Heraclitus: A LFSR-based Stream Cipher with Key Dependent Structure

Bernard Colbert Anthony H. Dekker
Lynn Margaret Batten

Abstract

We describe Heraclitus as an example of a stream cipher that uses a 128 bit index string to specify the structure of each instance in real time: each instance of Heraclitus will be a stream cipher based on mutually clocked shift registers.

Ciphers with key-dependent structures have been investigated and are generally based on Feistel networks. Heraclitus, however, is based on mutually clocked shift registers. Ciphers of this type have been extensively analysed, and published attacks on them will be infeasible against any instance of Heraclitus.

The speed and security of Heraclitus makes it suitable as a session cipher, that is, an instance is generated at key exchange and used for one session.

1 Introduction

This paper describes *Heraclitus*, a proof of concept cipher demonstrating the existence of sound dynamic ciphers. It is a stream cipher that uses a 128 bit string to specify each instance in real time.It can be considered to be a stream cipher with an index dependent structure which can be set up in real time, and can therefore also be a *session cipher*, that is, a cipher used for a single session and then discarded.

Ciphers with key/index dependent structures, including stream ciphers, have been discussed theoretically [4, 19] and several have been implemented [14, 16, 21]. These ciphers, including the stream cipher *Mir-1*, use Feistel networks, s-boxes (non-linear transformations) and p-boxes (permutations). In general, the s- and p- boxes are the only key-dependent elements of the cipher. The rest of the structure, such as the number of s- and p-boxes, and the network are considered to be fixed. We construct in this paper a ciphers whose structure completely dependent on the index by using it to determine the number of registers, their lengths and their associated polynomials. As with session keys, any given 'instance' of the cipher is used for a period and then transformed into another instance of the cipher in an unpredictable manner.

Note that the cipher described in this paper can be implemented in a number of ways. The initial description provides the most general implementation that can be extended, both in terms of the number of registers used and the register lengths. However, we also present alternative approaches to implementing Heraclitus and variants.

In the next section, we give the background to A5.1. In Sections 3 to 5, we give a detailed construction of Heraclitus. In Section 7 we propose two initialisation modes for Heraclitus, one being suitable in resource constrained environments. Section 8 briefly discusses the security of Heraclitus, with a more complete discussion in Appendix A.

2 Background

2.1 Linear Feedback Shift Registers

A *shift register* of length n at discrete time t is a system based on a sequence of bits $\{r^t[i]\}_{i=1}^n$ that:

1. has state at time t, $r^t[1], \ldots, r^t[n]$;
2. has a one bit input at time t, s^t; and
3. at time $t+1$:

$$r^{t+1}[i] = \begin{cases} r^t[i-1] & 2 \leq i \leq n, \text{and} \\ s(t) & i = 1. \end{cases}$$

A *linear feedback shift register* (LFSR) is a shift register that takes its input as a linear combination of the bits of the register; that is:

$$s(t+1) = \sum_{i=1}^{n} c_i r^t[i],$$

where $\{c_i\}_{i=1}^n$ are fixed constants, either 0 or 1. The set $\phi = \{i | c_i = 1\}$ is the set of feedback bits of the LFSR.

Associated with each LFSR is a *feedback polynomial*, which is related to the feedback bits. The feedback polynomial is

$$\sum_{k \in \phi} x^k + 1.$$

Every LFSR has a *period*, which is the time it takes for the LFSR to return to its initial state. If the feedback polynomial is irreducible, the period will be $2^n - 1$, which is the maximum period.

In most cases, and the cases in which we will be interested, the output of the LFSR is: $f(r^t[1], \ldots, r^t[n]) = r^t[n]$. Note that the output may be a nonlinear function of the bits of the LFSR.

2.2 Description of A5.1

Each instance of Heraclitus is similiar in structure to the A5.1 cipher, which will be now described.

A5.1 is a linear feedback shift register (LFSR) stream cipher using three shift registers. The registers are clocked in an irregular fashion which determines the algebraic complexity of the output of the cipher.

This description of A5.1 is based on [3]. A5.1 is an linear feedback shift register (LFSR) based stream cipher. A5.1 has three LFSRs, R_1, R_2, and R_3, of lengths 19, 22 and 23 respectively, which is denoted by $\|R_i\|$. The bits of R_i are denoted $R_i[j]$ for $1 \leq j \leq \|R_i\|$. Table 1 details the feedback bits and the clock bits. The output z at time t is:

$$z^t = r_1^t[19] + r_2^t[22] + r_3^t[23].$$

A5.1 is irregularly clocked, that is, each register does not necessarily move for each clock cycle. Rather, the register has a clock bit, which is close to the middle of the register — for each register the clock bit is $r_i[\|R_i\|]$. The register is only moved if the the clock bit is in the majority: that is, the clock bit has the same value as one of the other two clock bits. That is if

$$S_i^t = c_i^t + r_1^t[9] \cdot r_2^t[11] + r_1^t[9] \cdot r_3^t[11] + r_2^t[11] \cdot r_3^t[11] + 1 = 1,$$

then R_i is clocked, where c_i^t is the clock bit of R_i at time t. Observe that on average each register will move 5 times for every 8 clock cycles.

Al-Hinai *et al.* [1] derived the following algebraic expression for the values of the registers at time $t+1$:

$$r_i^{t+1}[j] \quad = \quad \begin{cases} S_i^t \cdot r^t[j-1] + (S_i^t + 1) \cdot r^t[j] & j > 1, \text{and} \\ S_i^t \cdot F_i^t + (S_i^t + 1) \cdot r^t[j] & j = 1, \end{cases} \tag{1}$$

where F_i^t is the value of the feedback polynomial of R_i at time t.

Note that the irregular clocking introduces nonlinearity into the cipher. Also note that in an algebraic representation of the cipher, the clocking also introduces a large number of monomial terms.

Register	**Feedback Bits**	**Feedback Polynomial**	**Clock Bit**
R_1	$r_1[19], r_1[18], r_1[17], r_1[14]$	$x^{19} + x^{18} + x^{17} + x^{14} + 1$	$r_1[9]$
R_2	$r_2[22], r_2[21]]$	$x^{22} + x^{21} + 1$	$r_1[11]$
R_3	$r_3[23], r_3[22], r_3[21], r_3[8],$	$x^{23} + x^{22} + x^{21} + x^8 + 1$	$r_1[11]$

Table 1: Summary of A5.1

2.3 Attacks on A5.1

Many attacks on A5.1 have been published, some recently [8]. The attacks generally follow one of the following strategies.

- Guessing attacks, guess the two shorter registers, and determine the longer register.
- Golić [9] uses a time-memory trade off to recover the initial state of the cipher. If M and T are the memory and time requirements, then $M \cdot T \geq 2^{63.32}$; thus this attack can be quickened by precomputation.
- Biryukov *et al.* [3] provide a refinement of Golić's attack. They focus on certain special states which produce particular patterns. This increases the speed of the attack. This attack requires about 2^{42} to 2^{48} steps of preprocessing and data storage and requires a deep knowledge of all aspects of the cipher.
- Al-Hinai *et al.* [1] applied algebraic attacks to A5.1.

Barkan *et al.* [2] improved the third attack, and Nohl and Krißler [17] have set up Rainbow tables to break A5.1 based on a similar approach.

In analysing the cipher, the last attack demonstrates that the clocking mechanisms provide significant immunity to algebraic attacks. The first three attacks have only become feasible in recent years with an increase in processing power and storage space but still face the difficulty of handling large amounts of data. Consequently, these attacks become infeasible if the overall size of the cipher is increased while retaining the essence of the clocking mechanism. This can be done in several ways, such as increasing the lengths of the three ciphers and adding registers to the cipher.

These considerations, together with the observation that A5.1 operates in a resource constrained and challenging environment, have motivated the authors of this paper to build a strengthened version of A5.1. In addition to the obvious extension to more and longer registers, we propose a novel approach which changes both the number and size of registers used in any one session as well as the polynomials underlying the feedback registers.

2.4 Elements of the Cipher

From the above description, A5.1 is an example of an LFSR based cipher with majority clocking. The parameters of this type of cipher are:

- the key length K',
- the number of registers r,
- the length of registers $\{\rho_i\}$ where the registers are denoted by $\{R_i\}_{i=1}^{r}$,
- the feedback bits of the registers, and
- the clocking mechanism.

With the exception of the clocking mechanism, we vary these parameters in Heraclitus.

3 Overview of Construction

As noted above, the parameters of the cipher that are varible are the number of registers r, the length of registers $\{\rho_i\}$, and the feedback polynomials of the registers. These parameters are specified used a 128 bit string, $\mathcal{X}$ called the *Index*. This is distinct from the key used by the instance of the cipher.

For Heraclitus the key length is fixed to be 128 bits: thus the total register length will need to exceed 128 plus the check bits required to any register having a trivial loading.

Our construction starts with an index which is a 128 bit string $\mathcal{X} = \xi_1 \cdots \xi_{128}$, and consists of three steps:

1. selection of number of registers;
2. selection of register sizes; and
3. selection of feedback bits for each register by generating irreducible polynomials.

The index $\mathcal{X}$ is used for each of these steps.

The majority clocking function used in the A5.1 cipher will be used due to its simplicity and to allow reuse of analyses of A5.1. Moreover, it was extensively analysed algebraically in [1].

Pseudocode for these algorithms can be found in Appendix C. A discussion of alternative design choices for Heraclitus is found in Appendix B

4 Number and Size of Registers

4.1 Considerations and Choices

The number and size of registers will be determined by the initial bits of the index $\mathcal{X}$. To use the majority clocking function unmodified, an odd number of registers will be used. Another concern is that if the registers are long and relatively few, a significant proportion of the bits, a third or a fifth, may be derived from other bits and the key stream, and therefore be more amenable to analysis.

Consequently, the number of registers will be chosen from $\{7, 9, 11\}$.

The following points were considered in determining the range of the lengths of the registers.

- The sum of lengths of the registers used in any session should equal or exceed the key size, including the check bits. That is:

$$\sum_{i=1}^{r} \rho_i \geq K.$$

- The lengths of the registers should not present difficulties in implementation either in software or in hardware.

- The register sizes need to be limited to ensure that the algorithms used to determine the feedback bits are tractable.
- The register sizes are not too small to limit the scope for analysis based on a brute force indexing of the smaller registers.
- The register lengths should be pairwise coprime.

Most modern operating systems are based on 64 bit registers. Moreover, the decryption algorithm used to determine the feedback bits has a complexity of $O(m^3)$, where m is the length of the register: this requires $O(2^{18})$ operations for registers of less than 64 bits which is feasible for most devices, including resource constrained devices. Consequently, 64 bits is chosen as the upper bound for register sizes.

The minimum size for registers is set to be 17 in order to limit the scope of brute force attacks on these systems.

The requirement that the register lengths are pairwise coprime reduces correlation attacks against the cipher and ensures that the cycle time is maximised [15]. This is certainly seen in most stream ciphers published. Consequently, in our implementation, we chose register lengths to be primes and prime powers. This has the added benefit of simplifying the selection of feedback bits.

We chose the set, ζ of possible register lengths to be:

$$\zeta = \{17, 19, 23, 25, 27, 29, 31, 32, 37, 41, 43, 47, 49, 53, 59, 61\}$$

This set has 16 elements: however, when using only 7 registers, only the 15 longest are used.

Based on a choice of 7, 9 or 11 registers, the first two bits of the index $\mathcal{X}$ are used to determine the number of registers. This will also determine the number of possible register lengths. The first 14 bits of $\mathcal{X}$ are used to make these determinations.

1. If $\xi_1\xi_2 = 00$, then $r = 7$, which allows $\binom{15}{7}$ or 6435 choices of register lengths. $\xi_3 \ldots \xi_{14}$ are used for 4096 possible register lengths.
2. If $\xi_1\xi_2 = 01$, then $r = 11$, which allows $\binom{16}{11}$ or 4368 choices of register lengths. $\xi_3 \ldots \xi_{14}$ are used for 4096 possible register lengths.
3. If $\xi_1\xi_2 = 1$, then $r = 9$, which allows $\binom{16}{9}$ or 11440 choices of register lengths. If 13 bits are used to make this choice up to 8192 register lengths can be chosen

In our implementation, a look-up table was used to choose the register lengths. This is an efficient method as the table is used repeatedly and the space needed is minimal.

In choosing the number of registers and the register lengths, care must be taken to ensure that the above inequality with K is satisfied.

5 Feedback Bits

The next step is to determine the feedback bits. As discussed in Golomb [10] and in Section 2.1 the feedback bits are directly related to the feedback polynomials. Moreover, the feedback polynomials should be irreducible to ensure that each register has maximal cycle length.

Thus, to select the feedback bits for the registers, irreducible polynomials are generated and the corresponding feedback bits for the register are used.

Elements of GF(2^n) can be used to generate irreducible polynomials of order n over GF(2). The algorithm chosen to generate the polynomials is outlined in §4.74 of [15].

Note that GF(2^n) is isomorphic to GF(2)$[x]/p(x)$, where $p(x)$ is an irreducible polynomial of order n over GF(2). Consequently, Heraclitus uses a fixed set of irreducible polynomials to generate feedback polynomials, one for each register, to use this algorithm. Table 2 lists the irreducible polynomials used in Heraclitus, $\mathcal{P}_i$: these were taken from Seroussi [20].

ρ	$\mathcal{P}_\rho$
17	$x^{17} + x^3 + 1$
19	$x^{19} + x^5 + x^2 + x + 1$
23	$x^{23} + x^5 + 1$
25	$x^{25} + x^3 + 1$
27	$x^{27} + x^5 + x^2 + x + 1$
29	$x^{29} + x^2 + 1$
31	$x^{31} + x^3 + 1$
32	$x^{32} + x^7 + x^3 + x^2 + 1$
37	$x^{37} + x^6 + x^4 + x + 1$
41	$x^{41} + x^3 + 1$
43	$x^{43} + x^6 + x^4 + x^3 + 1$
47	$x^{47} + x^5 + 1$
49	$x^{49} + x^9 + 1$
53	$x^{53} + x^6 + x^2 + x + 1$
59	$x^{59} + x^7 + x^4 + x^2 + 1$
61	$x^{61} + x^5 + x^2 + x + 1$

Table 2: Irreducible Polynomials

Elements can be represented as n bit strings, α where

$$\alpha \longleftrightarrow \sum_{i=1}^{n} \alpha_i x^{i-1}.$$

Addition and multiplication are as for polynomial rings, calculated modulo $p(x)$. 'The trivial elements are the additive identity 0 and the multiplicative identity 1.

For p prime, all non-trivial elements α of GF(2^p) will generate an irreducible polynomial by:

$$m_\alpha = \prod_{i=0}^{p-1}(x - \alpha^{2^i}).$$

Two elements, α and β will generate the same irreducible polynomial if, and only if, there exists an integer $1 \leq j \leq p$ such that: $\beta = \alpha^{2^j}$. The collection $\{\alpha^{2^i}\}_{i=0}^{p-1}$ is known as the *cyclotomic coset* of α [13].

For q, a prime power, similar properties hold: however, not every non-trivial element of GF(2^q) will generate irreducible polynomials — only those elements with exactly q elements in its cyclotomic coset. That is, those elements α for which the smallest integer t such that $\alpha^{2^t} = \alpha$ is q.

In order to use this method to choose irreducible polynomials from the values of $\mathcal{X}$, there needs to be some way of generating strings of the length of the registers. Ideally, some method of indexing cyclotomic cosets would be used in order that each value of $\mathcal{X}$ would generate a different set of irreducible polynomials. However, there is no generalised simply implementable indexing of cyclotomic cosets or representatives from these cosets.

The method that we have chosen is to use a one way hashing function to generate bit strings that have the following properties:

- uniform distribution over the strings;
- the correlation between chosen polynomials between registers is 0; and
- collisions are difficult to find.

The one way hashing function chosen is SHA512, which has these properties. Moreover, it is a standard and many implementations exist, including for constrained environments. This will provide a 512 bit string in which substrings of the appropriate lengths can be chosen.

The method for choosing the feedback bits for the register of length n is as follows.

1. Take the first n unused bits of $SHA512(\mathcal{X})$.
2. If it corresponds to a trivial element, set $\mathcal{Q}_n = \mathcal{P}_n$.
3. Else, if $n \in \{25, 27, 32, 49\}$ test the string. If it does not generate an irreducible polynomial, discard the first bit and concatenate the first unused bit of $SHA512(\mathcal{X})$. If the test fails three times, set $\mathcal{Q}_n = \mathcal{P}_n$ as the default feedback polynomial.
4. Else generate irreducible polynomial $\mathcal{Q}_n$.
5. Set the feedback bits to correspond with $\mathcal{Q}_n$.

Pseudocode for Algorithms 3 and 4 can be found in in Appendix C. Algorithm 3 tests whether a string will generate an irreducible polynomial for

$n \in \{25, 27, 32, 49\}$. Algorithm 4 generates irreducible polynomials from the strings, using Algorithm 3 to test as appropriate.

Each polynomial generation will have a running time of $O(m^3)$ operations. Note that an upper limit of three attempts to generate an irreducible polynomial is set. This is to ensure that the polynomial will be generated within a known amount of time.

Two concerns exist for the generation of irreducible polynomials (a) the proportion of irreducible polynomials generated, and (b) significant bias of polynomials generated.

For all registers, the probability that a particular polynomial will not be generated using this method is less than 2^{-85}: for polynomials of order 17, this probability will be significantly less. This indicates that a large proportion of the set of irreducible polynomials will be generated.

The algorithm will introduce a small bias for the default polynomial to be generated. For a prime, p, the default will occur at a rate of $(p + 2)2^{-p}$, the other polynomials occur at a rate of $p \cdot 2^{-p}$. For a prime power, $q = p^n$, the default polynomial will occur at a rate of about $(p+2)2^{-q}+2^{-3(q-q/p-1)}$. These increases are small and will not introduce a significant bias, and therefore will not provide significant assistance to an attacker.

6 Irregular Clocking

The clocking mechanism is the one that is used for A5.1: that is, majority clocking based on fixed bit positions in the LFSRs. In A5.1, the clock bits are at the centre of the registers. There appear to be a number of good reasons: the primary one seems to be that this position is furthest from the output and therefore most difficult to correlate.

Thus, for each register, R_i, the clocking bit is chosen to be $c_i = R_i[j]$ where $j = \lfloor \|R_i\|/2 \rfloor$, that is, about the middle bit.

7 Initialisation

For stream ciphers, synchronicity is essential for correct function. In harsh environments, such as radio interfaces, traffic will be lost, which poses a problem for stream ciphers and other mechanisms requiring synchronicity. This is usually addressed by putting the traffic into frames and assigning frame numbers or other identifiers for the frame. Many ciphers, e.g. A5.1 and RC4, incorporate the frame number into the session key. Thus, for each frame, the frame number may be XORed or appended to the session key and loaded into the cipher and used for the traffic of that frame.

However, this incorporation may lead to weaknesses using the cipher, in particular related IV attacks [5, 7, 18]. This is usually addressed by running the cipher for a fixed number of clock cycles to ensure that sufficient diffusion of the key bits and IV occurs.

An alternative to this is to put the IV and session key through a hash function such as SHA1 and to load the registers with the output of this hash function.

In resource constrained environments using hash functions may not be possible, and other mechanisms to avoid related IV attacks are required. Consequently, two initialisation modes are proposed for Heraclitus: Hash mode and Run mode; noting that Hash mode is preferred when it is possible. Both assume that the session key is $\mathcal{K} = \kappa_1 \cdots \kappa_{128}$ and IV is the initialisation vector or frame number or sequence number.

For Hash mode, the string $k_f = \text{SHA512}(\mathcal{K}, IV)$ is generated. The initial R bits of the string k_f are loaded into the cipher, where R is the total length of the registers. The registers are loaded from the shortest to the longest register. The remaining bits of k_f are discarded. Then the complement of the parity of each subkey of each register is loaded to avoid the null loading.

For Run mode the key for the frame is generated $k_f = \mathcal{K} \oplus IV$, IV being padded by repetition until it is 128 bits long. This is divided into subkeys, depending on the number of registers:

1. for 7 registers: $k_1 \cdots k_{18}$, $k_{19} \cdots k_{36}$, $k_{37} \cdots k_{54}$, $k_{55} \cdots k_{72}$, $k_{73} \cdots k_{90}$, $k_{91} \cdots k_{109}$, and $k_{110} \cdots k_{128}$;
2. for 9 registers: $k_1 \cdots k_{14}$, $k_{15} \cdots k_{28}$, $k_{29} \cdots k_{42}$, $k_{43} \cdots k_{56}$, $k_{57} \cdots k_{70}$, $k_{71} \cdots k_{84}$, $k_{85} \cdots k_{98}$, $k_{99} \cdots k_{113}$, and $k_{114} \cdots k_{128}$;
3. for 11 registers: $k_1 \cdots k_{11}$, $k_{12} \cdots k_{22}$, $k_{23} \cdots k_{33}$, $k_{34} \cdots k_{44}$, $k_{45} \cdots k_{56}$, $k_{57} \cdots k_{68}$, $k_{69} \cdots k_{80}$, $k_{81} \cdots k_{92}$, $k_{93} \cdots k_{104}$, $k_{105} \cdots k_{116}$, and $k_{117} \cdots k_{128}$.

Each subkey has a parity bit appended to prevent a null loading. The subkeys are loaded into each of the registers, starting with the smallest going to the longest.

The cipher is then run to ensure that every bit in the registers is dependent on every key bit and frame bit, which will provide resistance to related IV attacks. Majority clocking will ensure that each register moves about half the time, then running the cipher for three times the length of the longest register will ensure that this condition is met. For simplicity, 183 is used in all cases to ensure this.

The advantage of Hash mode is that the key loaded into each frame is generated by a hash function, and therefore even if an attacker were to determine the key for one frame, the keys for the other frames are protected by the hash function. However, hash functions require significant processing that may not be available in low resource environments. Although Run mode does not have this protection, experience with A5.1 and other ciphers indicate that this an effective mechanism to protected the cipher.

Figure 1 shows the feed-through of the hashed key and IV information through the 2^{64} frames of a session.

The fact that the key and IV are used to determine the structure of the cipher in the next 2^{64} frames adds to the unpredictability of future set-ups and consequently provides security against attacks.

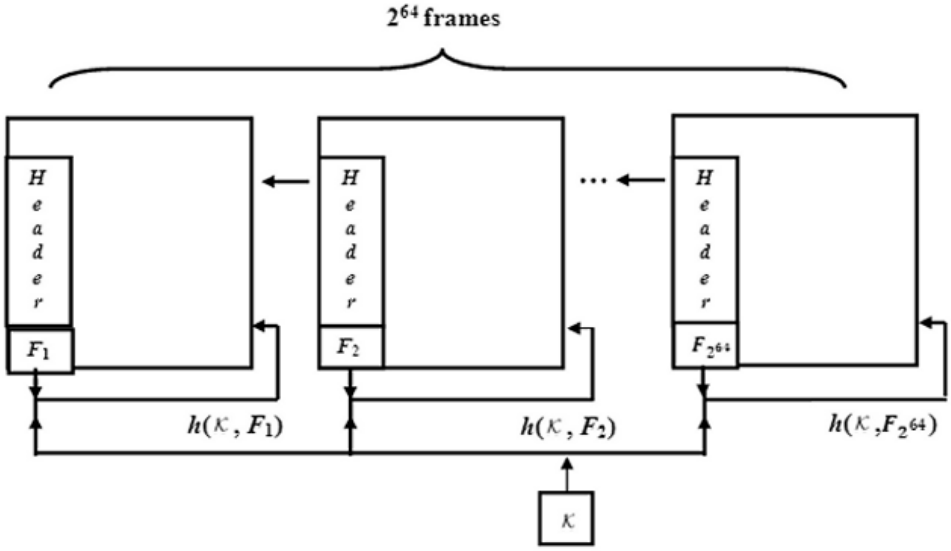

Figure 1: Each session key $\mathcal{K}$ is used for 2^{64} frames.

8 Security Analysis

The security of Heraclitus was analysed briefly. The following are the initial conclusions of the analysis.

- Due to the increase in key length, guessing attacks are not feasible.
- The time-memory trade of attacks of Golić are not feasible due the increase in key length and the number of registers [9].
- The refinements of Golić's attack are not feasible [2, 3]. This is due to the increase key size, thus requiring feasible precomputation: moreover, there are a large number of register number and length combinations, making the search for patterns and refinements more difficult. Each combination will require its own precomputed data.
- Each cipher within Heraclitus is immune to algebraic attacks due to the non-linear clocking. Thus, Heraclitus is immune to algebraic attacks [1].
- Each cipher within Heraclitus is independent of and uncorrelated with the other ciphers.

Each instantiation of Heraclitus is therefore immune to the known attacks against A5.1. However, possible weaknesses in transitions between instantiations are not yet fully known. For this reason, the authors have lodged

the code at the ECRYPT Benchmarking of Cryptographic Systems (eBACS http://bench.cr.yp.to/) for general testing by the cryptographic community.

9 Implementation

One of the practical concerns for any cipher is the implementation of the cipher — either in hardware or software. Variants of Heraclitus have been implemented in software, two of which are described briefly: the first by the second author, and the second by a group of undergraduate students in 2008 as a group project [11].

These implementations demonstrate:

- the method is extensible and adaptable,
- that it is implementable, and
- that it is usable.

9.1 Heraclitus-64

The description of Heraclitus given here is very general, and a number of variants can be made. A variation of Heraclitus was implemented in June 2008. In this version:

- the index was restricted to 64 bits;
- the number of registers is either 5 and 7;
- the set $\{25, 27, 29, 31, 32, 37, 41, 43, 49, 53, 59, 61\}$ was used for the register lengths;
- the number of registers and register sizes are chosen by an 11 bit string which is stored in a table; and
- 65536 irreducible polynomials for each register length were generated and stored as an array of a 64 bit word length.

Thus, to generate the cipher, up to 8 table look ups were required and the register implemented. The storage requirements are very modest. The array of irreducible polynomials for each register length requires 2^{19} bytes, or 0.1MB per array. Also the number and length of registers are determined by 11 bits, which can also be stored as an array.

Thus, the total storage required is less than 2MB.

9.2 A5+

This was a variation of Heraclitus that was implemented by a group of undergraduate students [11]. Its structure is:

- the index was restricted to 112 bits;
- 7 registers are used for all instances;
- the register lengths are chosen from a set of 14 possible lengths; and
- the feedback bits are chosen from a possible 65535 sets per register.

The possible feedback polynomials for each register are stored in tables, and the index is used to select them for the registers being used. Each of these feedback taps correspond to an irreducible polynomial.

The set up required a small amount of processing and 7 table look ups: the total time for setting up an instance took less than a microsecond on a standard business PC.

10 Efficiency

The efficiency of a cipher is dependent on the implementation and the resources required to implement the cipher, and the context in which it is being implemented.

The complexity of the algorithms specified is at most cubic in the *length of the register* — i.e. $O(n^3)$: and $n \leq 64$. This presents a small overhead in setting up the cipher, and indicates that the register lengths can be increased significantly in this type of cipher without a large performance impact.

However, with the availability of cheap memory on most standard PCs and servers, pre-computing these algorithms and implementing the algorithms as look-up tables provides significant time and implementation efficiecies. This is demonstrated by the choices made in implementing Heraclitus-64 and A5+. Estimates for the time and memory requirements of the variants are found in Table 3

Variant	**Set Up**	**Memory Runtime**	**Memory Storage**
Heraclitus	$c \cdot 3 \times 10^6$ instructions	$<$ 10kB	$<$ 10kB
Heraclitus-64	8 table lookups	$<$ 200kB	2 MB
A5+	7 table lookups	1 MB	8 MB

Table 3: Summary of A5.1

Heraclitus has not been implemented in hardware. However, there are some observations for a hardware implementation:

- each register of Heraclitus will need three physical registers to implement it — one to act as a register, one for the feedback bits and one as a length mask;
- the algorithms to determine the feedback bits can be either implemented as look ups, as for software, or implmented as the algorithms described.

The tables will take up significant memory. However, if a cipher with the structure of Heraclitus is to be implemented in hardware, then it can be modified to ensure that the memory requirements are reduced to acceptable levels. For example, Heraclitus-64 reduced the total memory requirement to less than 2MB. Similar modifications could be used in hardware.

11 Conclusions

Heraclitus demonstrates that ciphers with key dependent structures can be extended to LFSR based stream ciphers, and that each instance can be implemented in real time. Moreover, each instance of the cipher has

In describing Heraclitus, an extension of the A5.1 cipher, we wrote algorithms to specify the parameters of the ciphers such as register lengths and feedback taps. These algorithms all run in at most cubic time ($O(n^3)$), and $n \leq 64$. An adaptation of Heraclitus was implemented which uses up to 91 bits for an index and at most eight table lookups and 2MB of stored data. This implementation is suitable for resource constrained environments.

The design of Heraclitus exploited the choices available in cipher design, such as the choice of irreducible polynomials or the choice a function which satisfies certain conditions. They also represent an increase in strength of the ciphers because: (a) each cipher generated is designed to satisfy particular criteria to ensure the strength of the cipher; and (b) each cipher is expected to be only used once — therefore it is infeasible to determine any weakness, even if the cipher is known.

Heraclitus is only one example of this type of approach: it can be extended and adapted to other stream ciphers. Further investigation into key dependent block ciphers and hashing functions is currently underway.

References

[1] Sultan Al-Hinai, Lynn Margaret Batten, and Bernard Colbert. Mutually Clock-Controlled Feedback Shift Registers Provide Resistance to Algebraic Attacks. In Dingyi Pei, Moti Yung, Dongdai Lin, and Chuankun Wu, editors, *Inscrypt*, volume 4990 of *Lecture Notes in Computer Science*, pages 201–215. Springer, 2007.

[2] Elad Barkan, Eli Biham, and Nathan Keller. Instant Ciphertext-Only Cryptanalysis of GSM Encryption Communication. *Journal of Cryptology*, 21(3):240–245, July 2008.

[3] Alex Biryukov, Adi Shamir, and David Wagner. Real Time Cryptanalysis of A5/1 on a PC. In Bruce Schneier, editor, *FSE*, volume 1978 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2000.

[4] H. Englund, T. Johansson, and M.S. Turan. A framework for chosen iv statistical analysis of stream ciphers. In *Indocrypt 2007, LNCS 4859*, pages 268–281, 2007.

[5] Håkan Englund, Thomas Johansson, and Meltem Sönmez Turan. A Framework for Chosen IV Statistical Analysis of Stream Ciphers. In K. Srinathan, C. Pandu Rangan, and Moti Yung, editors, *INDOCRYPT*, volume 4859 of *Lecture Notes in Computer Science*, pages 268–281. Springer, 2007.

[6] P. Erdős and B. Richmond. On partitions of N into summands coprime to N. *Aequationes Mathematicae*, 18(1-2):178–186, 1978.

[7] Simon Fischer, Shahram Khazaei, and Willi Meier. Chosen IV Statistical Analysis for Key Recovery Attacks on Stream Ciphers. In Serge Vaudenay, editor, *AFRICACRYPT*, volume 5023 of *Lecture Notes in Computer Science*, pages 236–245. Springer, 2008.

[8] Timo Gendrullis, Martin Novotny, and Andy Rupp. A Real-World Attack Breaking A5/1 within Hours. Cryptology ePrint Archive, Report 2008/147, 2008. urlhttp://eprint.iacr.org/.

[9] Jovan Dj. Golić. Cryptanalysis of Alleged A5 Stream Cipher. In *EUROCRYPT*, pages 239–255, 1997.

[10] Solomon W. Golomb. *Shift Register Sequences*. Aegean Park Press, 1982.

[11] Justin Hancy, Mark Guzzo, Emma Sheldon, Andrew Cox, and Brett Gervasoni. An Examination of the A5+ Encryption Alogrithm. Technical report, Deakin University, 2008.

[12] Rudolf Lidl and Harald Niederreiter. *Introduction to Finite Fields and their Applications*. Cambridge University Press, 1994.

[13] F. J. MacWilliams and N. A. J. Sloane. *The Theory of Error-Correcting Codes*. Elsevier, 1977.

[14] Alexander Maximov. A New Stream Cipher "Mir-1", 2005. eSTREAM Submission.

[15] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography, Fifth Printing*. CRC Press, 2001.

[16] Ralph C. Merkle. Fast Software Encryption Functions. In Alfred Menezes and Scott A. Vanstone, editors, *CRYPTO*, volume 537 of *Lecture Notes in Computer Science*, pages 476–501. Springer, 1990.

[17] Karsten Nohl and Sascha Krißler. Subverting the security base of GSM. Hacking at Random, August 2009. https://har2009.org/program/attachments/119_GSM.A51.Cracking.Nohl.pdf.

[18] Markku-Juhani Olavi Saarinen. Chosen-IV Statistical Attacks on eStream Ciphers. In Manu Malek, Eduardo Fernández-Medina, and Javier Hernando, editors, *SECRYPT*, pages 260–266. INSTICC Press, 2006.

[19] Bruce Schneier. *Applied Cryptography.* Wiley, second edition, 1996.

[20] Gadiel Seroussi. Table of Low-Weight Binary Irreducible Polynomials. Technical Report HPL-98-135, Hewlett-Packard Laboratories, August 1998.

[21] Runtong Zhang and Like Chen. A Block Cipher Using Key-Dependent S-box and P-boxes. In *ISIE 2008. IEEE International Symposium on Industrial Electronics, 2008.* IEEE, July 2008.

A Security Analysis

Since Heraclitus is a proof of concept of one time ciphers, the aim of this section is not to provide a complete analysis, but to demonstrate the immunity of Heraclitus to the attacks used against A5.1.

It aims to provide indications that:

- each example of Heraclitus is secure, and immune to attacks currently published against this type of cipher; and
- the indexing will provide further security.

One of the first considerations of Heraclitus is the number of ciphers that are generated, and then if the ciphers generated are independent of each other and are not easily mapped to each other.

Once it has been established that a large number of ciphers are generated, then attacks against (a) the ciphers themselves and (b) the collection of the ciphers will need to be considered. As indicated in §2.3, three approaches have been used against this class of ciphers. These are:

- guessing attacks.
- time-space trade off as described in Golić [9] and the refinements proposed and implemented in Biryukov *et al.* [3] and Barkan *et al.* [2]; and
- algebraic attacks as described in Al-Hinai *et al.* [1].

Guessing attacks are not feasible due to the increase in key length and will not be further considered in this analysis.

In considering these attacks, we will first consider the attack against an arbitrary example of Heraclitus, which is denoted H.1. We then consider the further difficulty of analysing the entire family.

A.1 Number of Ciphers Generated

One of the concerns in designing Heraclitus was the number of distinct ciphers that were generated. This can be seen as the effective number of bits of the index $\mathcal{X}$. One measure of this efficiency is $(\log_2 \|\{H_\xi\}_{\xi\in\mathcal{X}}\|)/128$.

The number, $I(n)$, of degree n irreducible polynomials over $GF(2)$ is given by [12]:

$$I(n) = \frac{1}{n}\sum_{d|n} \mu(d) 2^{n/d},$$

where the Möbius function μ is defined:

$$\mu(d) = \begin{cases} 0 & \text{if } d \text{ has repeated prime factors;} \\ 1 & \text{if } d = 1\text{; and} \\ (-1)^k & \text{if } d \text{ is a product of } k \text{ distinct primes.} \end{cases}$$

The $d = 1$ term is the dominating correction term of the sum. For p prime, the sum is $(2^p - 2)/p$; for prime powers, $q = p^k$, p prime the sum is $(2^q - 2^{q/p} - 2)/n$. Table 4 lists the number of irreducible polynomials for each register length and the appropriate number of bits to index them. This is the number of suitable sets of taps for the LFSRs of that length.

Note that using the one-way function, SHA512, we have:

- uniform distribution over possible bits strings used to generate the irreducible polynomials;
- the bits strings for any two registers will be uncorrelated; and
- collisions will be minimal.

There is a small probability that a particular irreducible polynomial will not be generated as $\mathcal{X}$ goes through its range of values. For any particular set of registers, the first 14 bits can be considered to be fixed: consequently, there are 2^{114} possible values for $\mathcal{X}$. Thus, the probability that particular polynomial will not generated is:

$$\Pr(n) \leq \left(1 - \frac{1}{I(n)}\right)^{2^{114}}.$$

Consider $n = 61$, in this case, $\Pr(n) \sim 2^{-85}$: which indicates that almost all of the 37800705069077000 irreducible polynomials will be generated. For $n < 61$, $\Pr(n) < 2^{-85}$. Thus, we can assume that at least $I(n)/2$ will be generated for each of the registers.

Due to the properties of SHA512, the bit strings that will be generating the polynomials for each register will also be uncorrelated, and therefore will have a small probability of collision.

Thus, for the case of the smallest set of registers, the number of irreducible combinations expected to be generated is 2^{114}.

Length n	**Number of Irreducible Polynomials $I(n)$**	**Number of Bits**
17	7710	12.91
19	27594	14.75
23	364722	18.48
25	1342176	20.36
27	4971008	22.25
29	18512790	24.14
31	69273666	26.05
32	134215680	27.00
37	3714566310	31.79
41	53634713550	35.64
43	204560302842	37.57
47	2994414645858	41.45
49	11488774559616	43.39
53	169947155749830	47.27
59	9770521225481750	53.12
61	37800705069077000	55.07

Table 4: Number of Irreducible Polynomials for Each Register Length

This indicates that the index efficiency is close to 1. We note that in the variations of Heraclitus, described in §9, the index efficiency is 1, since every index is guaranteed to generate a different cipher.

While SHA512 is not compromised, guessing the index $\mathcal{X}$ is the most efficient approach to analysing the family of ciphers.

A.2 Independence of Ciphers

As indicated above, there are many ciphers generated. This alone does not, of itself, guarantee an increase in security. If the generated ciphers are linearly related, then the increase in security is minimal.

Consider two ciphers H_α and H_β that have:

- the same number of registers and register length,
- the same feedback bits in all but one register, and
- the same key.

That is, they are identical with the exception of one register, $r_{i,\alpha}$ for H_α and $r_{i,\beta}$ for H_β both being of length n. What is the correlation between the two key streams?

For any two regularly clock LFSRs of the same length, differing only in feedback bits, there exists a function $L : t \times (GF(2))^n \to (GF(2))^n$ such that

at time t the state r^t of the two registers are related:

$$r^t_{i,\alpha} = L(t, r_{i,\beta}).$$

For t fixed, the function L is linear. Similarly, the state of the LFSR at time t_2 and time t_2 will also be a similar function.

The irregular clocking increases the complexity of constructing H_β from H_α by:

1. introducing non-linearity into the register positions; and
2. introducing dependency of the values in the other registers for the position of any particular register.

Consequently, if the keystream of the ciphers H_α and H_β can be mapped in feasible time, then the key stream of H_α can be constructed from a small amount of known key stream. Therefore, relating two ciphers is at least as difficult as breaking the cipher H_α. There is currently no indication that A5.1 type ciphers can be easily broken for the parameters that Heraclitus uses.

Thus, the indications are that the ciphers generated within Heraclitus are independent of each other.

A.3 Time-Memory Trade Off

In considering the time-memory trade off attacks, we first consider the analysis of a single instance of the cipher: that is, a cipher in which the registers and irreducible polynomials have been determined. In the discussion, this will referred as H.1.

The attacks published by Biryukov *et al.* [3] and Barakan *et al.* [2] are based on refinements of Golić's original attack. Two important aspects of the refinement are:

- particular observations about recurring bit patterns in the key stream which correspond to particular states of the registers; and
- being able to precompute large amounts of data to be searched during the attack.

These are used to speed up the attack significantly. Note, however, that these improvements depend on:

- the way that A5.1 is used in GSM and that frame numbers are added into the key for the frame; and
- knowledge of the structure of the cipher and the feedback taps to derive some of the patterns.

This allows them to effectively remove one register from consideration, reducing the attack to about 2^{45} steps.

As estimated in Golić [9], if M and T are the memory and time requirements for A5.1, then $M \cdot T \geq 2^{63.32}$. The cipher H.1 has a very similar structure to A5.1: the total length of the registers is, however, at least 144 bits. Applying the time-memory trade off to H.1, the estimate for the required processing is $M \cdot T \geq 2^{140}$.

Similar improvements should be expected for H.1: thus, a similar attack will need about 2^{112} steps, and significant amounts of precomputation.

Thus, the time-memory attacks of Golić and Biryukov *et al.* are not feasible against H.1.

For Heraclitus, more generally, these attacks will also suffer if the index string $\mathcal{X}$ is also used as part of the key. Among the reasons:

1. there are 17308 possible register length combinations; and
2. the feedback polynomials for the registers are also varying.

The advantage provided by identifying various patterns in the key stream will not be applicable since the registers are varying in length and the feedback also varies.

Thus, using Golić's method and its refinements is currently infeasible against any particular instance of Heraclitus, and against Heraclitus generally.

A.4 Algebraic Attacks

Algebraic attacks were found by Al-Hinai *et al.* [1] to be less effective than Golić's attack. Al-Hinai *et al.* provide an algebraic description of A5.1, which can be generalised to Heraclitus. Equation 1 is generalised for the movement of the registers. In this case for register R_i

$$S_i^t = c_i^t + M_r(c_1^t, \ldots, c_r^t)$$

where c_j^t is the value of the clock bit of register R_j at time t.

The functions M_r can be represented in Algebraic Normal Form: below are the representations of M_5, M_7 and M_9. A generalisation is also given without proof.

$$\begin{aligned} M_5(x_1, x_2, x_3, x_4, x_5) &= 1 + x_1x_2x_3 + x_1x_2x_4 + x_1x_2x_5 + x_1x_3x_4 \\ &\quad + x_1x_3x_5 + x_1x_4x_5 + x_2x_3x_4 + x_2x_3x_5 \\ &\quad + x_2x_4x_5 + x_3x_4x_5 + x_1x_2x_3x_4 + x_1x_2x_3x_5 \\ &\quad + x_1x_2x_4x_5 + x_1x_3x_4x_5 + x_2x_3x_4x_5. \qquad (2) \\ M_7(x_1, x_2, x_3, x_4, x_5, x_6, x_7) &= 1 + \sum_{i<j<k<l} x_ix_jx_kx_l \\ &\quad + \sum_{i<j<k<l<m} x_ix_jx_kx_lx_m \\ &\quad + \sum x_1x_2x_3x_4x_5x_6x_7. \qquad (3) \end{aligned}$$

$$
\begin{aligned}
M_9(x_1, \cdots, x_9) &= 1 + \sum_{i_1<i_2<i_3<i_4<i_5} x_{i_1}x_{i_2}x_{i_3}x_{i_4}x_{i_5} \\
&+ \sum_{i_1<i_2<i_3<i_4<i_5<i_6} x_{i_1}x_{i_2}x_{i_3}x_{i_4}x_{i_5}x_{i_6} \\
&+ \sum_{i_1<\cdots<i_7} x_{i_1}x_{i_2}x_{i_3}x_{i_4}x_{i_5}x_{i_6}x_{i_7} \\
&+ \sum_{i_1<\cdots<i_8} x_{i_1}x_{i_2}x_{i_3}x_{i_4}x_{i_5}x_{i_6}x_{i_7}x_{i_8}.
\end{aligned} \tag{4}
$$

This can be generalised for more registers. Consider majority clocking for $2n-1$ registers. In this case:

$$
\begin{aligned}
M_{2n-1}(x_1, \ldots, x_{2n-1}) &= 1 + \sum_{i_1<\cdots<i_n} x_{i_1} \cdots x_{i_n} \\
&\quad + \text{possible higher order terms.}
\end{aligned}
$$

Recall Equation 1, the derived algebraic expression for the values of the registers at time $t+1$:

$$
r_i^{t+1}[j] = \begin{cases} S_i^t \cdot r^t[j-1] + (S_i^t + 1) \cdot r^t[j] & j > 1, \text{and} \\ S_i^t \cdot F_i^t + (S_i^t + 1) \cdot r^t[j] & j = 1, \end{cases}
$$

where F_i^t is the value of the feedback polynomial of R_i at time t. Note that for each keystream bit, r second order terms are generated, and many more higher order terms. In the case of $r = 7$, 56 fourth and fifth order terms are added. This significantly increases the number of monomials in the algebraic attack, thereby making it infeasible.

Thus, Heraclitus is immune to algebraic attacks.

B Alternatives to Element Generation

In this section we discuss the reasons why the particular methods for element generation were chosen for the cipher and some of the alternatives that were considered.

B.1 Number of Registers

An initial concern was to ensure that Heraclitus was as close as possible in structure to A5.1. The goal was to be able to reuse the analysis of A5.1 and the attacks against A5.1. These were to be compared with Heraclitus.

This meant that in the early designs of Heraclitus, the option of using three registers was included. However, this was eventually discarded for the following reasons:

- three registers provided for very few options for register length, and some would have required the registers to be longer than 64 bits;
- three registers also presented difficulties ensuring that the index was used efficiently; and
- concerns that the analyses which effectively removed one register significantly weakened these ciphers.

These reasons lead to the use of five or more registers.

B.2 Length of Registers

The register lengths have to satisfy a number of constraints — ensuring that they are not too long to be difficult to implement and not to sure to allow easy enumeration and analysis *et cetera*. Co-primality of the register lengths is necessary to provide protection against the correlation attacks against the cipher.

Initially, the requirement on register lengths was that the were to be a coprime partition of a particular number, such as 128 or 256, which is related to the key size of the cipher.

Erdős and Richmond [6] provide asymptotic estimates of the number of coprime partitions, which are also coprime to the number being parted. For 128, there are about 20 coprime partitions, and for 256 there are about 29. These increase to 29 and 40 for numbers close to 128 and 256 (127 and 257 respectively).

The numbers in each of these partitions is also coprime to the original number. This is beyond the requirement for Heraclitus, thus, these partitions can be considered as well as all partitions in which one of the parts is even. This means that for 128 and 256, the initial partition will have two even numbers: one of these can then be parted into a coprime partition. This will not guarantee that each partition will be coprime to the other even number.

Estimates based on Erdős and Richmond could provide a rough estimate of the number of partitions formed in this way. The initial estimates did not account for repetitions and other unsuitable partitions. This method was judged not to be satisfactory because:

- not all the partitions generated in this way will be usable — the number of coprime parts will be either more or less than the number of registers and some of the values may be too small for register sizes, for example, those with numbers less than 10, or more than 64;
- the extension of the partitions as described above will not guarantee that they will be coprime; and
- there is no obvious algorithm to map a bit stream to a partition.

It was then observed that the length of the registers do not have to add to a number: they need only exceed the length of the key. This freed the lengths from being a partition of a particular number. The original approach was discarded.

B.3 Selection of Irreducible Polynomials

Since the feedback tap selection is dependent on the generation of irreducible polynomials. The algorithm we used generates irreducible polynomials which are unique to cyclotomic cosets of elements. Thus, to ensure index efficiency we attempted to index these cosets using bit strings of appropriate length.

This approach was not readily amenable to simple implementation and was abandoned in favour of the approach above.

An alternative approach was proposed: if the number of irreducible polynomials for each register length is kept to less than 65536 (2^{16}) then they can be generated and stored in a table. This will ensure that each string will generate a unique irreducible polynomial, thereby achieving maximum index efficiency.

In practice, the alternative approach would be implemented. The algorithm above was described for generality and extensibility.

B.4 Clocking Mechanism

The clocking mechanism chosen was deliberately chosen to be the same as A5.1 so that the earlier analyses could be easily applied.

The clocking mechanism will be same as for A5.1 Many alternative clocking mechanisms could have been chosen.

- Bent Functions: the clocking bits are put through a bent function and c_i + bent function will determine the clocking for R_i.
- Step1-Step2: majority of registers are clocked two steps, the minority are clocked only one.
- Step-n–Step-m: the majority of registers are clocked by the number of registers in the majority, the minority are clocked by the number in the minority. For example, for five registers, if three have a clock bit of 1 and two have a clock bit of 0, the three with a clock bit of 1 are clocked three times, and the two with clock bits of 0 are clocked twice.

There are many variations. An extension of Heraclitus is that the clocking mechanism could be chosen from a set of clocking mechanisms by bits of $\mathcal{X}$.

Of these choices, bent functions are expected to be used since they provide the maximum non-linearity and greatest resistance to correlation attacks.

C Algorithms

Algorithm 1 Determining the Number of Registers

if $\xi_1 = 0$ **then**
  $r \leftarrow 9$
  $\alpha_0 = \xi_2 \dots \xi_{14}$
  $\zeta = \{19, 23, 25, 27, 29, 31, 32, 37, 41, 43, 47, 49, 53, 59, 61\}$
else if $\xi_1 \xi_2 = 10$ **then**
  $r \leftarrow 7$
  $\alpha_0 = \xi_3 \dots \xi_{15}$
  $\zeta = \{19, 23, 25, 27, 29, 31, 32, 37, 41, 43, 47, 49, 53, 59, 61\}$
else
  $r \leftarrow 5$
  $\alpha_0 = \xi_3 \dots \xi_{12}$
  $\zeta = \{25, 27, 29, 31, 32, 37, 41, 43, 47, 49, 53, 59, 61\}$
end if

Algorithm 2 Generating Selection String

input r, z, a
$\beta \leftarrow \{\}$
$i \leftarrow 1$
$j \leftarrow 1$
for $1 \leq i \leq r$ **do**
  $x \leftarrow \begin{pmatrix} z-j \\ r-i \end{pmatrix}$
  while $a \geq x$ **do**
    $\beta \| 0$
    $j \leftarrow j + 1$
    $x \leftarrow x + \begin{pmatrix} z-j \\ r-i \end{pmatrix}$
  end while
  $\beta \leftarrow \beta \| 1$
  $a \leftarrow a - x + \begin{pmatrix} z-j \\ r-i \end{pmatrix}$
  $j \leftarrow j + 1$
end for
while $j \leq z + 1$ **do**
  $\beta \leftarrow \beta \| 0$
  $j \leftarrow j + 1$
end while

Algorithm 3 `test` — Irreducibility Test

input $p(x)$ irreducible polynomial of order n; $\alpha = \alpha_1 \cdots \alpha_n$
output $T \in \{\text{true}, \text{false}\}$
$\alpha \leftarrow \sum_{i=1}^{n} \alpha_i x^{i-1}$
$j \leftarrow 1$
$\beta \leftarrow \alpha^2 \text{mod} p(x)$
while $\beta \neq \alpha$ **do**
  $j \leftarrow j + 1$
  $\beta \leftarrow \beta^2 \text{mod} p(x)$
end while
if $j = n$ **then**
  $T = \text{true}$
else
  $T = \text{false}$
end if

Algorithm 4 Polynomial Generation

input $\mathcal{X} = \xi_1 \cdots \xi_{128}$, $\{\rho_i\}_{i=1}^{r}$, $\{\mathcal{P}_{\rho_i}\}_{i=1}^{r}$
output $\{\mathcal{Q}_i\}_{i=1}^{r}$
$H = \eta_1 \cdots \eta_{512} \leftarrow \mathrm{SHA}_{512}(\mathcal{X})$
$l \leftarrow 0$
for $1 \leq j \leq r$ **do**
  $\alpha = \eta_{l+1} \cdots \eta_{l+\rho_j}$
  $l \leftarrow l + \rho_j$
  if $\rho_j \notin \{25, 27, 32, 49\}$ **then**
    if $\alpha \notin \{0, 1\}$ **then**
      $\mathcal{Q}_j = \prod_{n=1}^{\rho_j} (x - \alpha^{2^n})$
    else
      $\mathcal{Q}_j = \mathcal{P}_{\rho_j}$
    end if
  else
    $k \leftarrow 1$
    $T \leftarrow \mathtt{test}(\alpha, \mathcal{P}_{\rho_j})$
    while $T =$ false **and** $k \leq 3$ **do**
      $l \leftarrow l + 1$
      $\alpha = \eta_{l+1} \cdots \eta_{l+\rho_j}$
      $T \leftarrow \mathtt{test}(\alpha, \mathcal{P}_{\rho_j})$
      $k \leftarrow k + 1$
    end while
    if $T =$ false **or** $\alpha \in \{0, 1\}$ **then**
      $\mathcal{Q}_j = \mathcal{P}_{\rho_j}$
    else
      $\mathcal{Q}_j = \prod_{n=1}^{\rho_j} (x - \alpha^{2^n})$
    end if
  end if
end for

Algorithm 5 Subkey Generation

input $\mathcal{K} = \kappa_1 \cdots \kappa_{128}$, r
case $r = 5$
$K_1 = \kappa_1 \cdots \kappa_{22} \| \text{check bits}$
$K_2 = \kappa_{23} \cdots \kappa_{46} \| \text{check bits}$
$K_3 = \kappa_{47} \cdots \kappa_{72} \| \text{check bits}$
$K_4 = \kappa_{73} \cdots \kappa_{100} \| \text{check bits}$
$K_5 = \kappa_{101} \cdots \kappa_{128} \| \text{check bits}$
endcase
case $r = 5$
$K_1 = \kappa_1 \cdots \kappa_{12} \| \text{check bits}$
$K_2 = \kappa_{13} \cdots \kappa_{25} \| \text{check bits}$
$K_3 = \kappa_{26} \cdots \kappa_{42} \| \text{check bits}$
$K_4 = \kappa_{43} \cdots \kappa_{61} \| \text{check bits}$
$K_5 = \kappa_{63} \cdots \kappa_{82} \| \text{check bits}$
$K_6 = \kappa_{83} \cdots \kappa_{104} \| \text{check bits}$
$K_7 = \kappa_{105} \cdots \kappa_{128} \| \text{check bits}$
endcase
case $r = 9$
$K_1 = \kappa_1 \cdots \kappa_{14} \| \text{check bits}$
$K_2 = \kappa_{15} \cdots \kappa_{28} \| \text{check bits}$
$K_3 = \kappa_{29} \cdots \kappa_{42} \| \text{check bits}$
$K_4 = \kappa_{43} \cdots \kappa_{56} \| \text{check bits}$
$K_5 = \kappa_{57} \cdots \kappa_{70} \| \text{check bits}$
$K_6 = \kappa_{71} \cdots \kappa_{84} \| \text{check bits}$
$K_7 = \kappa_{85} \cdots \kappa_{98} \| \text{check bits}$
$K_8 = \kappa_{99} \cdots \kappa_{112} \| \text{check bits}$
$K_9 = \kappa_{113} \cdots \kappa_{128} \| \text{check bits}$
endcase