream ciphers as compared to TRIVIUM are due to the chaining functions in the CMPR-based stream cipher design. The larger a CMPR is, the more chaining functions it has, given our design approach. Generally, chaining functions contribute significantly to the hardware area and energy consumption of a CMPR-based design, since each chaining function consists of numerous AND and XOR gates.

CMPR stream ciphers v_2 and v_3 have higher area than TRIVIUM but the increase is not as drastic as in the case of v_1, owing to their reduced internal state sizes of 162 and 170 bits, respectively. Despite their decreased state sizes, CMPR-based stream ciphers v_2 and v_3 can generate far more keystream bits from a single (key, IV) pair than TRIVIUM (due to the period guarantees of Section 3). Ciphers v_2 and v_3 also offer equal or slightly higher key security consume roughly equal energy relative to TRIVIUM.

7.4.2 FPGA Implementation Hardware Analysis

We present an FPGA synthesis comparison between TRIVIUM and three choices from our CMPR-based stream cipher family in Table 20, comparing the hardware usage in terms of

registers, block RAM (BRAM) bits, ALUTs (adaptive lookup tables), and ALMs (adaptive logic modules). ALMs and ALUTs are the combinational logic elements in the Intel Quartus Prime Lite Edition software [Cor19]. Each design was synthesized onto the Intel DE10-Standard board [Ter17]. The Intel DE10-Standard board is a Cyclone®V FPGA[Cor13] with 28nm process technology (thus, the transistors at 28nm have approximately 62% of the channel length size comparted to the ASIC transistors used at 45nm). VHDL implementations of each of the three CMPR stream ciphers and TRIVIUM were synthesized onto the Intel DE10-Standard FPGA board at a clock frequency of 200MHz, which we empirically found to be the threshold frequency at which the design met FPGA timing constraints – in other words, choosing a higher clock frequency, e.g., 205MHz, resulted in failing to meet timing constraints. The key insight provided by the FPGA synthesis results in Table 20 is that CMPR implementations on an FPGA do not require BRAM. In the case of a CMPR, the chaining functions are defined bit-by-bit when generating RTL for a CMPR stream cipher, and the registers are contained in the adaptive logic modules (ALMs); thus, the FPGA synthesis tools will not map any part of the CMPR stream cipher VHDL design to BRAM elements.

Table 20: FPGA Resource Utilization Comparison Between TRIVIUM and CMPR-based Stream Ciphers

	TRIVIUM	CMPR cipher v_1	CMPR cipher v_2	CMPR cipher v_3
Registers	295	323	210	216
BRAM Bits	66	0	0	0
ALMs	283	651	349	366
ALUTs	556	1193	629	653

We also present an FPGA synthesis comparison for CMPR stream cipher v_3 to demonstrate the flexibility of CMPRs when implemented on a reconfigurable hardware target. In Figure 18, we varied the update polynomials of the MPRs used to construct the CMPR seven times – specifically, we investigate $U(x) = x$, $U(x) = x^2$, $U(x) = x^3$, $U(x) = x^4$, $U(x) = x^5$, $U(x) = x^{10}$ and $U(x) = x^{15}$. In this plot, it is implied that an update polynomial is only used if the MPR is large enough to accommodate the update polynomial; for example, for $U(x) = x^{15}$, only MPRs of sizes larger than 15 utilize this update value – the smaller MPRs still utilize $U(x) = x$. We observed slight variations in the lookup tables (ALUTs) and combinational logic blocks (ALMs) required for each FPGA implementation. These results seem to indicate that on reconfigurable hardware platforms, the update polynomials of a CMPR can be varied at little additional hardware implementation cost. In general, an MPR has an exponential number of possible update polynomials, where the number of possible update polynomials depends on the size of the MPR. We explored the FPGA implementation results for a select few out of the many possible update polynomials of Definition 18.

7.5 Comparison Summary

We conclude this section by discussing how our results illustrate the flexibility and advantages of CMPRs over NLFSRs in the context of stream cipher design. Unlike NLFSRs, CMPRs do not support nonlinear feedback. Nonlinear chaining functions in CMPRs feed forward from one MPR to the next, following Definition 22. As a result, a CMPR-based design cannot rely solely on initialization rounds to provide security and resistance against polynomial-based cryptanalytic methods such as cube testers, since there is no nonlinear feedback to increase the degree of the output polynomial. Instead, we can permute the internal state of the CMPR during the initialization stage. Despite the exclusion of nonlinear feedback, CMPRs are still incredibly flexible and provide a framework for scalable, nearly full-period register constructions. Due to the Chaining Period Theorem (Theorem 1), we

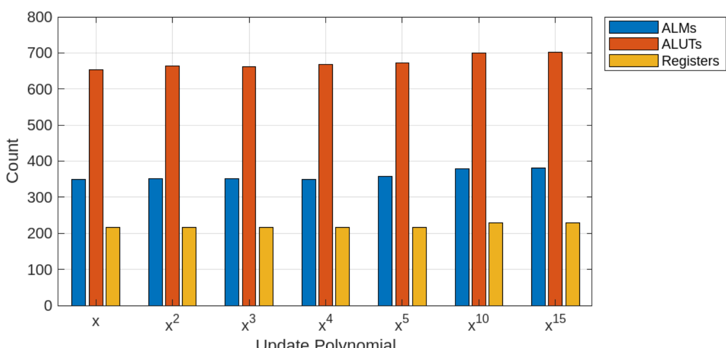


Figure 18: FPGA Resource Utilization for CMPR stream cipher v_3 Varied Update Functions

can guarantee that a CMPR will have exponential period provided that its constituent MPRs use irreducible feedback polynomials.

The motivation behind designing three CMPR-based stream ciphers was to demonstrate the flexibility designers have when constructing CMPR-based schemes. For each of our CMPR-based stream ciphers, we use a different CMPR size, only needing to ensure that the CMPR is large enough to store both the key and IV during the initialization phase. NLFSRs do not have this kind of design flexibility or period guarantee. Full-period NLFSRs are defined for only a few small register sizes [Dub12].

Finally, we observe that the keystreams generated by TRIVIUM and the CMPR stream ciphers both pass cryptographic statistical testing, namely the randomness tests of the NIST Statistical Test Suite and the confusion and avalanche effect principles tested by our bit contribution statistical tests, which verified that a bit flip in the key or IV resulted in a significant change in the keystream. We also observed that the CMPR stream ciphers with identical or similar security margins to TRIVIUM, namely CMPR stream ciphers v_2 and v_3, have ASIC and FPGA hardware implementations with roughly a quarter to a third higher area than TRIVIUM, but nearly identical energy consumption.

8 Discussion

In this section, we wish to acknowledge and highlight some areas of our research that provide promising opportunities for future work and exploration.

- Our restriction to using only Mersenne primes allows for a simpler statement of Theorem 2, as well as simplifying the analysis used to estimate linear complexity. However, with stronger casework and more careful handling, the general ideas presented in Theorem 1 and Section 5 might be extended to a wider array of systems.
- In CMPRs, and all of the specific examples in this paper, all feedback registers are chosen to be linear in order to make the induction simpler and more consistent. All nonlinearity in the update is derived from the nonlinear chaining functions. However, again, this is not a strict requirement, and we foresee future work exploring using chaining to extend nonlinear structures. At a minimum, the very first register in a CMPR can clearly be replaced by a nonlinear feedback register such as an NLFSR.
- Our notion of root expressions seems to offer several powerful advantages for calculating linear complexity in the situations where they are applicable, but suffers from limitations such as requiring Mersenne primes and distinct exponents.
- We are committed to continue our efforts to find attack avenues that have the potential to succeed against CMPR-based structures. For example, although we have

searched for high-order key-independent distinguishers without success to date, it is possible such attacks may succeed in the future.

- Although we focus on the applications of CMPRs to stream ciphers and present a comparison with the TRIVIUM stream cipher, we believe CMPRs can serve as building blocks for different types of cryptographic schemes such as hash functions and block ciphers. We anticipate future work exploring the wider cryptographic applications of CMPRs.
- Our initial investigation of FPGA implementations of CMPRs with varied update polynomials indicates that, with additional analysis, it may be possible for update polynomials to be treated as a hardware-based key not revealed to the adversary. For a given MPR, there exists an exponential number of possible update polynomials (and thus can be quite large depending on the size of the MPR), and based on the update polynomials used, the state sequence of the CMPR will be different. However, there is still a lot of work to be done in this space.

9 Conclusion

The main mathematical proof on which the majority of the results in this paper depend is the Chaining Period Theorem (Theorem 1). The Chaining Period Theorem brings together prior work from a range of areas including cascades and T-functions, and which applies to many feedback register constructions. Based on this theorem, we present a new type of linear feedback register which we term a Product Register (PR) and show that if the PR length is a Mersenne exponent, we can create update functions with greater variety. Furthermore, these Mersenne Product Registers (MPRs) are well-suited for use with the Chaining Period Theorem to create a Composite Mersenne Product Register (CMPR). This allows for nonlinear state sequences and drastically increases the space of potential CMPR constructions. Regardless of the chaining functions chosen, we are able to show any CMPR has an exponential expected period. To estimate CMPR linear complexity more efficiently than what is provided by the classic Berlekamp-Massey algorithm, we develop the concepts of root expressions, which we use to create the CMPR Root Expression Algorithm (Algorithm 1). We apply a set of experiments consisting of cryptanalytic and statistical tests on simple CMPRs, which entail a restriction on the chaining functions with the aim of protecting against cube attacks. Furthermore, we add swapping to the initialization rounds to resist linear distinguishers. Finally, we design and test a family of three CMPR-based stream ciphers that vary in key length, IV length, and internal state size (number of register bits). We also implement the CMPR-based stream ciphers on ASIC and FPGA hardware targets, with a hardware cost comparison to the NLFSR-based TRIVIUM stream cipher. Overall, our results appear to confirm that CMPRs are suitable for generating pseudorandom numbers for cryptographic applications and that the hardware implementations of CMPRs can compare to TRIVIUM which is a representative NLFSR-based cryptographic scheme from the literature. We believe that this research sets the stage for the design of new hardware-oriented cryptographic primitives utilizing CMPRs.

References

[Abr94] Miron Abramovici. *Digital Systems Testing and Testable Design*. IEEE Wiley-Interscience, New York, NY, 1994. ISBN: 0780310624.

[ADMS09] Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. *Cube testers and key recovery attacks on reduced-round md6 and trivium*. In *Lecture Notes in Computer Science*. Springer, Cham, 2009, pages 1–22. ISBN: 978-3-642-03317-9. DOI: 10.1007/978-3-642-03317-9_1. URL: https://doi.org/10.1007/978-3-642-03317-9_1.

[AFI+04] Gwénolé Ars, Jean-Charles Faugère, Hideki Imai, Mitsuru Kawazoe, and Makoto Sugita. Comparison Between XL and Gröbner Basis Algorithms. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004*, pages 338–353, Berlin, Heidelberg. Springer Berlin Heidelberg, 2004. ISBN: 978-3-540-30539-2. DOI: 10.1007/978-3-540-30539-2_24.

[AHMN12] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and María Naya-Plasencia. Quark: A Lightweight Hash. *Journal of Cryptology*, 26(2):313–339, May 2012. DOI: 10.1007/s00145-012-9125-6. URL: https://doi.org/10.1007/s00145-012-9125-6.

[Arm04] Frederik Armknecht. Improving Fast Algebraic Attacks. In Bimal Roy and Willi Meier, editors, *Fast Software Encryption*, pages 65–82, Berlin, Heidelberg. Springer Berlin Heidelberg, 2004. ISBN: 978-3-540-25937-4. DOI: 10.1007/978-3-540-25937-4_5.

[BRS+10] Lawrence E. Bassham, Andrew L. Rukhin, Juan Soto, James R. Nechvatal, Miles E. Smid, Elaine B. Barker, Stefan D. Leigh, Mark Levenson, Mark Vangel, David L. Banks, Nathanael Alan Heckert, James F. Dray, and San Vo. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Technical report 22, National Institute of Standards and Techonology, April 2010, pages 171–186. URL: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf.

[Cad15] Cadence Design Systems, Inc. Innovus Implementation System, 2015. URL: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/soc-implementation-and-floorplanning/innovus-implementation-system.html. (accessed: 10.07.2024).

[Cad91] Cadence Design Systems, Inc. Virtuoso layout suite, 1991. URL: https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/layout-design/virtuoso-layout-suite.html. (accessed: 10.07.2024).

[Can05] Anne Canteaut. *Filter Generator*. In *Encyclopedia of Cryptography and Security*. Henk C. A. van Tilborg, editor. Springer US, Boston, MA, 2005, pages 223–224. ISBN: 978-0-387-23483-0. DOI: 10.1007/0-387-23483-7_165. URL: https://doi.org/10.1007/0-387-23483-7_165.

[Can11] Anne Canteaut. Combination Generator. In *Encyclopedia of Cryptography and Security (2nd ed.)* Pages 82–83. Springer US, 2011. DOI: 10.1007/0-387-23483-7_70. URL: https://doi.org/10.1007/0-387-23483-7_70.

[CGW20] Zuling Chang, Guang Gong, and Qiang Wang. Cycle Structures of a Class of Cascaded FSRs. *IEEE Transactions on Information Theory*, 66(6):3766–3774, 2020. DOI: 10.1109/TIT.2019.2956741.

[CKPS00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In *Advances in Cryptology — EUROCRYPT 2000*, pages 392–407. Springer, Berlin, Heidelberg, 2000. ISBN: 978-3-540-45539-4. DOI: 10.1007/3-540-45539-6_27.

[CM03] Nicolas T. Courtois and Willi Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In Eli Biham, editor, *Advances in Cryptology — EUROCRYPT 2003*, pages 345–359, Berlin, Heidelberg. Springer Berlin Heidelberg, 2003. DOI: 10.1007/3-540-39200-9_21.

[COOP23] Marco Cianfriglia, Elia Onofri, Silvia Onofri, and Marco Pedicini. Fourteen Years of Cube Attacks. *Applicable Algebra in Engineering, Communication, and Computing*, May 2023. DOI: 10.1007/s00200-023-00602-w. URL: https://doi.org/10.1007/s00200-023-00602-w.

[Cor13] Intel Corporation. Cyclone® V FPGA and SoC FPGA, 2013. URL: https://www.intel.com/content/www/us/en/products/details/fpga/cyclone/v.html. (accessed: 10.07.2024).

[Cor19] Intel Corporation. Intel® Quartus® Prime Lite Edition Design Software Version 20.1.1 for Windows, 2019. URL: https://www.intel.com/content/www/us/en/software-kit/660907/intel-quartus-prime-lite-edition-design-software-version-20-1-1-for-windows.html. (accessed: 10.06.2024).

[Cou03] Nicolas T. Courtois. Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729, pages 176–194, Berlin, Heidelberg. Springer Berlin Heidelberg, 2003. DOI: 10.1007/978-3-540-45146-4_11.

[CP03] N. Courtois and J. Patarin. About the XL algorithm over GF(2). In *Topics in Cryptology — CT-RSA 2003*, volume 2612, pages 141–157. Springer, Berlin, Heidelberg, 2003. DOI: 10.1007/3-540-36563-X_10. URL: https://doi.org/10.1007/3-540-36563-X_10.

[dBru46] N.G. de Bruijn. A Combinatorial Problem. *Proceedings of the Section of Sciences of the Koninklijke Nederlandse Akademie van Wetenschappen te Amsterdam*, 49(7):758–764, 1946.

[De 06] Christophe De Cannière. Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In *Proceedings of the 9th International Conference on Information Security*, ISC'06, pages 171–186, Samos Island, Greece. Springer-Verlag, 2006. ISBN: 3540383417. DOI: 10.1007/11836810_13. URL: https://doi.org/10.1007/11836810_13.

[DP08] Christophe De Cannière and Bart Preneel. Trivium. In *Lecture Notes in Computer Science*, pages 244–266, Berlin, Heidelberg. Springer-Verlag, 2008. DOI: 10.1007/978-3-540-68351-3_18. URL: https://doi.org/10.1007/978-3-540-68351-3_18.

[DS09] Itai Dinur and Adi Shamir. Cube Attacks on Tweakable Black Box Polynomials. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 278–299, Berlin, Heidelberg. Springer Berlin Heidelberg, 2009. DOI: 10.1007/978-3-642-01001-9_16.

[Dub09] Elena Dubrova. A Transformation From the Fibonacci to the Galois NLFSRs. *IEEE Transactions on Information Theory*, 55(11):5263–5271, 2009. DOI: 10.1109/TIT.2009.2030467.

[Dub12] Elena Dubrova. A List of Maximum Period NLFSRs, 2012. URL: https://eprint.iacr.org/2012/166.pdf. (accessed: 10.1.2024).

[Dub14] Elena Dubrova. Generation of full cycles by a composition of NLFSRs. *Designs, Codes, and Cryptography*, 73:469–486, March 2014. DOI: 10.1007/s10623-014-9947-3. URL: https://doi.org/10.1007/s10623-014-9947-3.

[Fau99] JC. Faugère. A New Efficient Algorithm for Computing Gröbner Bases (F4). *Journal of Pure and Applied Algebra*, 139(1):61–88, 1999. DOI: https://doi.org/10.1016/S0022-4049(99)00005-5. URL: https://www.sciencedirect.com/science/article/pii/S0022404999000055.

[Fei73] Horst Feistel. Cryptography and Computer Privacy. *Scientific American*, 228(5):15–23, 1973. ISSN: 00368733, 19467087. URL: http://www.jstor.org/stable/24923044 (visited on 05/30/2023).

[GD70] D.H. Green and K.R. Dimond. Nonlinear Product-Feedback Shift Registers. *Proceedings of the Institution of Electrical Engineers*, 117(4):681–686, 1970. ISSN: 0020-3270. DOI: https://doi.org/10.1049/piee.1970.0134. URL: https://digital-library.theiet.org/content/journals/10.1049/piee.1970.0134.

[Gol82] Solomon Golomb. *Shift Register Sequences*. Aegean Park Press, Laguna Hills, Calif, 1982. ISBN: 978-0894120480.

[Gro10] Larry C. Grove. *Groups and Characters*. John Wiley & Sons, Inc., 2010. DOI: 10.1007/978-3-642-04101-3. URL: https://doi.org/10.1007/978-3-642-04101-3.

[KD79] John B. Kam and George I. Davida. Structured Design of Substitution-Permutation Encryption Networks. *IEEE Transactions on Computers*, C-28(10):747–753, 1979. DOI: 10.1109/TC.1979.1675242.

[Key76] E. Key. An Analysis of the Structure and Complexity of Nonlinear Binary Sequence Generators. *IEEE Transactions on Information Theory*, 22(6):732–736, November 1976. DOI: 10.1109/tit.1976.1055626. URL: https://doi.org/10.1109/tit.1976.1055626.

[KS04] Alexander Klimov and Adi Shamir. Cryptographic Applications of T-Functions. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*, pages 248–261, Berlin, Heidelberg. Springer Berlin Heidelberg, 2004. ISBN: 978-3-540-24654-1. DOI: 10.1007/978-3-540-24654-1_18.

[Leo09] Steven Leon. *Linear Algebra with Applications*. In Pearson, New York, 2009. Chapter 6.1. ISBN: 2009023730.

[LH24] He Lei and Wang Hu. More Balanced Polynomials: Cube Attacks on 810- and 825-Round Trivium with Practical Complexities. *Selected Areas in Cryptography – SAC 2023*, February 2024. DOI: 10.1007/978-3-031-53368-6_1. URL: https://doi.org/10.1007/978-3-031-53368-6_1.

[LN94] Rudolf Lidl and Harald Niederreiter. *Introduction to Finite Fields and their Applications*. Cambridge University Press, 2nd edition, 1994. DOI: 10.1017/CBO9781139172769.

[Mas69] J. Massey. Shift-register Synthesis and BCH Decoding. *IEEE Transactions on Information Theory*, 15(1):122–127, January 1969. DOI: 10.1109/tit.1969.1054260. URL: https://doi.org/10.1109/tit.1969.1054260.

[Mat94] Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, pages 386–397, Berlin, Heidelberg. Springer Berlin Heidelberg, 1994. ISBN: 978-3-540-48285-7. DOI: 10.1007/3-540-48285-7_33.

[MG16] Kalikinkar Mandal and Guang Gong. Feedback Reconstruction and Implementations of Pseudorandom Number Generators from Composited De Bruijn Sequences. *IEEE Transactions on Computers*, 65(9):2725–2738, 2016. DOI: 10.1109/TC.2015.2506557.

[MST79] Johannes Mykkeltveit, Man-Keung Siu, and Po Tong. On the Cycle Structure of Some Nonlinear Shift Register Sequences. *Information and Control*, 43(2):202–215, 1979. ISSN: 0019-9958. DOI: https://doi.org/10.1016/S0019-9958(79)90708-3. URL: https://www.sciencedirect.com/science/article/pii/S0019995879907083.

[MvOV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Public-Key Parameters*. In *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, 1997, pages 154–160. ISBN: 9780429466335. DOI: 10.1201/9780429466335. URL: https://doi.org/10.1201/9780429466335.

[PP97] Christof Paar and Jan Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer Heidelberg Dordrecht London New York, May 1997. DOI: 10.1002/9781118032688. URL: https://doi.org/10.1002/9781118032688.

[PRKK19] Sydney Pugh, MS Raunak, D Richard Kuhn, and Raghu Kacker. Systematic Testing of Lightweight Cryptographic Implementations. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, USA, November 2019.

[Rot95] Joseph Rotman. *An Introduction to the Theory of Groups*. Springer-Verlag, New York, 1995. ISBN: 0387942858.

[Rue86] Rainer A Rueppel. *Analysis and Design of Stream Ciphers*. en. Communications and Control Engineering. Springer, Berlin, Germany, 1986th edition, August 1986.

[Sha45] Claude Shannon. A Mathematical Theory of Cryptography. Classified Report, Bell Laboratories, Murray Hill, NJ, USA, September 1945.

[SM08] Mutsuo Saito and Makoto Matsumoto. SIMD-Oriented Fast Mersenne Twister: a 128-bit Pseudorandom Number Generator. In Alexander Keller, Stefan Heinrich, and Harald Niederreiter, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2006*, pages 607–622, Berlin, Heidelberg. Springer Berlin Heidelberg, 2008. ISBN: 978-3-540-74496-2. DOI: 10.1007/978-3-540-74496-2_36.

[Syn03] Synopsys, Inc. Design Vision User Guide, 2003. URL: http://beethoven.ee.ncku.edu.tw/testlab/course/VLSIdesign_course/course_96/Tool/Design_Vision_User_Guide.pdf. (accessed: 10.06.2024).

[Ter17] Terasic, Inc. De10-standard, 2017. URL: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=1081. (accessed: 09.26.2024).

[Uni11] North Carolina State University. FreePDK45(TM), 2011. URL: https://eda.ncsu.edu/freepdk/freepdk45/. (accessed: 05.22.2023).

[WT86] A. F. Webster and S. E. Tavares. On the Design of S-Boxes. In Hugh C. Williams, editor, *Advances in Cryptology — CRYPTO '85 Proceedings*, pages 523–534, Berlin, Heidelberg. Springer Berlin Heidelberg, 1986. ISBN: 978-3-540-39799-1. DOI: 10.1007/3-540-39799-X_41.

[ZW06] Wenying Zhang and Chuan-Kun Wu. The Algebraic Normal Form, Linear Complexity and k-Error Linear Complexity of Single-Cycle T-Function. In Guang Gong, Tor Helleseth, Hong-Yeop Song, and Kyeongcheol Yang, editors, *Sequences and Their Applications – SETA 2006*, pages 391–401, Berlin, Heidelberg. Springer Berlin Heidelberg, 2006. ISBN: 978-3-540-44524-1. DOI: 10.1007/11863854_34.

Appendices

The content of this appendix can also be found at https://github.com/gt-hwswcosec/cmprs2025

A ANF for CMPRs used in the CMPR-based Stream Cipher Family

The ANF for the large CMPRs used in our stream ciphers is formatted in the same way as in Section 6.3.

A.1 ANF for 288-bit CMPR

$c_{287}[t+1] = c_{286}[t]$
$c_{286}[t+1] = c_{285}[t]$
$c_{285}[t+1] = c_{284}[t]$
$c_{284}[t+1] = c_{283}[t]$
$c_{283}[t+1] = c_{282}[t]$
$c_{282}[t+1] = c_{281}[t]$
$c_{281}[t+1] = c_{280}[t]$
$c_{280}[t+1] = c_{279}[t]$
$c_{279}[t+1] = c_{278}[t]$
$c_{278}[t+1] = c_{277}[t]$
$c_{277}[t+1] = c_{276}[t]$
$c_{276}[t+1] = c_{275}[t]$
$c_{275}[t+1] = c_{274}[t]$
$c_{274}[t+1] = c_{273}[t]$
$c_{273}[t+1] = c_{272}[t]$
$c_{272}[t+1] = c_{271}[t]$
$c_{271}[t+1] = c_{270}[t]$
$c_{270}[t+1] = c_{287}[t] \oplus c_{269}[t]$
$c_{269}[t+1] = c_{268}[t]$
$c_{268}[t+1] = c_{267}[t]$
$c_{267}[t+1] = c_{266}[t]$
$c_{266}[t+1] = c_{265}[t]$
$c_{265}[t+1] = c_{287}[t] \oplus c_{264}[t]$
$c_{264}[t+1] = c_{263}[t]$
$c_{263}[t+1] = c_{262}[t]$
$c_{262}[t+1] = c_{261}[t]$
$c_{261}[t+1] = c_{260}[t]$
$c_{260}[t+1] = c_{259}[t]$
$c_{259}[t+1] = c_{258}[t]$
$c_{258}[t+1] = c_{257}[t]$
$c_{257}[t+1] = c_{256}[t]$
$c_{256}[t+1] = c_{255}[t]$

$c_{255}[t+1] = c_{254}[t]$
$c_{254}[t+1] = c_{253}[t]$
$c_{253}[t+1] = c_{252}[t]$
$c_{252}[t+1] = c_{251}[t]$
$c_{251}[t+1] = c_{250}[t]$
$c_{250}[t+1] = c_{249}[t]$
$c_{249}[t+1] = c_{248}[t]$
$c_{248}[t+1] = c_{247}[t]$
$c_{247}[t+1] = c_{246}[t]$
$c_{246}[t+1] = c_{245}[t]$
$c_{245}[t+1] = c_{244}[t]$
$c_{244}[t+1] = c_{243}[t]$
$c_{243}[t+1] = c_{242}[t]$
$c_{242}[t+1] = c_{241}[t]$
$c_{241}[t+1] = c_{240}[t]$
$c_{240}[t+1] = c_{239}[t]$
$c_{239}[t+1] = c_{238}[t]$
$c_{238}[t+1] = c_{237}[t]$
$c_{237}[t+1] = c_{236}[t]$
$c_{236}[t+1] = c_{235}[t]$
$c_{235}[t+1] = c_{234}[t]$
$c_{234}[t+1] = c_{233}[t]$
$c_{233}[t+1] = c_{232}[t]$
$c_{232}[t+1] = c_{231}[t]$
$c_{231}[t+1] = c_{230}[t]$
$c_{230}[t+1] = c_{229}[t]$
$c_{229}[t+1] = c_{228}[t]$
$c_{228}[t+1] = c_{227}[t]$
$c_{227}[t+1] = c_{226}[t]$
$c_{226}[t+1] = c_{225}[t]$
$c_{225}[t+1] = c_{224}[t]$
$c_{224}[t+1] = c_{223}[t]$
$c_{223}[t+1] = c_{222}[t]$
$c_{222}[t+1] = c_{221}[t]$
$c_{221}[t+1] = c_{287}[t] \oplus c_{220}[t]$
$c_{220}[t+1] = c_{219}[t]$
$c_{219}[t+1] = c_{218}[t]$
$c_{218}[t+1] = c_{217}[t]$
$c_{217}[t+1] = c_{216}[t]$
$c_{216}[t+1] = c_{215}[t]$
$c_{215}[t+1] = c_{214}[t]$
$c_{214}[t+1] = c_{213}[t]$
$c_{213}[t+1] = c_{212}[t]$
$c_{212}[t+1] = c_{211}[t]$
$c_{211}[t+1] = c_{210}[t]$
$c_{210}[t+1] = c_{287}[t] \oplus c_{209}[t]$
$c_{209}[t+1] = c_{208}[t]$
$c_{208}[t+1] = c_{207}[t]$
$c_{207}[t+1] = c_{206}[t]$
$c_{206}[t+1] = c_{205}[t]$
$c_{205}[t+1] = c_{204}[t]$
$c_{204}[t+1] = c_{287}[t] \oplus c_{203}[t]$
$c_{203}[t+1] = c_{202}[t]$
$c_{202}[t+1] = c_{201}[t]$
$c_{201}[t+1] = c_{200}[t]$
$c_{200}[t+1] = c_{199}[t]$
$c_{199}[t+1] = c_{198}[t]$

$c_{198}[t+1] = c_{197}[t]$
$c_{197}[t+1] = c_{196}[t]$
$c_{196}[t+1] = c_{195}[t]$
$c_{195}[t+1] = c_{194}[t]$
$c_{194}[t+1] = c_{193}[t]$
$c_{193}[t+1] = c_{192}[t]$
$c_{192}[t+1] = c_{191}[t]$
$c_{191}[t+1] = c_{190}[t]$
$c_{190}[t+1] = c_{189}[t]$
$c_{189}[t+1] = c_{188}[t]$
$c_{188}[t+1] = c_{187}[t]$
$c_{187}[t+1] = c_{186}[t]$
$c_{186}[t+1] = c_{185}[t]$
$c_{185}[t+1] = c_{184}[t]$
$c_{184}[t+1] = c_{183}[t]$
$c_{183}[t+1] = c_{182}[t]$
$c_{182}[t+1] = c_{181}[t]$
$c_{181}[t+1] = c_{287}[t]$
$c_{180}[t+1] = c_{179}[t] \oplus c_{270}[t]c_{286}[t]c_{261}[t]c_{192}[t] \oplus c_{195}[t]c_{215}[t]c_{194}[t] \oplus c_{230}[t]c_{284}[t] \oplus c_{268}[t]c_{251}[t]c_{192}[t]$
$c_{179}[t+1] = c_{178}[t] \oplus c_{214}[t]c_{281}[t]c_{232}[t]c_{208}[t] \oplus c_{184}[t]c_{218}[t] \oplus c_{220}[t]c_{268}[t]$
$c_{178}[t+1] = c_{177}[t] \oplus c_{212}[t]c_{211}[t]c_{190}[t]c_{271}[t] \oplus c_{181}[t]c_{275}[t]c_{215}[t] \oplus c_{194}[t]c_{227}[t]c_{217}[t]c_{272}[t] \oplus c_{202}[t]c_{251}[t]c_{210}[t]c_{195}[t]$
$c_{177}[t+1] = c_{176}[t] \oplus c_{230}[t]c_{181}[t]c_{242}[t] \oplus c_{182}[t]c_{232}[t]c_{248}[t] \oplus c_{245}[t]c_{239}[t]c_{267}[t]c_{251}[t]$
$c_{176}[t+1] = c_{175}[t] \oplus c_{215}[t]c_{236}[t]c_{254}[t]c_{227}[t] \oplus c_{218}[t]c_{272}[t]c_{285}[t]c_{226}[t] \oplus c_{281}[t]c_{209}[t]$
$c_{175}[t+1] = c_{174}[t] \oplus c_{241}[t]c_{217}[t]c_{262}[t] \oplus c_{266}[t]c_{244}[t]c_{205}[t]c_{285}[t]$
$c_{174}[t+1] = c_{173}[t] \oplus c_{186}[t]c_{228}[t]c_{220}[t]c_{251}[t] \oplus c_{270}[t]c_{203}[t] \oplus c_{248}[t]c_{222}[t]c_{256}[t]$
$c_{173}[t+1] = c_{172}[t] \oplus c_{180}[t] \oplus c_{245}[t]c_{192}[t] \oplus c_{227}[t]c_{203}[t] \oplus c_{269}[t]c_{198}[t]c_{223}[t] \oplus c_{196}[t]c_{186}[t]c_{258}[t]c_{230}[t]$
$c_{172}[t+1] = c_{171}[t] \oplus c_{285}[t]c_{260}[t] \oplus c_{249}[t]c_{259}[t] \oplus c_{279}[t]c_{266}[t]c_{238}[t]c_{220}[t] \oplus c_{278}[t]c_{267}[t]c_{220}[t]c_{202}[t]$
$c_{171}[t+1] = c_{170}[t] \oplus c_{285}[t]c_{187}[t]c_{223}[t]c_{198}[t] \oplus c_{276}[t]c_{198}[t]c_{211}[t] \oplus c_{181}[t]c_{238}[t]c_{260}[t] \oplus c_{202}[t]c_{191}[t]c_{199}[t]c_{196}[t]$
$c_{170}[t+1] = c_{169}[t] \oplus c_{262}[t]c_{204}[t] \oplus c_{256}[t]c_{200}[t]$
$c_{169}[t+1] = c_{168}[t] \oplus c_{215}[t]c_{282}[t] \oplus c_{192}[t]c_{244}[t]c_{280}[t]c_{275}[t] \oplus c_{286}[t]c_{192}[t]c_{263}[t] \oplus c_{197}[t]c_{247}[t]c_{199}[t]$
$c_{168}[t+1] = c_{167}[t] \oplus c_{271}[t]c_{204}[t] \oplus c_{194}[t]c_{276}[t] \oplus c_{283}[t]c_{241}[t] \oplus c_{257}[t]c_{209}[t]c_{275}[t]$
$c_{167}[t+1] = c_{166}[t] \oplus c_{276}[t]c_{213}[t]c_{214}[t]c_{190}[t] \oplus c_{247}[t]c_{238}[t]c_{275}[t]c_{219}[t] \oplus c_{204}[t]c_{252}[t]c_{287}[t]c_{257}[t] \oplus c_{247}[t]c_{257}[t]c_{267}[t]$
$c_{166}[t+1] = c_{165}[t] \oplus c_{211}[t]c_{221}[t] \oplus c_{255}[t]c_{208}[t]c_{272}[t] \oplus c_{257}[t]c_{285}[t]c_{200}[t] \oplus c_{201}[t]c_{245}[t]$
$c_{165}[t+1] = c_{164}[t] \oplus c_{219}[t]c_{245}[t]c_{252}[t]c_{226}[t] \oplus c_{221}[t]c_{201}[t]c_{186}[t]c_{250}[t] \oplus c_{198}[t]c_{253}[t]c_{193}[t]$
$c_{164}[t+1] = c_{163}[t] \oplus c_{264}[t]c_{246}[t]c_{245}[t] \oplus c_{183}[t]c_{201}[t]$
$c_{163}[t+1] = c_{162}[t] \oplus c_{245}[t]c_{223}[t] \oplus c_{280}[t]c_{252}[t]c_{229}[t] \oplus c_{249}[t]c_{226}[t]$
$c_{162}[t+1] = c_{161}[t] \oplus c_{277}[t]c_{194}[t] \oplus c_{226}[t]c_{251}[t]c_{192}[t]c_{242}[t] \oplus c_{270}[t]c_{205}[t]c_{211}[t]c_{266}[t]$
$c_{161}[t+1] = c_{160}[t] \oplus c_{284}[t]c_{191}[t]c_{198}[t] \oplus c_{231}[t]c_{247}[t]$
$c_{160}[t+1] = c_{180}[t] \oplus c_{159}[t] \oplus c_{220}[t]c_{263}[t] \oplus c_{236}[t]c_{228}[t]c_{266}[t] \oplus c_{251}[t]c_{249}[t]c_{199}[t]c_{221}[t]$
$c_{159}[t+1] = c_{158}[t] \oplus c_{188}[t]c_{216}[t]c_{265}[t]c_{261}[t] \oplus c_{231}[t]c_{233}[t]c_{190}[t] \oplus c_{186}[t]c_{190}[t]c_{260}[t] \oplus c_{227}[t]c_{217}[t]$
$c_{158}[t+1] = c_{157}[t] \oplus c_{203}[t]c_{283}[t] \oplus c_{260}[t]c_{220}[t]c_{185}[t] \oplus c_{204}[t]c_{226}[t]c_{193}[t]$
$c_{157}[t+1] = c_{156}[t] \oplus c_{268}[t]c_{197}[t]c_{215}[t]c_{286}[t] \oplus c_{256}[t]c_{205}[t]c_{204}[t]c_{245}[t] \oplus c_{216}[t]c_{282}[t]c_{248}[t] \oplus c_{281}[t]c_{282}[t]c_{250}[t]c_{197}[t]$
$c_{156}[t+1] = c_{155}[t] \oplus c_{201}[t]c_{202}[t]c_{277}[t]c_{213}[t] \oplus c_{233}[t]c_{253}[t]c_{183}[t]c_{277}[t] \oplus c_{215}[t]c_{200}[t]c_{278}[t]c_{184}[t]$
$c_{155}[t+1] = c_{154}[t] \oplus c_{194}[t]c_{218}[t]c_{253}[t] \oplus c_{243}[t]c_{251}[t]c_{282}[t] \oplus c_{281}[t]c_{274}[t]$
$c_{154}[t+1] = c_{153}[t] \oplus c_{195}[t]c_{234}[t] \oplus c_{229}[t]c_{190}[t]c_{248}[t]c_{226}[t]$
$c_{153}[t+1] = c_{152}[t] \oplus c_{256}[t]c_{280}[t] \oplus c_{208}[t]c_{242}[t] \oplus c_{258}[t]c_{195}[t]c_{239}[t]c_{211}[t] \oplus c_{240}[t]c_{244}[t]$
$c_{152}[t+1] = c_{151}[t] \oplus c_{214}[t]c_{187}[t]c_{185}[t]c_{248}[t] \oplus c_{214}[t]c_{201}[t]c_{262}[t] \oplus c_{253}[t]c_{233}[t]$
$c_{151}[t+1] = c_{150}[t] \oplus c_{185}[t]c_{189}[t] \oplus c_{211}[t]c_{191}[t]$
$c_{150}[t+1] = c_{149}[t] \oplus c_{255}[t]c_{239}[t] \oplus c_{234}[t]c_{245}[t]c_{239}[t]c_{222}[t] \oplus c_{227}[t]c_{281}[t] \oplus c_{233}[t]c_{201}[t]c_{255}[t]c_{276}[t]$
$c_{149}[t+1] = c_{148}[t] \oplus c_{200}[t]c_{212}[t] \oplus c_{218}[t]c_{185}[t]c_{273}[t]c_{240}[t] \oplus c_{248}[t]c_{276}[t]c_{253}[t] \oplus c_{185}[t]c_{222}[t]c_{277}[t]$
$c_{148}[t+1] = c_{147}[t] \oplus c_{273}[t]c_{224}[t] \oplus c_{195}[t]c_{187}[t]$
$c_{147}[t+1] = c_{146}[t] \oplus c_{212}[t]c_{257}[t] \oplus c_{218}[t]c_{238}[t]c_{216}[t]$
$c_{146}[t+1] = c_{145}[t] \oplus c_{284}[t]c_{206}[t]c_{216}[t] \oplus c_{209}[t]c_{230}[t]c_{181}[t] \oplus c_{219}[t]c_{269}[t]c_{185}[t]$
$c_{145}[t+1] = c_{144}[t] \oplus c_{183}[t]c_{240}[t]c_{234}[t] \oplus c_{207}[t]c_{249}[t]c_{263}[t]c_{246}[t] \oplus c_{218}[t]c_{221}[t]c_{214}[t] \oplus c_{269}[t]c_{219}[t]$
$c_{144}[t+1] = c_{143}[t] \oplus c_{280}[t]c_{276}[t]c_{183}[t] \oplus c_{268}[t]c_{185}[t]c_{213}[t]c_{225}[t] \oplus c_{220}[t]c_{277}[t]c_{239}[t]$
$c_{143}[t+1] = c_{142}[t] \oplus c_{267}[t]c_{239}[t]c_{245}[t] \oplus c_{270}[t]c_{198}[t] \oplus c_{266}[t]c_{191}[t]c_{226}[t]c_{206}[t]$
$c_{142}[t+1] = c_{141}[t] \oplus c_{206}[t]c_{276}[t]c_{232}[t] \oplus c_{229}[t]c_{206}[t]c_{266}[t]c_{248}[t] \oplus c_{185}[t]c_{245}[t] \oplus c_{277}[t]c_{230}[t]c_{234}[t]c_{227}[t]$

$c_{141}[t+1] = c_{140}[t] \oplus c_{221}[t]c_{214}[t] \oplus c_{264}[t]c_{239}[t] \oplus c_{214}[t]c_{240}[t]c_{226}[t]$
$c_{140}[t+1] = c_{139}[t] \oplus c_{219}[t]c_{235}[t] \oplus c_{217}[t]c_{230}[t]c_{280}[t]c_{258}[t]$
$c_{139}[t+1] = c_{138}[t] \oplus c_{226}[t]c_{263}[t]c_{239}[t] \oplus c_{209}[t]c_{264}[t]c_{214}[t]c_{204}[t] \oplus c_{227}[t]c_{198}[t]c_{275}[t] \oplus c_{245}[t]c_{184}[t]$
$c_{138}[t+1] = c_{137}[t] \oplus c_{218}[t]c_{226}[t]c_{230}[t] \oplus c_{243}[t]c_{257}[t]c_{269}[t]c_{214}[t]$
$c_{137}[t+1] = c_{136}[t] \oplus c_{271}[t]c_{191}[t]c_{183}[t]c_{253}[t] \oplus c_{236}[t]c_{240}[t]c_{195}[t] \oplus c_{225}[t]c_{214}[t]c_{219}[t]c_{218}[t] \oplus c_{258}[t]c_{268}[t]$
$c_{136}[t+1] = c_{135}[t] \oplus c_{287}[t]c_{237}[t] \oplus c_{260}[t]c_{217}[t]$
$c_{135}[t+1] = c_{134}[t] \oplus c_{220}[t]c_{214}[t]c_{256}[t]c_{230}[t] \oplus c_{212}[t]c_{217}[t]c_{198}[t] \oplus c_{213}[t]c_{220}[t]c_{214}[t] \oplus c_{256}[t]c_{265}[t]c_{192}[t]c_{194}[t]$
$c_{134}[t+1] = c_{133}[t] \oplus c_{238}[t]c_{195}[t] \oplus c_{266}[t]c_{199}[t]c_{268}[t] \oplus c_{261}[t]c_{287}[t] \oplus c_{279}[t]c_{222}[t]c_{188}[t]c_{221}[t]$
$c_{133}[t+1] = c_{132}[t] \oplus c_{250}[t]c_{266}[t]c_{281}[t]c_{217}[t] \oplus c_{279}[t]c_{188}[t]c_{211}[t] \oplus c_{278}[t]c_{208}[t]c_{274}[t]$
$c_{132}[t+1] = c_{131}[t] \oplus c_{264}[t]c_{253}[t] \oplus c_{210}[t]c_{230}[t]c_{215}[t] \oplus c_{259}[t]c_{242}[t] \oplus c_{229}[t]c_{222}[t]$
$c_{131}[t+1] = c_{130}[t] \oplus c_{237}[t]c_{249}[t]c_{195}[t] \oplus c_{192}[t]c_{250}[t] \oplus c_{281}[t]c_{284}[t]c_{280}[t]c_{283}[t] \oplus c_{279}[t]c_{265}[t]c_{269}[t]$
$c_{130}[t+1] = c_{129}[t] \oplus c_{250}[t]c_{205}[t]c_{213}[t]c_{253}[t] \oplus c_{285}[t]c_{189}[t]c_{278}[t] \oplus c_{194}[t]c_{239}[t]c_{215}[t] \oplus c_{277}[t]c_{208}[t]c_{218}[t]c_{261}[t]$
$c_{129}[t+1] = c_{128}[t] \oplus c_{247}[t]c_{279}[t]c_{202}[t] \oplus c_{231}[t]c_{234}[t]c_{217}[t] \oplus c_{263}[t]c_{224}[t]c_{258}[t] \oplus c_{223}[t]c_{254}[t]c_{267}[t]$
$c_{128}[t+1] = c_{127}[t] \oplus c_{237}[t]c_{246}[t] \oplus c_{237}[t]c_{195}[t]c_{279}[t]$
$c_{127}[t+1] = c_{126}[t] \oplus c_{185}[t]c_{227}[t] \oplus c_{250}[t]c_{206}[t]c_{183}[t]c_{224}[t] \oplus c_{224}[t]c_{274}[t]c_{285}[t]$
$c_{126}[t+1] = c_{125}[t] \oplus c_{268}[t]c_{266}[t]c_{254}[t]c_{282}[t] \oplus c_{277}[t]c_{285}[t]c_{281}[t] \oplus c_{236}[t]c_{234}[t]c_{191}[t]c_{280}[t] \oplus c_{251}[t]c_{282}[t]$
$c_{125}[t+1] = c_{124}[t] \oplus c_{212}[t]c_{221}[t]c_{274}[t]c_{203}[t] \oplus c_{246}[t]c_{285}[t]c_{254}[t] \oplus c_{265}[t]c_{225}[t] \oplus c_{259}[t]c_{181}[t]$
$c_{124}[t+1] = c_{123}[t] \oplus c_{209}[t]c_{281}[t]c_{187}[t] \oplus c_{193}[t]c_{218}[t]c_{197}[t]$
$c_{123}[t+1] = c_{180}[t] \oplus c_{122}[t] \oplus c_{230}[t]c_{210}[t]c_{259}[t] \oplus c_{210}[t]c_{282}[t] \oplus c_{257}[t]c_{207}[t]c_{234}[t] \oplus c_{266}[t]c_{223}[t]c_{193}[t]$
$c_{122}[t+1] = c_{121}[t] \oplus c_{230}[t]c_{208}[t] \oplus c_{243}[t]c_{267}[t]c_{202}[t]$
$c_{121}[t+1] = c_{120}[t] \oplus c_{200}[t]c_{263}[t]c_{205}[t] \oplus c_{258}[t]c_{196}[t] \oplus c_{242}[t]c_{219}[t]c_{259}[t]$
$c_{120}[t+1] = c_{119}[t] \oplus c_{274}[t]c_{198}[t]c_{240}[t]c_{263}[t] \oplus c_{205}[t]c_{252}[t]c_{185}[t]c_{274}[t] \oplus c_{218}[t]c_{189}[t]c_{214}[t] \oplus c_{206}[t]c_{226}[t]c_{220}[t]$
$c_{119}[t+1] = c_{118}[t] \oplus c_{285}[t]c_{287}[t]c_{214}[t]c_{281}[t] \oplus c_{266}[t]c_{243}[t] \oplus c_{236}[t]c_{242}[t]c_{212}[t]c_{202}[t] \oplus c_{280}[t]c_{185}[t]c_{251}[t]$
$c_{118}[t+1] = c_{117}[t] \oplus c_{237}[t]c_{266}[t]c_{248}[t]c_{246}[t] \oplus c_{281}[t]c_{206}[t]$
$c_{117}[t+1] = c_{116}[t] \oplus c_{192}[t]c_{233}[t]c_{277}[t] \oplus c_{268}[t]c_{264}[t] \oplus c_{274}[t]c_{261}[t]c_{224}[t]c_{206}[t]$
$c_{116}[t+1] = c_{115}[t] \oplus c_{259}[t]c_{187}[t]c_{197}[t] \oplus c_{287}[t]c_{202}[t] \oplus c_{188}[t]c_{211}[t]c_{194}[t] \oplus c_{222}[t]c_{212}[t]c_{240}[t]$
$c_{115}[t+1] = c_{114}[t] \oplus c_{183}[t]c_{210}[t]c_{275}[t] \oplus c_{229}[t]c_{278}[t]c_{213}[t] \oplus c_{197}[t]c_{258}[t] \oplus c_{285}[t]c_{241}[t]c_{204}[t]$
$c_{114}[t+1] = c_{113}[t] \oplus c_{191}[t]c_{254}[t]c_{185}[t] \oplus c_{269}[t]c_{264}[t]c_{274}[t] \oplus c_{270}[t]c_{216}[t]c_{218}[t] \oplus c_{247}[t]c_{222}[t]c_{235}[t]$
$c_{113}[t+1] = c_{180}[t] \oplus c_{112}[t] \oplus c_{181}[t]c_{220}[t] \oplus c_{243}[t]c_{251}[t]c_{274}[t]c_{195}[t] \oplus c_{257}[t]c_{243}[t]c_{240}[t]$
$c_{112}[t+1] = c_{111}[t] \oplus c_{258}[t]c_{282}[t] \oplus c_{255}[t]c_{272}[t]c_{278}[t]c_{204}[t] \oplus c_{228}[t]c_{196}[t]c_{269}[t] \oplus c_{183}[t]c_{191}[t]c_{283}[t]c_{226}[t]$
$c_{111}[t+1] = c_{110}[t] \oplus c_{227}[t]c_{216}[t] \oplus c_{187}[t]c_{225}[t]c_{188}[t]c_{285}[t]$
$c_{110}[t+1] = c_{180}[t] \oplus c_{109}[t] \oplus c_{207}[t]c_{266}[t]c_{219}[t]c_{283}[t] \oplus c_{280}[t]c_{275}[t]$
$c_{109}[t+1] = c_{108}[t] \oplus c_{257}[t]c_{266}[t] \oplus c_{278}[t]c_{239}[t]$
$c_{108}[t+1] = c_{107}[t] \oplus c_{228}[t]c_{241}[t]c_{200}[t] \oplus c_{219}[t]c_{248}[t]$
$c_{107}[t+1] = c_{106}[t] \oplus c_{258}[t]c_{265}[t]c_{244}[t]c_{195}[t] \oplus c_{187}[t]c_{228}[t]c_{286}[t]c_{222}[t]$
$c_{106}[t+1] = c_{105}[t] \oplus c_{233}[t]c_{185}[t] \oplus c_{229}[t]c_{266}[t]c_{202}[t]c_{212}[t]$
$c_{105}[t+1] = c_{104}[t] \oplus c_{188}[t]c_{274}[t] \oplus c_{218}[t]c_{287}[t] \oplus c_{243}[t]c_{194}[t]c_{221}[t]$
$c_{104}[t+1] = c_{103}[t] \oplus c_{286}[t]c_{248}[t]c_{259}[t]c_{236}[t] \oplus c_{248}[t]c_{185}[t]c_{265}[t] \oplus c_{239}[t]c_{261}[t]c_{273}[t]c_{212}[t] \oplus c_{185}[t]c_{195}[t]$
$c_{103}[t+1] = c_{102}[t] \oplus c_{185}[t]c_{241}[t] \oplus c_{188}[t]c_{282}[t]c_{208}[t] \oplus c_{226}[t]c_{204}[t]c_{201}[t]$
$c_{102}[t+1] = c_{101}[t] \oplus c_{270}[t]c_{187}[t]c_{222}[t] \oplus c_{275}[t]c_{218}[t]c_{222}[t]$
$c_{101}[t+1] = c_{100}[t] \oplus c_{277}[t]c_{237}[t]c_{252}[t]c_{182}[t] \oplus c_{268}[t]c_{246}[t]c_{273}[t]$
$c_{100}[t+1] = c_{99}[t] \oplus c_{272}[t]c_{220}[t]c_{211}[t] \oplus c_{222}[t]c_{209}[t]c_{223}[t]c_{256}[t] \oplus c_{279}[t]c_{191}[t] \oplus c_{220}[t]c_{236}[t]c_{205}[t]$
$c_{99}[t+1] = c_{98}[t] \oplus c_{206}[t]c_{272}[t] \oplus c_{207}[t]c_{196}[t]c_{231}[t]c_{193}[t] \oplus c_{252}[t]c_{209}[t]c_{256}[t]$
$c_{98}[t+1] = c_{97}[t] \oplus c_{268}[t]c_{273}[t]c_{184}[t] \oplus c_{278}[t]c_{231}[t]$
$c_{97}[t+1] = c_{96}[t] \oplus c_{238}[t]c_{257}[t]c_{191}[t] \oplus c_{192}[t]c_{278}[t]c_{226}[t]c_{206}[t] \oplus c_{275}[t]c_{256}[t]c_{264}[t] \oplus c_{281}[t]c_{277}[t]$
$c_{96}[t+1] = c_{95}[t] \oplus c_{243}[t]c_{199}[t]c_{238}[t]c_{242}[t] \oplus c_{262}[t]c_{205}[t]$
$c_{95}[t+1] = c_{94}[t] \oplus c_{209}[t]c_{281}[t] \oplus c_{224}[t]c_{237}[t] \oplus c_{285}[t]c_{190}[t]c_{213}[t]c_{258}[t]$
$c_{94}[t+1] = c_{93}[t] \oplus c_{181}[t]c_{251}[t]c_{221}[t]c_{199}[t] \oplus c_{286}[t]c_{227}[t]c_{209}[t] \oplus c_{248}[t]c_{276}[t]c_{241}[t]c_{225}[t]$
$c_{93}[t+1] = c_{92}[t] \oplus c_{262}[t]c_{223}[t] \oplus c_{279}[t]c_{200}[t]c_{283}[t]c_{233}[t]$
$c_{92}[t+1] = c_{180}[t] \oplus c_{263}[t]c_{225}[t] \oplus c_{250}[t]c_{265}[t]c_{258}[t]c_{270}[t] \oplus c_{285}[t]c_{252}[t]c_{202}[t]c_{250}[t] \oplus c_{243}[t]c_{194}[t]c_{241}[t]c_{209}[t]$
$c_{91}[t+1] = c_{90}[t] \oplus c_{163}[t]c_{128}[t] \oplus c_{120}[t]c_{162}[t] \oplus c_{117}[t]c_{165}[t] \oplus c_{130}[t]c_{163}[t]c_{180}[t]c_{119}[t]$
$c_{90}[t+1] = c_{89}[t] \oplus c_{177}[t]c_{147}[t]c_{162}[t]c_{152}[t] \oplus c_{115}[t]c_{92}[t]c_{164}[t]c_{136}[t] \oplus c_{164}[t]c_{119}[t]c_{118}[t]c_{122}[t] \oplus c_{128}[t]c_{106}[t]$
$c_{89}[t+1] = c_{88}[t] \oplus c_{101}[t]c_{163}[t] \oplus c_{118}[t]c_{141}[t]c_{97}[t]c_{162}[t] \oplus c_{142}[t]c_{171}[t] \oplus c_{102}[t]c_{178}[t]c_{98}[t]c_{155}[t]$
$c_{88}[t+1] = c_{87}[t] \oplus c_{156}[t]c_{172}[t] \oplus c_{97}[t]c_{156}[t]c_{127}[t]c_{115}[t] \oplus c_{96}[t]c_{137}[t]$
$c_{87}[t+1] = c_{86}[t] \oplus c_{131}[t]c_{142}[t]c_{118}[t]c_{115}[t] \oplus c_{113}[t]c_{96}[t]c_{137}[t] \oplus c_{107}[t]c_{144}[t]c_{178}[t]$
$c_{86}[t+1] = c_{85}[t] \oplus c_{107}[t]c_{174}[t]c_{116}[t]c_{153}[t] \oplus c_{174}[t]c_{106}[t]c_{146}[t]c_{96}[t] \oplus c_{93}[t]c_{96}[t]c_{150}[t]c_{156}[t]$
$c_{85}[t+1] = c_{84}[t] \oplus c_{144}[t]c_{135}[t]c_{103}[t]c_{127}[t] \oplus c_{170}[t]c_{145}[t]c_{147}[t]c_{158}[t] \oplus c_{156}[t]c_{125}[t]c_{137}[t]$

$c_{84}[t+1] = c_{83}[t] \oplus c_{107}[t]c_{108}[t]c_{140}[t]c_{149}[t] \oplus c_{102}[t]c_{110}[t]c_{105}[t] \oplus c_{115}[t]c_{141}[t]c_{101}[t]$
$c_{83}[t+1] = c_{82}[t] \oplus c_{124}[t]c_{118}[t]c_{142}[t] \oplus c_{133}[t]c_{125}[t] \oplus c_{96}[t]c_{148}[t]c_{131}[t]$
$c_{82}[t+1] = c_{81}[t] \oplus c_{152}[t]c_{137}[t]c_{116}[t] \oplus c_{129}[t]c_{125}[t]c_{124}[t]$
$c_{81}[t+1] = c_{80}[t] \oplus c_{126}[t]c_{135}[t]c_{158}[t]c_{143}[t] \oplus c_{154}[t]c_{121}[t] \oplus c_{153}[t]c_{126}[t] \oplus c_{140}[t]c_{173}[t]c_{137}[t]c_{136}[t]$
$c_{80}[t+1] = c_{79}[t] \oplus c_{106}[t]c_{142}[t]c_{108}[t] \oplus c_{151}[t]c_{107}[t]c_{118}[t]c_{169}[t] \oplus c_{95}[t]c_{153}[t]c_{99}[t]$
$c_{79}[t+1] = c_{78}[t] \oplus c_{111}[t]c_{160}[t]c_{152}[t] \oplus c_{93}[t]c_{100}[t]c_{178}[t] \oplus c_{96}[t]c_{165}[t]c_{177}[t]c_{130}[t]$
$c_{78}[t+1] = c_{77}[t] \oplus c_{113}[t]c_{169}[t] \oplus c_{103}[t]c_{176}[t]c_{167}[t] \oplus c_{126}[t]c_{116}[t]c_{124}[t]c_{172}[t]$
$c_{77}[t+1] = c_{76}[t] \oplus c_{115}[t]c_{172}[t]c_{174}[t]c_{166}[t] \oplus c_{130}[t]c_{128}[t] \oplus c_{132}[t]c_{154}[t]c_{93}[t]c_{131}[t] \oplus c_{168}[t]c_{109}[t]$
$c_{76}[t+1] = c_{75}[t] \oplus c_{104}[t]c_{105}[t]c_{94}[t] \oplus c_{162}[t]c_{157}[t] \oplus c_{93}[t]c_{152}[t]c_{164}[t]c_{132}[t]$
$c_{75}[t+1] = c_{91}[t] \oplus c_{74}[t] \oplus c_{116}[t]c_{99}[t]c_{142}[t] \oplus c_{125}[t]c_{126}[t]c_{174}[t]c_{134}[t] \oplus c_{125}[t]c_{152}[t]c_{97}[t] \oplus c_{114}[t]c_{119}[t]c_{177}[t]c_{101}[t]$
$c_{74}[t+1] = c_{73}[t] \oplus c_{141}[t]c_{132}[t]c_{125}[t] \oplus c_{117}[t]c_{93}[t]c_{113}[t] \oplus c_{164}[t]c_{115}[t]c_{148}[t] \oplus c_{148}[t]c_{107}[t]$
$c_{73}[t+1] = c_{72}[t] \oplus c_{166}[t]c_{144}[t]c_{94}[t] \oplus c_{108}[t]c_{152}[t]c_{162}[t]c_{131}[t]$
$c_{72}[t+1] = c_{71}[t] \oplus c_{170}[t]c_{97}[t]c_{100}[t] \oplus c_{160}[t]c_{142}[t]c_{167}[t]c_{137}[t]$
$c_{71}[t+1] = c_{70}[t] \oplus c_{169}[t]c_{118}[t]c_{180}[t]c_{156}[t] \oplus c_{101}[t]c_{127}[t] \oplus c_{125}[t]c_{177}[t]c_{149}[t] \oplus c_{137}[t]c_{96}[t]c_{104}[t]c_{123}[t]$
$c_{70}[t+1] = c_{69}[t] \oplus c_{134}[t]c_{112}[t]c_{148}[t] \oplus c_{169}[t]c_{163}[t]c_{137}[t]c_{180}[t] \oplus c_{109}[t]c_{149}[t]$
$c_{69}[t+1] = c_{68}[t] \oplus c_{139}[t]c_{162}[t] \oplus c_{129}[t]c_{102}[t]$
$c_{68}[t+1] = c_{67}[t] \oplus c_{132}[t]c_{93}[t]c_{168}[t]c_{137}[t] \oplus c_{131}[t]c_{174}[t]c_{98}[t] \oplus c_{118}[t]c_{135}[t]c_{138}[t] \oplus c_{153}[t]c_{111}[t]c_{133}[t]$
$c_{67}[t+1] = c_{66}[t] \oplus c_{111}[t]c_{179}[t]c_{98}[t]c_{138}[t] \oplus c_{163}[t]c_{134}[t]$
$c_{66}[t+1] = c_{65}[t] \oplus c_{139}[t]c_{131}[t] \oplus c_{149}[t]c_{128}[t]c_{117}[t]c_{113}[t] \oplus c_{116}[t]c_{104}[t]c_{121}[t]c_{151}[t] \oplus c_{109}[t]c_{147}[t]c_{159}[t]$
$c_{65}[t+1] = c_{64}[t] \oplus c_{109}[t]c_{123}[t]c_{92}[t]c_{149}[t] \oplus c_{115}[t]c_{179}[t]c_{99}[t]$
$c_{64}[t+1] = c_{63}[t] \oplus c_{106}[t]c_{121}[t] \oplus c_{169}[t]c_{122}[t]c_{154}[t]c_{135}[t] \oplus c_{157}[t]c_{156}[t]c_{166}[t]c_{131}[t]$
$c_{63}[t+1] = c_{62}[t] \oplus c_{121}[t]c_{157}[t] \oplus c_{167}[t]c_{122}[t]c_{123}[t]$
$c_{62}[t+1] = c_{61}[t] \oplus c_{130}[t]c_{151}[t]c_{119}[t]c_{144}[t] \oplus c_{179}[t]c_{119}[t]c_{121}[t]c_{108}[t] \oplus c_{130}[t]c_{180}[t]c_{102}[t]c_{160}[t]$
$c_{61}[t+1] = c_{60}[t] \oplus c_{99}[t]c_{174}[t]c_{149}[t] \oplus c_{165}[t]c_{174}[t]c_{92}[t]c_{98}[t] \oplus c_{116}[t]c_{101}[t]c_{131}[t] \oplus c_{167}[t]c_{157}[t]c_{168}[t]c_{137}[t]$
$c_{60}[t+1] = c_{59}[t] \oplus c_{152}[t]c_{156}[t]c_{96}[t]c_{158}[t] \oplus c_{174}[t]c_{171}[t] \oplus c_{96}[t]c_{113}[t] \oplus c_{108}[t]c_{129}[t]c_{118}[t]c_{136}[t]$
$c_{59}[t+1] = c_{58}[t] \oplus c_{123}[t]c_{104}[t] \oplus c_{180}[t]c_{156}[t]c_{172}[t]c_{166}[t] \oplus c_{122}[t]c_{155}[t]c_{146}[t] \oplus c_{151}[t]c_{127}[t]c_{108}[t]c_{165}[t]$
$c_{58}[t+1] = c_{57}[t] \oplus c_{113}[t]c_{124}[t]c_{106}[t]c_{103}[t] \oplus c_{112}[t]c_{173}[t]c_{99}[t]c_{107}[t] \oplus c_{153}[t]c_{113}[t]c_{96}[t]c_{161}[t]$
$c_{57}[t+1] = c_{56}[t] \oplus c_{155}[t]c_{134}[t]c_{121}[t] \oplus c_{97}[t]c_{158}[t]c_{128}[t] \oplus c_{147}[t]c_{120}[t]$
$c_{56}[t+1] = c_{55}[t] \oplus c_{110}[t]c_{130}[t] \oplus c_{122}[t]c_{161}[t] \oplus c_{96}[t]c_{167}[t] \oplus c_{106}[t]c_{147}[t]$
$c_{55}[t+1] = c_{54}[t] \oplus c_{115}[t]c_{143}[t]c_{172}[t] \oplus c_{107}[t]c_{174}[t]c_{150}[t]$
$c_{54}[t+1] = c_{53}[t] \oplus c_{122}[t]c_{99}[t]c_{172}[t] \oplus c_{145}[t]c_{137}[t]c_{97}[t]c_{119}[t] \oplus c_{146}[t]c_{136}[t]c_{169}[t]c_{172}[t]$
$c_{53}[t+1] = c_{52}[t] \oplus c_{107}[t]c_{99}[t]c_{168}[t] \oplus c_{133}[t]c_{115}[t]$
$c_{52}[t+1] = c_{51}[t] \oplus c_{164}[t]c_{151}[t]c_{106}[t]c_{133}[t] \oplus c_{155}[t]c_{176}[t]$
$c_{51}[t+1] = c_{50}[t] \oplus c_{116}[t]c_{104}[t]c_{167}[t] \oplus c_{150}[t]c_{96}[t] \oplus c_{95}[t]c_{100}[t]c_{96}[t]c_{116}[t] \oplus c_{162}[t]c_{177}[t]c_{124}[t]c_{141}[t]$
$c_{50}[t+1] = c_{91}[t] \oplus c_{49}[t] \oplus c_{127}[t]c_{164}[t]c_{93}[t] \oplus c_{178}[t]c_{151}[t]$
$c_{49}[t+1] = c_{48}[t] \oplus c_{153}[t]c_{139}[t]c_{128}[t] \oplus c_{105}[t]c_{135}[t] \oplus c_{129}[t]c_{136}[t]c_{168}[t]c_{145}[t]$
$c_{48}[t+1] = c_{47}[t] \oplus c_{159}[t]c_{110}[t]c_{98}[t] \oplus c_{125}[t]c_{94}[t]$
$c_{47}[t+1] = c_{46}[t] \oplus c_{167}[t]c_{113}[t] \oplus c_{100}[t]c_{156}[t] \oplus c_{107}[t]c_{145}[t] \oplus c_{169}[t]c_{157}[t]$
$c_{46}[t+1] = c_{91}[t] \oplus c_{45}[t] \oplus c_{173}[t]c_{108}[t]c_{160}[t]c_{99}[t] \oplus c_{144}[t]c_{141}[t]c_{127}[t]c_{178}[t] \oplus c_{135}[t]c_{127}[t]c_{172}[t]c_{174}[t] \oplus c_{138}[t]c_{100}[t]c_{176}[t]$
$c_{45}[t+1] = c_{44}[t] \oplus c_{129}[t]c_{161}[t] \oplus c_{160}[t]c_{147}[t]$
$c_{44}[t+1] = c_{43}[t] \oplus c_{163}[t]c_{97}[t]c_{100}[t] \oplus c_{167}[t]c_{141}[t] \oplus c_{122}[t]c_{135}[t]c_{152}[t]$
$c_{43}[t+1] = c_{42}[t] \oplus c_{176}[t]c_{120}[t]c_{110}[t]c_{140}[t] \oplus c_{160}[t]c_{149}[t]c_{127}[t]c_{136}[t] \oplus c_{159}[t]c_{101}[t]c_{171}[t]c_{115}[t]$
$c_{42}[t+1] = c_{41}[t] \oplus c_{110}[t]c_{160}[t]c_{164}[t]c_{124}[t] \oplus c_{137}[t]c_{92}[t] \oplus c_{118}[t]c_{115}[t]c_{153}[t]$
$c_{41}[t+1] = c_{40}[t] \oplus c_{180}[t]c_{131}[t]c_{112}[t]c_{117}[t] \oplus c_{174}[t]c_{107}[t] \oplus c_{152}[t]c_{101}[t]$
$c_{40}[t+1] = c_{39}[t] \oplus c_{178}[t]c_{102}[t]c_{164}[t]c_{94}[t] \oplus c_{150}[t]c_{147}[t]c_{143}[t] \oplus c_{104}[t]c_{111}[t]c_{180}[t]$
$c_{39}[t+1] = c_{38}[t] \oplus c_{180}[t]c_{175}[t] \oplus c_{163}[t]c_{136}[t]c_{158}[t] \oplus c_{138}[t]c_{163}[t]c_{176}[t]c_{132}[t] \oplus c_{132}[t]c_{165}[t]$
$c_{38}[t+1] = c_{37}[t] \oplus c_{178}[t]c_{180}[t] \oplus c_{128}[t]c_{135}[t]c_{173}[t] \oplus c_{107}[t]c_{98}[t] \oplus c_{105}[t]c_{162}[t]$
$c_{37}[t+1] = c_{36}[t] \oplus c_{98}[t]c_{131}[t] \oplus c_{100}[t]c_{127}[t]$
$c_{36}[t+1] = c_{35}[t] \oplus c_{160}[t]c_{104}[t]c_{176}[t] \oplus c_{101}[t]c_{148}[t]c_{107}[t] \oplus c_{99}[t]c_{144}[t]c_{96}[t] \oplus c_{97}[t]c_{137}[t]$
$c_{35}[t+1] = c_{34}[t] \oplus c_{159}[t]c_{160}[t]c_{166}[t] \oplus c_{175}[t]c_{136}[t] \oplus c_{128}[t]c_{118}[t] \oplus c_{144}[t]c_{151}[t]c_{104}[t]$
$c_{34}[t+1] = c_{33}[t] \oplus c_{170}[t]c_{107}[t] \oplus c_{103}[t]c_{173}[t] \oplus c_{99}[t]c_{127}[t]c_{94}[t]c_{176}[t] \oplus c_{113}[t]c_{133}[t]$
$c_{33}[t+1] = c_{32}[t] \oplus c_{113}[t]c_{134}[t] \oplus c_{124}[t]c_{108}[t]$
$c_{32}[t+1] = c_{31}[t] \oplus c_{119}[t]c_{112}[t]c_{176}[t]c_{140}[t] \oplus c_{141}[t]c_{99}[t]c_{101}[t] \oplus c_{112}[t]c_{179}[t]c_{165}[t]c_{123}[t]$
$c_{31}[t+1] = c_{91}[t] \oplus c_{139}[t]c_{110}[t]c_{122}[t]c_{152}[t] \oplus c_{127}[t]c_{116}[t]c_{134}[t] \oplus c_{120}[t]c_{137}[t]c_{170}[t] \oplus c_{125}[t]c_{127}[t]c_{123}[t]c_{93}[t]$
$c_{30}[t+1] = c_{29}[t] \oplus c_{60}[t]c_{47}[t]c_{38}[t] \oplus c_{69}[t]c_{76}[t] \oplus c_{70}[t]c_{91}[t]c_{67}[t]$
$c_{29}[t+1] = c_{28}[t] \oplus c_{90}[t]c_{83}[t]c_{67}[t] \oplus c_{48}[t]c_{61}[t] \oplus c_{57}[t]c_{46}[t] \oplus c_{75}[t]c_{48}[t]$
$c_{28}[t+1] = c_{27}[t] \oplus c_{82}[t]c_{43}[t] \oplus c_{38}[t]c_{52}[t] \oplus c_{63}[t]c_{79}[t]c_{34}[t] \oplus c_{52}[t]c_{45}[t]c_{80}[t]c_{44}[t]$

$c_{27}[t+1] = c_{26}[t] \oplus c_{42}[t]c_{38}[t]c_{71}[t] \oplus c_{84}[t]c_{49}[t]c_{80}[t]c_{47}[t]$
$c_{26}[t+1] = c_{25}[t] \oplus c_{86}[t]c_{70}[t]c_{47}[t] \oplus c_{59}[t]c_{41}[t]c_{74}[t]c_{42}[t] \oplus c_{63}[t]c_{75}[t] \oplus c_{89}[t]c_{46}[t]c_{37}[t]$
$c_{25}[t+1] = c_{24}[t] \oplus c_{44}[t]c_{63}[t] \oplus c_{62}[t]c_{52}[t]c_{44}[t] \oplus c_{74}[t]c_{56}[t] \oplus c_{45}[t]c_{31}[t]$
$c_{24}[t+1] = c_{23}[t] \oplus c_{47}[t]c_{64}[t] \oplus c_{50}[t]c_{37}[t] \oplus c_{77}[t]c_{34}[t] \oplus c_{58}[t]c_{33}[t]c_{34}[t]c_{73}[t]$
$c_{23}[t+1] = c_{22}[t] \oplus c_{86}[t]c_{53}[t] \oplus c_{58}[t]c_{89}[t]c_{81}[t] \oplus c_{43}[t]c_{31}[t]$
$c_{22}[t+1] = c_{21}[t] \oplus c_{65}[t]c_{44}[t]c_{43}[t]c_{56}[t] \oplus c_{81}[t]c_{51}[t]c_{76}[t]$
$c_{21}[t+1] = c_{20}[t] \oplus c_{76}[t]c_{77}[t]c_{49}[t] \oplus c_{37}[t]c_{62}[t]c_{67}[t] \oplus c_{49}[t]c_{34}[t] \oplus c_{71}[t]c_{80}[t]c_{61}[t]c_{74}[t]$
$c_{20}[t+1] = c_{19}[t] \oplus c_{71}[t]c_{61}[t]c_{79}[t]c_{42}[t] \oplus c_{84}[t]c_{41}[t]$
$c_{19}[t+1] = c_{18}[t] \oplus c_{56}[t]c_{37}[t]c_{31}[t]c_{90}[t] \oplus c_{47}[t]c_{76}[t]c_{77}[t] \oplus c_{56}[t]c_{53}[t]c_{74}[t]c_{32}[t]$
$c_{18}[t+1] = c_{17}[t] \oplus c_{58}[t]c_{86}[t] \oplus c_{38}[t]c_{73}[t]c_{50}[t]c_{68}[t] \oplus c_{48}[t]c_{65}[t]c_{91}[t] \oplus c_{56}[t]c_{66}[t]$
$c_{17}[t+1] = c_{16}[t] \oplus c_{32}[t]c_{43}[t]c_{55}[t] \oplus c_{57}[t]c_{40}[t]c_{52}[t] \oplus c_{74}[t]c_{91}[t]c_{43}[t] \oplus c_{72}[t]c_{60}[t]c_{37}[t]$
$c_{16}[t+1] = c_{15}[t] \oplus c_{50}[t]c_{62}[t] \oplus c_{41}[t]c_{66}[t]c_{46}[t] \oplus c_{58}[t]c_{84}[t] \oplus c_{80}[t]c_{36}[t]$
$c_{15}[t+1] = c_{14}[t] \oplus c_{43}[t]c_{45}[t]c_{48}[t]c_{66}[t] \oplus c_{64}[t]c_{41}[t]c_{67}[t]c_{86}[t] \oplus c_{50}[t]c_{41}[t]c_{47}[t]$
$c_{14}[t+1] = c_{13}[t] \oplus c_{60}[t]c_{82}[t] \oplus c_{47}[t]c_{64}[t] \oplus c_{77}[t]c_{85}[t]$
$c_{13}[t+1] = c_{12}[t] \oplus c_{52}[t]c_{40}[t]c_{50}[t]c_{53}[t] \oplus c_{58}[t]c_{67}[t]c_{87}[t]$
$c_{12}[t+1] = c_{11}[t] \oplus c_{32}[t]c_{38}[t]c_{58}[t] \oplus c_{82}[t]c_{86}[t]c_{59}[t]c_{81}[t] \oplus c_{63}[t]c_{54}[t] \oplus c_{39}[t]c_{32}[t]c_{52}[t]c_{82}[t]$
$c_{11}[t+1] = c_{10}[t] \oplus c_{82}[t]c_{81}[t]c_{59}[t] \oplus c_{80}[t]c_{86}[t]c_{60}[t]c_{62}[t]$
$c_{10}[t+1] = c_{9}[t] \oplus c_{33}[t]c_{40}[t]c_{56}[t] \oplus c_{80}[t]c_{58}[t]c_{81}[t]c_{74}[t] \oplus c_{58}[t]c_{83}[t]c_{64}[t]c_{57}[t] \oplus c_{68}[t]c_{59}[t]c_{36}[t]c_{33}[t]$
$c_{9}[t+1] = c_{8}[t] \oplus c_{68}[t]c_{59}[t] \oplus c_{65}[t]c_{61}[t]c_{74}[t]c_{59}[t] \oplus c_{55}[t]c_{43}[t]$
$c_{8}[t+1] = c_{7}[t] \oplus c_{34}[t]c_{68}[t] \oplus c_{56}[t]c_{66}[t]$
$c_{7}[t+1] = c_{6}[t] \oplus c_{55}[t]c_{31}[t] \oplus c_{68}[t]c_{48}[t]c_{74}[t] \oplus c_{39}[t]c_{32}[t] \oplus c_{47}[t]c_{33}[t]$
$c_{6}[t+1] = c_{5}[t] \oplus c_{86}[t]c_{49}[t] \oplus c_{67}[t]c_{36}[t]c_{81}[t]c_{42}[t] \oplus c_{89}[t]c_{64}[t] \oplus c_{62}[t]c_{41}[t]c_{52}[t]c_{80}[t]$
$c_{5}[t+1] = c_{4}[t] \oplus c_{62}[t]c_{64}[t]c_{67}[t]c_{66}[t] \oplus c_{48}[t]c_{60}[t]c_{74}[t]$
$c_{4}[t+1] = c_{3}[t] \oplus c_{83}[t]c_{42}[t]c_{47}[t] \oplus c_{32}[t]c_{64}[t] \oplus c_{76}[t]c_{53}[t]c_{75}[t] \oplus c_{68}[t]c_{87}[t]$
$c_{3}[t+1] = c_{2}[t] \oplus c_{30}[t] \oplus c_{83}[t]c_{81}[t]c_{74}[t]c_{89}[t] \oplus c_{56}[t]c_{60}[t] \oplus c_{85}[t]c_{75}[t] \oplus c_{67}[t]c_{73}[t]c_{36}[t]c_{49}[t]$
$c_{2}[t+1] = c_{1}[t] \oplus c_{30}[t] \oplus c_{71}[t]c_{87}[t]c_{60}[t]c_{83}[t] \oplus c_{67}[t]c_{62}[t]c_{87}[t]c_{66}[t] \oplus c_{49}[t]c_{69}[t]c_{90}[t] \oplus c_{90}[t]c_{51}[t]$
$c_{1}[t+1] = c_{30}[t] \oplus c_{0}[t] \oplus c_{69}[t]c_{79}[t] \oplus c_{47}[t]c_{61}[t]c_{41}[t] \oplus c_{42}[t]c_{71}[t]c_{62}[t]c_{51}[t] \oplus c_{42}[t]c_{70}[t]c_{32}[t]$
$c_{0}[t+1] = c_{30}[t] \oplus c_{73}[t]c_{89}[t] \oplus c_{38}[t]c_{90}[t]c_{69}[t]c_{55}[t] \oplus c_{69}[t]c_{72}[t]c_{75}[t] \oplus c_{70}[t]c_{59}[t]c_{88}[t]$

A.2 ANF for 162-bit CMPR

$c_{161}[t+1] = c_{160}[t]$
$c_{160}[t+1] = c_{159}[t]$
$c_{159}[t+1] = c_{158}[t]$
$c_{158}[t+1] = c_{157}[t]$
$c_{157}[t+1] = c_{156}[t]$
$c_{156}[t+1] = c_{155}[t]$
$c_{155}[t+1] = c_{154}[t]$
$c_{154}[t+1] = c_{153}[t]$
$c_{153}[t+1] = c_{152}[t]$
$c_{152}[t+1] = c_{151}[t]$
$c_{151}[t+1] = c_{150}[t]$
$c_{150}[t+1] = c_{149}[t]$
$c_{149}[t+1] = c_{148}[t]$
$c_{148}[t+1] = c_{147}[t]$
$c_{147}[t+1] = c_{146}[t]$
$c_{146}[t+1] = c_{145}[t]$
$c_{145}[t+1] = c_{144}[t]$
$c_{144}[t+1] = c_{161}[t] \oplus c_{143}[t]$
$c_{143}[t+1] = c_{142}[t]$
$c_{142}[t+1] = c_{141}[t]$
$c_{141}[t+1] = c_{140}[t]$
$c_{140}[t+1] = c_{139}[t]$
$c_{139}[t+1] = c_{161}[t] \oplus c_{138}[t]$
$c_{138}[t+1] = c_{137}[t]$
$c_{137}[t+1] = c_{136}[t]$
$c_{136}[t+1] = c_{135}[t]$

$$c_{135}[t+1] = c_{134}[t]$$
$$c_{134}[t+1] = c_{133}[t]$$
$$c_{133}[t+1] = c_{132}[t]$$
$$c_{132}[t+1] = c_{131}[t]$$
$$c_{131}[t+1] = c_{130}[t]$$
$$c_{130}[t+1] = c_{129}[t]$$
$$c_{129}[t+1] = c_{128}[t]$$
$$c_{128}[t+1] = c_{127}[t]$$
$$c_{127}[t+1] = c_{126}[t]$$
$$c_{126}[t+1] = c_{125}[t]$$
$$c_{125}[t+1] = c_{124}[t]$$
$$c_{124}[t+1] = c_{123}[t]$$
$$c_{123}[t+1] = c_{122}[t]$$
$$c_{122}[t+1] = c_{121}[t]$$
$$c_{121}[t+1] = c_{120}[t]$$
$$c_{120}[t+1] = c_{119}[t]$$
$$c_{119}[t+1] = c_{118}[t]$$
$$c_{118}[t+1] = c_{117}[t]$$
$$c_{117}[t+1] = c_{116}[t]$$
$$c_{116}[t+1] = c_{115}[t]$$
$$c_{115}[t+1] = c_{114}[t]$$
$$c_{114}[t+1] = c_{113}[t]$$
$$c_{113}[t+1] = c_{112}[t]$$
$$c_{112}[t+1] = c_{111}[t]$$
$$c_{111}[t+1] = c_{110}[t]$$
$$c_{110}[t+1] = c_{109}[t]$$
$$c_{109}[t+1] = c_{108}[t]$$
$$c_{108}[t+1] = c_{107}[t]$$
$$c_{107}[t+1] = c_{106}[t]$$
$$c_{106}[t+1] = c_{105}[t]$$
$$c_{105}[t+1] = c_{104}[t]$$
$$c_{104}[t+1] = c_{103}[t]$$
$$c_{103}[t+1] = c_{102}[t]$$
$$c_{102}[t+1] = c_{101}[t]$$
$$c_{101}[t+1] = c_{100}[t]$$
$$c_{100}[t+1] = c_{99}[t]$$
$$c_{99}[t+1] = c_{98}[t]$$
$$c_{98}[t+1] = c_{97}[t]$$
$$c_{97}[t+1] = c_{96}[t]$$
$$c_{96}[t+1] = c_{95}[t]$$
$$c_{95}[t+1] = c_{161}[t] \oplus c_{94}[t]$$
$$c_{94}[t+1] = c_{93}[t]$$
$$c_{93}[t+1] = c_{92}[t]$$
$$c_{92}[t+1] = c_{91}[t]$$
$$c_{91}[t+1] = c_{90}[t]$$
$$c_{90}[t+1] = c_{89}[t]$$
$$c_{89}[t+1] = c_{88}[t]$$
$$c_{88}[t+1] = c_{87}[t]$$
$$c_{87}[t+1] = c_{86}[t]$$
$$c_{86}[t+1] = c_{85}[t]$$
$$c_{85}[t+1] = c_{84}[t]$$
$$c_{84}[t+1] = c_{161}[t] \oplus c_{83}[t]$$
$$c_{83}[t+1] = c_{82}[t]$$
$$c_{82}[t+1] = c_{81}[t]$$
$$c_{81}[t+1] = c_{80}[t]$$
$$c_{80}[t+1] = c_{79}[t]$$
$$c_{79}[t+1] = c_{78}[t]$$

$c_{78}[t+1] = c_{161}[t] \oplus c_{77}[t]$
$c_{77}[t+1] = c_{76}[t]$
$c_{76}[t+1] = c_{75}[t]$
$c_{75}[t+1] = c_{74}[t]$
$c_{74}[t+1] = c_{73}[t]$
$c_{73}[t+1] = c_{72}[t]$
$c_{72}[t+1] = c_{71}[t]$
$c_{71}[t+1] = c_{70}[t]$
$c_{70}[t+1] = c_{69}[t]$
$c_{69}[t+1] = c_{68}[t]$
$c_{68}[t+1] = c_{67}[t]$
$c_{67}[t+1] = c_{66}[t]$
$c_{66}[t+1] = c_{65}[t]$
$c_{65}[t+1] = c_{64}[t]$
$c_{64}[t+1] = c_{63}[t]$
$c_{63}[t+1] = c_{62}[t]$
$c_{62}[t+1] = c_{61}[t]$
$c_{61}[t+1] = c_{60}[t]$
$c_{60}[t+1] = c_{59}[t]$
$c_{59}[t+1] = c_{58}[t]$
$c_{58}[t+1] = c_{57}[t]$
$c_{57}[t+1] = c_{56}[t]$
$c_{56}[t+1] = c_{55}[t]$
$c_{55}[t+1] = c_{161}[t]$
$c_{54}[t+1] = c_{53}[t] \oplus c_{153}[t]c_{72}[t]c_{77}[t] \oplus c_{71}[t]c_{102}[t]c_{132}[t]c_{141}[t] \oplus c_{59}[t]c_{108}[t]$
$c_{53}[t+1] = c_{52}[t] \oplus c_{67}[t]c_{104}[t]c_{99}[t]c_{160}[t] \oplus c_{97}[t]c_{145}[t]$
$c_{52}[t+1] = c_{51}[t] \oplus c_{160}[t]c_{141}[t]c_{143}[t]c_{94}[t] \oplus c_{146}[t]c_{160}[t]c_{100}[t] \oplus c_{81}[t]c_{155}[t] \oplus c_{114}[t]c_{135}[t]c_{140}[t]$
$c_{51}[t+1] = c_{50}[t] \oplus c_{59}[t]c_{134}[t] \oplus c_{148}[t]c_{60}[t]$
$c_{50}[t+1] = c_{49}[t] \oplus c_{90}[t]c_{96}[t]c_{106}[t] \oplus c_{72}[t]c_{67}[t] \oplus c_{62}[t]c_{75}[t]c_{101}[t]c_{153}[t] \oplus c_{106}[t]c_{63}[t]c_{156}[t]$
$c_{49}[t+1] = c_{48}[t] \oplus c_{59}[t]c_{118}[t] \oplus c_{104}[t]c_{135}[t]c_{157}[t]c_{68}[t] \oplus c_{119}[t]c_{79}[t]c_{135}[t]c_{66}[t] \oplus c_{130}[t]c_{127}[t]c_{60}[t]c_{123}[t]$
$c_{48}[t+1] = c_{47}[t] \oplus c_{89}[t]c_{138}[t] \oplus c_{78}[t]c_{68}[t]$
$c_{47}[t+1] = c_{46}[t] \oplus c_{144}[t]c_{67}[t]c_{146}[t]c_{87}[t] \oplus c_{73}[t]c_{153}[t] \oplus c_{86}[t]c_{108}[t]c_{107}[t]c_{68}[t]$
$c_{46}[t+1] = c_{45}[t] \oplus c_{133}[t]c_{77}[t] \oplus c_{159}[t]c_{118}[t]c_{72}[t] \oplus c_{156}[t]c_{105}[t] \oplus c_{150}[t]c_{156}[t]c_{152}[t]c_{134}[t]$
$c_{45}[t+1] = c_{44}[t] \oplus c_{131}[t]c_{90}[t]c_{109}[t]c_{129}[t] \oplus c_{149}[t]c_{68}[t]c_{77}[t]c_{127}[t] \oplus c_{105}[t]c_{112}[t] \oplus c_{91}[t]c_{93}[t]c_{153}[t]c_{92}[t]$
$c_{44}[t+1] = c_{43}[t] \oplus c_{127}[t]c_{78}[t]c_{148}[t] \oplus c_{78}[t]c_{55}[t]c_{62}[t]c_{152}[t]$
$c_{43}[t+1] = c_{42}[t] \oplus c_{152}[t]c_{153}[t]c_{160}[t]c_{75}[t] \oplus c_{108}[t]c_{88}[t]c_{100}[t]c_{107}[t] \oplus c_{63}[t]c_{96}[t]c_{100}[t]$
$c_{42}[t+1] = c_{41}[t] \oplus c_{68}[t]c_{119}[t]c_{135}[t]c_{98}[t] \oplus c_{158}[t]c_{77}[t]c_{92}[t]$
$c_{41}[t+1] = c_{40}[t] \oplus c_{100}[t]c_{142}[t]c_{118}[t] \oplus c_{123}[t]c_{64}[t]c_{93}[t]c_{110}[t] \oplus c_{149}[t]c_{110}[t]c_{62}[t]c_{128}[t] \oplus c_{107}[t]c_{125}[t]$
$c_{40}[t+1] = c_{39}[t] \oplus c_{76}[t]c_{134}[t]c_{98}[t] \oplus c_{65}[t]c_{93}[t]$
$c_{39}[t+1] = c_{38}[t] \oplus c_{132}[t]c_{93}[t]c_{60}[t] \oplus c_{151}[t]c_{161}[t]c_{118}[t]$
$c_{38}[t+1] = c_{37}[t] \oplus c_{115}[t]c_{125}[t]c_{105}[t]c_{134}[t] \oplus c_{76}[t]c_{98}[t] \oplus c_{90}[t]c_{61}[t]c_{101}[t]c_{115}[t]$
$c_{37}[t+1] = c_{36}[t] \oplus c_{133}[t]c_{86}[t] \oplus c_{125}[t]c_{145}[t]c_{105}[t]c_{84}[t]$
$c_{36}[t+1] = c_{35}[t] \oplus c_{84}[t]c_{90}[t]c_{55}[t] \oplus c_{160}[t]c_{64}[t]c_{138}[t]c_{95}[t] \oplus c_{144}[t]c_{139}[t]$
$c_{35}[t+1] = c_{34}[t] \oplus c_{98}[t]c_{99}[t]c_{86}[t] \oplus c_{95}[t]c_{72}[t] \oplus c_{69}[t]c_{123}[t]c_{137}[t] \oplus c_{158}[t]c_{112}[t]c_{58}[t]$
$c_{34}[t+1] = c_{33}[t] \oplus c_{87}[t]c_{91}[t]c_{120}[t] \oplus c_{123}[t]c_{158}[t]$
$c_{33}[t+1] = c_{32}[t] \oplus c_{58}[t]c_{120}[t] \oplus c_{87}[t]c_{142}[t]c_{86}[t] \oplus c_{79}[t]c_{126}[t] \oplus c_{99}[t]c_{76}[t]c_{96}[t]$
$c_{32}[t+1] = c_{31}[t] \oplus c_{63}[t]c_{146}[t]c_{107}[t] \oplus c_{94}[t]c_{139}[t]c_{110}[t]$
$c_{31}[t+1] = c_{30}[t] \oplus c_{131}[t]c_{126}[t] \oplus c_{116}[t]c_{99}[t] \oplus c_{60}[t]c_{121}[t]c_{122}[t]c_{110}[t]$
$c_{30}[t+1] = c_{29}[t] \oplus c_{71}[t]c_{142}[t]c_{108}[t]c_{69}[t] \oplus c_{100}[t]c_{158}[t]c_{99}[t]c_{125}[t]$
$c_{29}[t+1] = c_{28}[t] \oplus c_{64}[t]c_{79}[t]c_{119}[t] \oplus c_{106}[t]c_{58}[t]$
$c_{28}[t+1] = c_{27}[t] \oplus c_{57}[t]c_{74}[t]c_{84}[t]c_{65}[t] \oplus c_{55}[t]c_{147}[t]c_{57}[t]c_{98}[t] \oplus c_{102}[t]c_{138}[t]$
$c_{27}[t+1] = c_{26}[t] \oplus c_{54}[t] \oplus c_{151}[t]c_{79}[t]c_{84}[t] \oplus c_{99}[t]c_{110}[t]c_{80}[t]c_{62}[t] \oplus c_{149}[t]c_{130}[t] \oplus c_{98}[t]c_{70}[t]c_{63}[t]$
$c_{26}[t+1] = c_{25}[t] \oplus c_{54}[t] \oplus c_{95}[t]c_{77}[t]c_{61}[t]c_{87}[t] \oplus c_{65}[t]c_{84}[t] \oplus c_{137}[t]c_{111}[t]c_{158}[t]c_{118}[t] \oplus c_{139}[t]c_{156}[t]c_{129}[t]$
$c_{25}[t+1] = c_{54}[t] \oplus c_{24}[t] \oplus c_{108}[t]c_{127}[t]c_{95}[t] \oplus c_{108}[t]c_{112}[t]c_{84}[t]c_{151}[t] \oplus c_{144}[t]c_{62}[t]c_{147}[t]$
$c_{24}[t+1] = c_{54}[t] \oplus c_{140}[t]c_{74}[t]c_{92}[t]c_{95}[t] \oplus c_{155}[t]c_{66}[t]c_{105}[t] \oplus c_{143}[t]c_{117}[t]c_{113}[t]$
$c_{23}[t+1] = c_{22}[t] \oplus c_{28}[t]c_{37}[t]c_{39}[t]c_{44}[t] \oplus c_{27}[t]c_{38}[t] \oplus c_{27}[t]c_{42}[t]c_{52}[t]c_{37}[t] \oplus c_{38}[t]c_{48}[t]c_{29}[t]$
$c_{22}[t+1] = c_{21}[t] \oplus c_{38}[t]c_{53}[t]c_{39}[t] \oplus c_{41}[t]c_{36}[t] \oplus c_{53}[t]c_{46}[t]$

$$
\begin{aligned}
c_{21}[t+1] &= c_{20}[t] \oplus c_{41}[t]c_{33}[t]c_{47}[t]c_{45}[t] \oplus c_{48}[t]c_{51}[t]c_{45}[t]c_{34}[t] \\
c_{20}[t+1] &= c_{19}[t] \oplus c_{53}[t]c_{34}[t]c_{42}[t] \oplus c_{47}[t]c_{37}[t]c_{45}[t]c_{33}[t] \oplus c_{53}[t]c_{42}[t]c_{33}[t] \oplus c_{40}[t]c_{39}[t]c_{35}[t] \\
c_{19}[t+1] &= c_{18}[t] \oplus c_{39}[t]c_{47}[t]c_{41}[t]c_{53}[t] \oplus c_{51}[t]c_{34}[t] \oplus c_{48}[t]c_{39}[t] \oplus c_{49}[t]c_{46}[t] \\
c_{18}[t+1] &= c_{17}[t] \oplus c_{50}[t]c_{24}[t]c_{37}[t] \oplus c_{30}[t]c_{35}[t]c_{36}[t] \oplus c_{53}[t]c_{35}[t] \\
c_{17}[t+1] &= c_{16}[t] \oplus c_{48}[t]c_{35}[t]c_{41}[t]c_{26}[t] \oplus c_{31}[t]c_{50}[t]c_{40}[t] \\
c_{16}[t+1] &= c_{15}[t] \oplus c_{31}[t]c_{41}[t]c_{37}[t] \oplus c_{27}[t]c_{49}[t]c_{47}[t]c_{26}[t] \oplus c_{34}[t]c_{41}[t] \oplus c_{50}[t]c_{44}[t]c_{39}[t]c_{48}[t] \\
c_{15}[t+1] &= c_{23}[t] \oplus c_{14}[t] \oplus c_{27}[t]c_{48}[t]c_{36}[t] \oplus c_{50}[t]c_{45}[t]c_{27}[t]c_{35}[t] \\
c_{14}[t+1] &= c_{13}[t] \oplus c_{23}[t] \oplus c_{37}[t]c_{49}[t]c_{44}[t] \oplus c_{38}[t]c_{39}[t]c_{34}[t]c_{35}[t] \oplus c_{44}[t]c_{33}[t]c_{39}[t] \\
c_{13}[t+1] &= c_{12}[t] \oplus c_{23}[t] \oplus c_{34}[t]c_{48}[t]c_{31}[t] \oplus c_{32}[t]c_{27}[t]c_{47}[t] \oplus c_{31}[t]c_{32}[t]c_{50}[t] \oplus c_{29}[t]c_{28}[t]c_{42}[t] \\
c_{12}[t+1] &= c_{11}[t] \oplus c_{36}[t]c_{42}[t]c_{52}[t] \oplus c_{53}[t]c_{39}[t]c_{26}[t] \oplus c_{48}[t]c_{41}[t]c_{39}[t]c_{35}[t] \oplus c_{32}[t]c_{52}[t]c_{37}[t]c_{44}[t] \\
c_{11}[t+1] &= c_{10}[t] \oplus c_{23}[t] \oplus c_{43}[t]c_{54}[t]c_{31}[t] \oplus c_{37}[t]c_{48}[t]c_{40}[t]c_{45}[t] \oplus c_{42}[t]c_{47}[t] \oplus c_{33}[t]c_{37}[t]c_{39}[t]c_{49}[t] \\
c_{10}[t+1] &= c_{9}[t] \oplus c_{23}[t] \oplus c_{35}[t]c_{24}[t]c_{54}[t]c_{41}[t] \oplus c_{39}[t]c_{43}[t]c_{48}[t]c_{51}[t] \oplus c_{36}[t]c_{35}[t]c_{24}[t]c_{39}[t] \\
c_{9}[t+1] &= c_{8}[t] \oplus c_{34}[t]c_{31}[t]c_{43}[t] \oplus c_{53}[t]c_{26}[t]c_{36}[t]c_{45}[t] \oplus c_{36}[t]c_{29}[t]c_{37}[t] \\
c_{8}[t+1] &= c_{7}[t] \oplus c_{35}[t]c_{46}[t] \oplus c_{26}[t]c_{34}[t]c_{31}[t] \\
c_{7}[t+1] &= c_{23}[t] \oplus c_{30}[t]c_{39}[t]c_{52}[t] \oplus c_{51}[t]c_{36}[t]c_{42}[t] \oplus c_{47}[t]c_{52}[t]c_{48}[t]c_{24}[t] \oplus c_{53}[t]c_{52}[t]c_{38}[t] \\
c_{6}[t+1] &= c_{5}[t] \oplus c_{6}[t] \oplus c_{21}[t]c_{18}[t]c_{22}[t] \oplus c_{21}[t]c_{8}[t]c_{11}[t] \oplus c_{18}[t]c_{15}[t]c_{14}[t] \oplus c_{18}[t]c_{23}[t]c_{19}[t] \\
c_{5}[t+1] &= c_{4}[t] \oplus c_{6}[t] \oplus c_{23}[t]c_{19}[t]c_{9}[t] \oplus c_{19}[t]c_{21}[t] \\
c_{4}[t+1] &= c_{3}[t] \oplus c_{6}[t] \oplus c_{21}[t]c_{14}[t]c_{7}[t] \oplus c_{7}[t]c_{16}[t]c_{17}[t]c_{23}[t] \\
c_{3}[t+1] &= c_{2}[t] \oplus c_{18}[t]c_{13}[t]c_{21}[t]c_{11}[t] \oplus c_{19}[t]c_{17}[t] \oplus c_{23}[t]c_{12}[t]c_{22}[t]c_{18}[t] \\
c_{2}[t+1] &= c_{6}[t] \oplus c_{19}[t]c_{23}[t]c_{7}[t]c_{12}[t] \oplus c_{14}[t]c_{13}[t] \oplus c_{19}[t]c_{9}[t] \\
c_{1}[t+1] &= c_{1}[t] \oplus c_{0}[t] \oplus c_{2}[t]c_{5}[t]c_{4}[t]c_{6}[t] \oplus c_{2}[t]c_{4}[t] \\
c_{0}[t+1] &= c_{1}[t] \oplus c_{3}[t]c_{2}[t]c_{5}[t] \oplus c_{2}[t]c_{4}[t]c_{5}[t]c_{6}[t] \oplus c_{3}[t]c_{4}[t] \oplus c_{3}[t]c_{5}[t]
\end{aligned}
$$

A.3 ANF for 170-bit CMPR

$c_{169}[t+1] = c_{168}[t]$
$c_{168}[t+1] = c_{167}[t]$
$c_{167}[t+1] = c_{166}[t]$
$c_{166}[t+1] = c_{165}[t]$
$c_{165}[t+1] = c_{164}[t]$
$c_{164}[t+1] = c_{163}[t]$
$c_{163}[t+1] = c_{162}[t]$
$c_{162}[t+1] = c_{161}[t]$
$c_{161}[t+1] = c_{160}[t]$
$c_{160}[t+1] = c_{159}[t]$
$c_{159}[t+1] = c_{158}[t]$
$c_{158}[t+1] = c_{157}[t]$
$c_{157}[t+1] = c_{156}[t]$
$c_{156}[t+1] = c_{155}[t]$
$c_{155}[t+1] = c_{154}[t]$
$c_{154}[t+1] = c_{153}[t]$
$c_{153}[t+1] = c_{152}[t]$
$c_{152}[t+1] = c_{151}[t]$
$c_{151}[t+1] = c_{150}[t]$
$c_{150}[t+1] = c_{149}[t]$
$c_{149}[t+1] = c_{148}[t]$
$c_{148}[t+1] = c_{147}[t]$
$c_{147}[t+1] = c_{146}[t]$
$c_{146}[t+1] = c_{145}[t]$
$c_{145}[t+1] = c_{144}[t]$
$c_{144}[t+1] = c_{143}[t]$
$c_{143}[t+1] = c_{142}[t]$
$c_{142}[t+1] = c_{141}[t]$
$c_{141}[t+1] = c_{140}[t]$
$c_{140}[t+1] = c_{139}[t]$
$c_{139}[t+1] = c_{138}[t]$
$c_{138}[t+1] = c_{137}[t]$

$c_{137}[t+1] = c_{136}[t]$
$c_{136}[t+1] = c_{135}[t]$
$c_{135}[t+1] = c_{134}[t]$
$c_{134}[t+1] = c_{133}[t]$
$c_{133}[t+1] = c_{132}[t]$
$c_{132}[t+1] = c_{131}[t]$
$c_{131}[t+1] = c_{130}[t]$
$c_{130}[t+1] = c_{129}[t]$
$c_{129}[t+1] = c_{128}[t]$
$c_{128}[t+1] = c_{127}[t]$
$c_{127}[t+1] = c_{126}[t]$
$c_{126}[t+1] = c_{125}[t]$
$c_{125}[t+1] = c_{124}[t]$
$c_{124}[t+1] = c_{123}[t]$
$c_{123}[t+1] = c_{122}[t]$
$c_{122}[t+1] = c_{121}[t]$
$c_{121}[t+1] = c_{120}[t]$
$c_{120}[t+1] = c_{119}[t]$
$c_{119}[t+1] = c_{118}[t]$
$c_{118}[t+1] = c_{117}[t]$
$c_{117}[t+1] = c_{116}[t]$
$c_{116}[t+1] = c_{115}[t]$
$c_{115}[t+1] = c_{114}[t]$
$c_{114}[t+1] = c_{113}[t]$
$c_{113}[t+1] = c_{112}[t]$
$c_{112}[t+1] = c_{111}[t]$
$c_{111}[t+1] = c_{110}[t]$
$c_{110}[t+1] = c_{109}[t]$
$c_{109}[t+1] = c_{108}[t]$
$c_{108}[t+1] = c_{107}[t]$
$c_{107}[t+1] = c_{106}[t]$
$c_{106}[t+1] = c_{105}[t]$
$c_{105}[t+1] = c_{104}[t]$
$c_{104}[t+1] = c_{103}[t]$
$c_{103}[t+1] = c_{102}[t]$
$c_{102}[t+1] = c_{101}[t]$
$c_{101}[t+1] = c_{100}[t]$
$c_{100}[t+1] = c_{99}[t]$
$c_{99}[t+1] = c_{98}[t]$
$c_{98}[t+1] = c_{97}[t]$
$c_{97}[t+1] = c_{96}[t] \oplus c_{169}[t]$
$c_{96}[t+1] = c_{95}[t]$
$c_{95}[t+1] = c_{94}[t]$
$c_{94}[t+1] = c_{93}[t]$
$c_{93}[t+1] = c_{92}[t]$
$c_{92}[t+1] = c_{91}[t]$
$c_{91}[t+1] = c_{90}[t]$
$c_{90}[t+1] = c_{89}[t]$
$c_{89}[t+1] = c_{88}[t]$
$c_{88}[t+1] = c_{87}[t] \oplus c_{169}[t]$
$c_{87}[t+1] = c_{86}[t]$
$c_{86}[t+1] = c_{85}[t]$
$c_{85}[t+1] = c_{84}[t]$
$c_{84}[t+1] = c_{83}[t]$
$c_{83}[t+1] = c_{82}[t]$
$c_{82}[t+1] = c_{81}[t]$
$c_{81}[t+1] = c_{80}[t]$

$c_{80}[t+1] = c_{79}[t]$
$c_{79}[t+1] = c_{78}[t]$
$c_{78}[t+1] = c_{77}[t]$
$c_{77}[t+1] = c_{76}[t]$
$c_{76}[t+1] = c_{75}[t]$
$c_{75}[t+1] = c_{74}[t]$
$c_{74}[t+1] = c_{73}[t]$
$c_{73}[t+1] = c_{72}[t]$
$c_{72}[t+1] = c_{71}[t]$
$c_{71}[t+1] = c_{70}[t]$
$c_{70}[t+1] = c_{69}[t]$
$c_{69}[t+1] = c_{68}[t]$
$c_{68}[t+1] = c_{67}[t]$
$c_{67}[t+1] = c_{66}[t]$
$c_{66}[t+1] = c_{65}[t]$
$c_{65}[t+1] = c_{64}[t]$
$c_{64}[t+1] = c_{63}[t]$
$c_{63}[t+1] = c_{62}[t]$
$c_{62}[t+1] = c_{61}[t]$
$c_{61}[t+1] = c_{60}[t]$
$c_{60}[t+1] = c_{59}[t]$
$c_{59}[t+1] = c_{58}[t]$
$c_{58}[t+1] = c_{57}[t]$
$c_{57}[t+1] = c_{56}[t]$
$c_{56}[t+1] = c_{55}[t] \oplus c_{169}[t]$
$c_{55}[t+1] = c_{54}[t]$
$c_{54}[t+1] = c_{53}[t]$
$c_{53}[t+1] = c_{52}[t]$
$c_{52}[t+1] = c_{51}[t]$
$c_{51}[t+1] = c_{50}[t]$
$c_{50}[t+1] = c_{49}[t]$
$c_{49}[t+1] = c_{48}[t]$
$c_{48}[t+1] = c_{47}[t]$
$c_{47}[t+1] = c_{46}[t]$
$c_{46}[t+1] = c_{45}[t]$
$c_{45}[t+1] = c_{44}[t]$
$c_{44}[t+1] = c_{43}[t]$
$c_{43}[t+1] = c_{169}[t]$
$c_{42}[t+1] = c_{41}[t] \oplus c_{58}[t]c_{78}[t]c_{129}[t] \oplus c_{44}[t]c_{131}[t]$
$c_{41}[t+1] = c_{40}[t] \oplus c_{128}[t]c_{146}[t]c_{143}[t] \oplus c_{117}[t]c_{121}[t]c_{167}[t]c_{86}[t] \oplus c_{169}[t]c_{108}[t]c_{103}[t]c_{43}[t] \oplus c_{101}[t]c_{144}[t]c_{68}[t]c_{136}[t]$
$c_{40}[t+1] = c_{39}[t] \oplus c_{86}[t]c_{45}[t] \oplus c_{48}[t]c_{106}[t] \oplus c_{77}[t]c_{139}[t]c_{109}[t]c_{47}[t] \oplus c_{126}[t]c_{104}[t]c_{124}[t]$
$c_{39}[t+1] = c_{38}[t] \oplus c_{95}[t]c_{55}[t] \oplus c_{116}[t]c_{101}[t]c_{60}[t]c_{126}[t] \oplus c_{106}[t]c_{81}[t]c_{70}[t]$
$c_{38}[t+1] = c_{37}[t] \oplus c_{104}[t]c_{161}[t]c_{160}[t] \oplus c_{157}[t]c_{122}[t]c_{114}[t] \oplus c_{50}[t]c_{157}[t]c_{84}[t]c_{105}[t] \oplus c_{46}[t]c_{119}[t]$
$c_{37}[t+1] = c_{36}[t] \oplus c_{55}[t]c_{99}[t]c_{121}[t]c_{94}[t] \oplus c_{143}[t]c_{50}[t] \oplus c_{144}[t]c_{104}[t]c_{66}[t]c_{65}[t]$
$c_{36}[t+1] = c_{35}[t] \oplus c_{81}[t]c_{47}[t]c_{164}[t] \oplus c_{168}[t]c_{140}[t]c_{59}[t]$
$c_{35}[t+1] = c_{34}[t] \oplus c_{45}[t]c_{49}[t]c_{64}[t] \oplus c_{151}[t]c_{50}[t]c_{138}[t]$
$c_{34}[t+1] = c_{33}[t] \oplus c_{51}[t]c_{72}[t]c_{90}[t] \oplus c_{113}[t]c_{84}[t]c_{92}[t]c_{135}[t] \oplus c_{134}[t]c_{63}[t]c_{117}[t]c_{89}[t] \oplus c_{110}[t]c_{132}[t]$
$c_{33}[t+1] = c_{32}[t] \oplus c_{140}[t]c_{160}[t]c_{45}[t] \oplus c_{120}[t]c_{103}[t]c_{59}[t]c_{90}[t] \oplus c_{154}[t]c_{140}[t]c_{64}[t]$
$c_{32}[t+1] = c_{31}[t] \oplus c_{83}[t]c_{66}[t] \oplus c_{152}[t]c_{64}[t] \oplus c_{151}[t]c_{64}[t] \oplus c_{136}[t]c_{155}[t]$
$c_{31}[t+1] = c_{30}[t] \oplus c_{57}[t]c_{143}[t]c_{110}[t] \oplus c_{53}[t]c_{117}[t] \oplus c_{120}[t]c_{60}[t]c_{96}[t]c_{115}[t] \oplus c_{145}[t]c_{79}[t]c_{55}[t]$
$c_{30}[t+1] = c_{29}[t] \oplus c_{134}[t]c_{121}[t]c_{160}[t] \oplus c_{135}[t]c_{117}[t]c_{161}[t]$
$c_{29}[t+1] = c_{42}[t] \oplus c_{28}[t] \oplus c_{160}[t]c_{135}[t] \oplus c_{129}[t]c_{157}[t]$
$c_{28}[t+1] = c_{27}[t] \oplus c_{42}[t] \oplus c_{101}[t]c_{66}[t] \oplus c_{85}[t]c_{138}[t] \oplus c_{108}[t]c_{60}[t]c_{122}[t]$
$c_{27}[t+1] = c_{26}[t] \oplus c_{42}[t] \oplus c_{134}[t]c_{129}[t]c_{88}[t]c_{85}[t] \oplus c_{111}[t]c_{147}[t]$
$c_{26}[t+1] = c_{42}[t] \oplus c_{25}[t] \oplus c_{88}[t]c_{135}[t] \oplus c_{46}[t]c_{117}[t]c_{100}[t]c_{125}[t] \oplus c_{92}[t]c_{88}[t]c_{167}[t]c_{143}[t]$
$c_{25}[t+1] = c_{42}[t] \oplus c_{24}[t] \oplus c_{43}[t]c_{100}[t]c_{84}[t]c_{131}[t] \oplus c_{146}[t]c_{113}[t]c_{138}[t] \oplus c_{115}[t]c_{134}[t]c_{91}[t]c_{84}[t] \oplus c_{139}[t]c_{71}[t]$
$c_{24}[t+1] = c_{42}[t] \oplus c_{156}[t]c_{97}[t] \oplus c_{122}[t]c_{55}[t]c_{59}[t]c_{168}[t] \oplus c_{104}[t]c_{139}[t] \oplus c_{79}[t]c_{65}[t]c_{88}[t]$

$$
\begin{aligned}
c_{23}[t+1] &= c_{22}[t] \oplus c_{26}[t]c_{27}[t] \oplus c_{37}[t]c_{24}[t] \oplus c_{41}[t]c_{24}[t] \oplus c_{41}[t]c_{26}[t] \\
c_{22}[t+1] &= c_{21}[t] \oplus c_{30}[t]c_{35}[t]c_{27}[t] \oplus c_{32}[t]c_{24}[t] \oplus c_{25}[t]c_{38}[t]c_{32}[t] \oplus c_{27}[t]c_{32}[t]c_{29}[t] \\
c_{21}[t+1] &= c_{20}[t] \oplus c_{33}[t]c_{28}[t]c_{31}[t]c_{25}[t] \oplus c_{25}[t]c_{30}[t]c_{38}[t] \oplus c_{32}[t]c_{35}[t] \oplus c_{27}[t]c_{37}[t]c_{40}[t] \\
c_{20}[t+1] &= c_{19}[t] \oplus c_{35}[t]c_{36}[t]c_{34}[t] \oplus c_{32}[t]c_{24}[t]c_{31}[t]c_{28}[t] \oplus c_{42}[t]c_{36}[t]c_{37}[t]c_{41}[t] \oplus c_{36}[t]c_{26}[t]c_{28}[t] \\
c_{19}[t+1] &= c_{18}[t] \oplus c_{26}[t]c_{40}[t]c_{32}[t]c_{38}[t] \oplus c_{31}[t]c_{26}[t] \oplus c_{25}[t]c_{42}[t]c_{26}[t]c_{38}[t] \\
c_{18}[t+1] &= c_{17}[t] \oplus c_{29}[t]c_{40}[t]c_{31}[t]c_{35}[t] \oplus c_{27}[t]c_{37}[t]c_{39}[t]c_{40}[t] \oplus c_{24}[t]c_{31}[t]c_{38}[t]c_{26}[t] \\
c_{17}[t+1] &= c_{16}[t] \oplus c_{25}[t]c_{30}[t] \oplus c_{40}[t]c_{32}[t]c_{25}[t]c_{24}[t] \\
c_{16}[t+1] &= c_{15}[t] \oplus c_{40}[t]c_{34}[t] \oplus c_{33}[t]c_{41}[t]c_{32}[t] \oplus c_{35}[t]c_{31}[t]c_{33}[t]c_{42}[t] \oplus c_{26}[t]c_{28}[t]c_{36}[t]c_{33}[t] \\
c_{15}[t+1] &= c_{14}[t] \oplus c_{36}[t]c_{35}[t]c_{34}[t]c_{31}[t] \oplus c_{28}[t]c_{29}[t]c_{34}[t] \oplus c_{31}[t]c_{26}[t] \oplus c_{41}[t]c_{40}[t] \\
c_{14}[t+1] &= c_{13}[t] \oplus c_{38}[t]c_{36}[t]c_{34}[t] \oplus c_{40}[t]c_{28}[t]c_{30}[t] \oplus c_{24}[t]c_{27}[t] \oplus c_{32}[t]c_{39}[t]c_{33}[t]c_{24}[t] \\
c_{13}[t+1] &= c_{12}[t] \oplus c_{27}[t]c_{38}[t]c_{25}[t]c_{35}[t] \oplus c_{27}[t]c_{34}[t]c_{26}[t] \\
c_{12}[t+1] &= c_{11}[t] \oplus c_{36}[t]c_{31}[t]c_{42}[t] \oplus c_{42}[t]c_{39}[t]c_{41}[t] \oplus c_{26}[t]c_{39}[t] \oplus c_{39}[t]c_{30}[t] \\
c_{11}[t+1] &= c_{10}[t] \oplus c_{31}[t]c_{39}[t]c_{40}[t]c_{24}[t] \oplus c_{33}[t]c_{41}[t]c_{26}[t]c_{40}[t] \\
c_{10}[t+1] &= c_{9}[t] \oplus c_{23}[t] \oplus c_{29}[t]c_{25}[t] \oplus c_{28}[t]c_{38}[t]c_{24}[t] \oplus c_{33}[t]c_{29}[t] \oplus c_{35}[t]c_{27}[t]c_{26}[t] \\
c_{9}[t+1] &= c_{8}[t] \oplus c_{23}[t] \oplus c_{39}[t]c_{37}[t]c_{24}[t] \oplus c_{37}[t]c_{38}[t]c_{35}[t]c_{27}[t] \oplus c_{25}[t]c_{30}[t]c_{38}[t] \oplus c_{38}[t]c_{42}[t]c_{40}[t] \\
c_{8}[t+1] &= c_{23}[t] \oplus c_{7}[t] \oplus c_{27}[t]c_{39}[t]c_{40}[t] \oplus c_{38}[t]c_{34}[t]c_{26}[t]c_{39}[t] \oplus c_{33}[t]c_{40}[t]c_{32}[t] \oplus c_{36}[t]c_{24}[t]c_{31}[t] \\
c_{7}[t+1] &= c_{23}[t] \oplus c_{32}[t]c_{24}[t]c_{29}[t]c_{40}[t] \oplus c_{42}[t]c_{32}[t]c_{30}[t]c_{40}[t] \\
c_{6}[t+1] &= c_{5}[t] \oplus c_{6}[t] \oplus c_{21}[t]c_{9}[t] \oplus c_{22}[t]c_{15}[t]c_{14}[t]c_{19}[t] \\
c_{5}[t+1] &= c_{4}[t] \oplus c_{6}[t] \oplus c_{12}[t]c_{16}[t]c_{15}[t]c_{20}[t] \oplus c_{10}[t]c_{22}[t]c_{8}[t] \\
c_{4}[t+1] &= c_{3}[t] \oplus c_{6}[t] \oplus c_{19}[t]c_{7}[t]c_{9}[t] \oplus c_{15}[t]c_{19}[t]c_{13}[t]c_{9}[t] \oplus c_{19}[t]c_{17}[t]c_{20}[t]c_{23}[t] \oplus c_{10}[t]c_{21}[t] \\
c_{3}[t+1] &= c_{2}[t] \oplus c_{13}[t]c_{16}[t]c_{18}[t]c_{10}[t] \oplus c_{10}[t]c_{13}[t] \oplus c_{18}[t]c_{7}[t]c_{9}[t] \\
c_{2}[t+1] &= c_{6}[t] \oplus c_{22}[t]c_{19}[t]c_{7}[t] \oplus c_{20}[t]c_{9}[t]c_{17}[t]c_{22}[t] \oplus c_{7}[t]c_{10}[t]c_{13}[t] \oplus c_{14}[t]c_{22}[t]c_{21}[t]c_{11}[t] \\
c_{1}[t+1] &= c_{1}[t] \oplus c_{0}[t] \oplus c_{4}[t]c_{6}[t]c_{3}[t]c_{5}[t] \oplus c_{4}[t]c_{2}[t]c_{5}[t] \oplus c_{3}[t]c_{5}[t]c_{2}[t] \oplus c_{4}[t]c_{2}[t]c_{6}[t] \\
c_{0}[t+1] &= c_{1}[t] \oplus c_{5}[t]c_{2}[t]c_{6}[t]c_{4}[t] \oplus c_{3}[t]c_{5}[t]c_{6}[t]c_{2}[t]
\end{aligned}
$$