The RAKAPOSHI Stream Cipher

Carlos Cid[1], Shinsaku Kiyomoto[2], and Jun Kurihara[2]

[1] Information Security Group,
Royal Holloway, University of London
Egham, United Kingdom
carlos.cid@rhul.ac.uk

[2] KDDI R & D Laboratories Inc.
2-1-15 Ohara, Fujimino-shi, Saitama 356-8502, Japan
kiyomoto@kddilabs.jp, kurihara@kddilabs.jp

Abstract. In this paper, we introduce the RAKAPOSHI stream cipher. The algorithm is based on Dynamic Linear Feedback Shift Registers, with a simple and potentially scalable design, and is particularly suitable for hardware applications with restricted resources. The RAKAPOSHI stream cipher offers 128-bit security, and aims to complement the current eSTREAM portfolio of hardware-oriented stream ciphers.

1 Introduction

A stream cipher is a type of encryption algorithm that encrypts individual alphabet elements of plaintext, one at a time, with a time-varying transformation [28]. Stream ciphers are very popular due to their many attractive features: they are generally fast, can typically be efficiently implemented in hardware, have no (or limited) error propagation, and are particularly suitable for use in environments where no buffering is available and/or plaintext elements need to be processed individually. These are particularly important features in the telecommunication sector, and stream ciphers are ubiquitous in the field.

Recent years have witnessed an increase in the research of design and analysis of stream ciphers, primarily motivated by eSTREAM, the ECRYPT Stream Cipher Project [12]. eSTREAM was a multi-year project, which started in 2004, and had the objective of selecting a portfolio of promising stream cipher designs. The selection of algorithms was based on two usage profiles, corresponding to specific applications identified for stream ciphers of dedicated design:

- Profile 1: stream ciphers for software applications with high throughput.
- Profile 2: stream ciphers for hardware applications with highly restricted resources.

The project received 34 submissions, of which 16 were selected to the final phase [31]. The final portfolio was announced in April 2008, containing eight ciphers: four in profile 1 and four in profile 2 [4]. The portfolio was later revised [3], due to new cryptanalytic results [15] against one of the selected ciphers in profile 2 (namely, the F-FSCR-H stream cipher [1]).

Despite the end of the eSTREAM project, the research area of analysis and design of stream ciphers remains active, with particularly eSTREAM portfolio ciphers continuing to attract much attention of the cryptographic community [8]. In this trend, we propose in this paper a new stream cipher, called RAKAPOSHI. The algorithm presents a simple and potentially scalable design, and is particularly suitable for hardware applications with restricted resources (and thus it would fall into profile 2 of the eSTREAM project). The motivation for such a proposal soon after the end of the eSTREAM project is manifold:

- The cipher provides 128-bit security. Most eSTREAM candidates in profile 2 employ 80-bit keys, as suggested in the original call for proposals[1]. However we believe 128-bit security is a very attractive feature when aiming for long-term deployment of such ciphers.
- The cipher uses a simple and elegant design, based on Dynamic Linear Feedback Shift Registers (Section 2). The design can be seem as a generalisation of constructions found in early designs [6,33,24], and may motivate a more detailed mathematical analysis of properties of such constructions.
- The cipher design and security evaluation incorporates lessons learned during the several years of extensive analysis in the eSTREAM process, and thus RAKAPOSHI is less likely to be susceptible to more recent attacks, such as initialisation attacks.
- More importantly, as noted by the eSTREAM selection committee, ciphers in the final portfolio are still very new, and analysis may not be mature enough to consider widespread deployment [3]. This was indeed illustrated by the need to revise the eSTREAM portfolio soon after its announcement. eSTREAM algorithms in profile 2 may in fact become the most popular stream ciphers in practice, due to the growing use of cryptographic mechanisms in small electronic devices (such as RFIDs). Thus we believe that alternatives to eSTREAM ciphers in this particular profile may prove to be desirable for future use.

In summary, we believe that RAKAPOSHI complements the current eSTREAM portfolio in profile 2, while presenting some attractive additional features, increasing thus the choice of secure lightweight stream ciphers suitable for hardware applications with restricted resources.

This paper is organised as follows. In Section 2 we give a brief overview of the main component of the RAKAPOSHI design (namely, DLFSR - Dynamic Linear Feedback Shift Registers). In Section 3 we describe the details of the cipher specification and design, and in Section 4 we describe the cipher operation. In Section 5 we present a provisional security evaluation of the cipher. In Section 6 we discuss some implementation aspects of the RAKAPOSHI cipher, and present our concluding remarks in Section 7.

[1] We note however that some designers later also defined 128-bit versions of the original 80-bit submissions to eSTREAM.

2 Dynamic Linear Feedback Shift Registers

A Dynamic Linear Feedback Shift Register (DLFSR) scheme is a general construction consisting usually of two registers: the first subregister $\mathcal{A}$, of length r, is clocked regularly and updated using a fixed mapping $\lambda_{\mathcal{A}}$. Subregister $\mathcal{B}$, of length n, is updated using a linear mapping $\lambda^t_{\mathcal{B}}$, which varies with time and depends of the state in register $\mathcal{A}$ at time t. Thus subregister $\mathcal{A}$ is used to select and dynamically modify the feedback function of LFSR $\mathcal{B}$, and as a result, $\mathcal{B}$ presents an irregular updating mechanism, as opposed to the regular clocking that the register would present if a unique, fixed linear feedback function was used. In practice, the state of $\mathcal{B}$ is used as input to a non-linear function to produce the output sequence of a stream cipher.

Although used in early ciphers, the general concept of Dynamic Linear Feedback Shift Registers seems to have been first proposed in open literature in the short article by Mita *et al* [29]. A simple example was presented, and properties of the output sequence were studied and compared with the output of a conventional LFSR of similar size. A more detailed characterisation of DLFSRs was presented in [27]. We note that we may generalise the idea to have non-linear updating functions, or possibly more than two subregisters.

Examples of stream ciphers based on the Dynamic Linear Feedback Shift Register primitive include the stop-and-go generator [6], LILI [33], dynamic feedback polynomial switch [21], and K2 [24]. The RAKAPOSHI stream cipher is in fact a successor of the K2 stream cipher, but aiming at low-cost hardware implementations. Furthermore, in the RAKAPOSHI design, the FSR $\mathcal{A}$ uses a non-linear updating function, and the state of both subregisters are used as input to a non-linear output function to produce the cipher's keystream.

3 Cipher Design

In this section we describe the main design criteria for the RAKAPOSHI stream cipher, as well as the details of the cipher specification.

3.1 Design Criteria

The main criteria used in the design of RAKAPOSHI were: 128-bit security, use of elegant and structurally rich FSR-based construction, and competitive performance and hardware implementation requirements when compared to ciphers in profile 2 of the eSTREAM portfolio. Furthermore, the cipher design is such that increasing the speed of the algorithm may be efficiently done by implementation of circuits for parallelization of the algorithm.

The RAKAPOSHI parameters are the following:

Key length:	128 bits.
Initialisation Vector length:	192 bits.
NLFSR $\mathcal{A}$ length r:	128 bits.
DLFSR $\mathcal{B}$ length n:	192 bits.

Thus the size of the internal state $\mathcal{S} = (\mathcal{A}, \mathcal{B})$ is $s = r + n = 320$ bits. When aiming for 128-bit security, we note that the internal state size of the cipher should be at least 320 bits, in view of the TMTO attack recently proposed by Dunkelman and Keller [11] (see Section 5.1).

We define below the notation used in this paper:

User-provided Key:	$\{k_0, k_1, ..., k_{127}\}$
Initialisation Vector (IV) :	$\{iv_0, iv_1, ..., iv_{191}\}$
Registers of NLFSR at time t:	$[a_t, a_{t+1}, ..., a_{t+127}] = \mathcal{A}^t$
Registers of DLFSR at time t:	$[b_t, b_{t+1}, ..., b_{t+191}] = \mathcal{B}^t$
Keystream at time t:	z_t

3.2 RAKAPOSHI Specification

The RAKAPOSHI stream cipher main component is the bit-oriented Dynamic Linear Feedback Shift Register (DLFSR). It consists of a 128-bit Non-Linear Feedback Shift Register and a 192-bit Linear Feedback Shift Register, denoted as registers $\mathcal{A}$ and $\mathcal{B}$, respectively. The cipher uses two bits from the state of the NLFSR to select, and dynamically modify the (linear) feedback function of the LFSR. The cipher keystream is produced by combining the output of both registers with the output of a non-linear Boolean function over $(\mathbb{F}_2)^8$. This function takes as input six bits from the state of register $\mathcal{B}$ and two bits from the state of register $\mathcal{A}$. The cipher schematic is depicted in Figure 1.

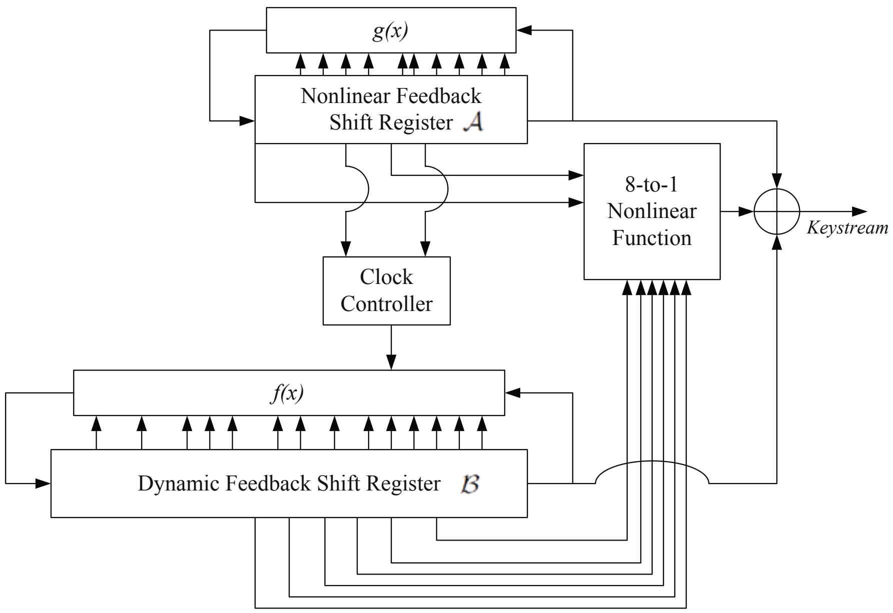
g(x)
Nonlinear Feedback Shift Register $\mathcal{A}$
8-to-1 Nonlinear Function
Keystream
Clock Controller
f(x)
Dynamic Feedback Shift Register $\mathcal{B}$

Fig. 1. RAKAPOSHI Stream Cipher

Non-Linear Feedback Shift Register $\mathcal{A}$. The cipher's register $\mathcal{A}$ is a 128-bit NLSFR, defined using the feedback function

$$\begin{aligned} g(x_0,x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9) = {} & x_1x_3x_9 + x_1x_7x_9 + x_5x_8 + x_2x_5 + \\ & x_3x_8 + x_2x_7 + x_9 + x_8 + x_7 + x_6 + \\ & x_5 + x_4 + x_3 + x_2 + x_1 + x_0 + 1, \end{aligned}$$

as $a_{t+128} = g(a_t, a_{t+6}, a_{t+7}, a_{t+11}, a_{t+16}, a_{t+28}, a_{t+36}, a_{t+45}, a_{t+55}, a_{t+62})$. Explicitly, we have the following recurrence relation:

$$\begin{aligned} a_{t+128} = {} & 1 \oplus a_t \oplus a_{t+6} \oplus a_{t+7} \oplus a_{t+11} \oplus a_{t+16} \oplus a_{t+28} \oplus a_{t+36} \oplus a_{t+45} \oplus \\ & a_{t+55} \oplus a_{t+62} \oplus a_{t+7}a_{t+45} \oplus a_{t+11}a_{t+55} \oplus a_{t+7}a_{t+28} \oplus \\ & a_{t+28}a_{t+55} \oplus a_{t+6}a_{t+45}a_{t+62} \oplus a_{t+6}a_{t+11}a_{t+62}. \end{aligned}$$

The feedback function g was selected according to criteria for non-linear feedback shift registers with maximum period [34] and consists of a primitive linear feedback shift register and a non-linear function of degree 3. This is a balanced function, of which the best linear approximation has bias 2^{-4}.

Linear Feedback Shift Register $\mathcal{B}$. The register $\mathcal{B}$ is a 192-bit Dynamic LFSR, which can use four different linear recursive functions. These are selected using two bits from the state of register $\mathcal{A}$. Let c_0 and c_1 be the 42nd and 90th bits of register $\mathcal{A}$ at time t, respectively (that is, $c_0 = a_{t+41}$ and $c_1 = a_{t+89}$). Then LFSR $\mathcal{B}$ at time t is defined by the following characteristic polynomial:

$$\begin{aligned} f(x) = {} & x^{192} + x^{176} + c_0x^{158} + (1+c_0)x^{155} + c_0c_1x^{136} + \\ & c_0(1+c_1)x^{134} + c_1(1+c_0)x^{120} + (1+c_0)(1+c_1)x^{107} + \\ & x^{93} + x^{51} + x^{49} + x^{41} + x^{37} + x^{14} + 1. \end{aligned}$$

We note that $f(x) \in \mathbb{F}_2[x]$ is a primitive polynomial for all four choices of (c_0, c_1). Thus the recurrence relation for register $\mathcal{B}$ is given by:

$$\begin{aligned} b_{t+192} = {} & b_t \oplus b_{t+14} \oplus b_{t+37} \oplus b_{t+41} \oplus b_{t+49} \oplus b_{t+51} \oplus b_{t+93} \oplus \\ & \overline{c_0} \cdot \overline{c_1} \cdot b_{t+107} \oplus \overline{c_0} \cdot c_1 \cdot b_{t+120} \oplus c_0 \cdot \overline{c_1} \cdot b_{t+134} \oplus c_0 \cdot c_1 \cdot b_{t+136} \oplus \\ & \overline{c_0} \cdot b_{t+155} \oplus c_0 \cdot b_{t+158} \oplus b_{t+176}, \end{aligned}$$

where $\overline{c}_i = 1 \oplus c_i$ represents the negation of c_i.

Non-Linear Filter. RAKAPOSHI uses as non-linear filtering function the 8-to-1 Boolean function obtained by considering the input as an element of the field $\mathbb{F}_2[x]/\langle p(x)\rangle \simeq \mathbb{F}_{2^8}$, where $p(x) = x^8 + x^4 + x^3 + x + 1$, and extracting the least significant bit of the result of the inverse operation in this field (with $0 \mapsto 0$). We note that this is the same function used as the non-affine component of the AES S-Box. This function $v(x_0,x_1,x_2,x_3,x_4,x_5,x_6,x_7)$ is a balanced Boolean function, with polynomial representation (ANF) of degree 7. We give the explicit polynomial expression of the function $v(\cdot)$ in the Appendix A.

In the RAKAPOSHI stream cipher, the input bits for the function v are extracted from both registers $\mathcal{A}$ and $\mathcal{B}$, as

$$s_t = v(a_{t+67}, a_{t+127}, b_{t+23}, b_{t+53}, b_{t+77}, b_{t+81}, b_{t+103}, b_{t+128}).$$

We note that both the sets $B = \{b_{t+23}, b_{t+53}, b_{t+77}, b_{t+81}, b_{t+103}, b_{t+128}\}$ and $A = \{a_{t+67}, a_{t+127}\}$ are full-positive difference sets [14].

4 Cipher Operation

In this section we describe the details of the cipher operation.

4.1 Keystream Generation

The cipher outputs one keystream bit at each cycle. Given the cipher state $\mathcal{S}^t = (\mathcal{A}^t, \mathcal{B}^t)$ at time t, the cipher operates as follows:

1. The keystream bit z_t is computed as $z_t = b_t \oplus a_t \oplus s_t$ and is output.
2. $c_0, c_1 \in \mathcal{A}^t$ are used to compute the updating function λ_B^t for $\mathcal{B}$.
3. Registers $\mathcal{A}$ and $\mathcal{B}$ are updated to obtain $\mathcal{A}^{t+1} = \lambda_A(\mathcal{A}^t)$ and $\mathcal{B}^{t+1} = \lambda_B^t(\mathcal{B}^t)$, respectively.

In typical operational mode, a fixed key and initialisation vector must not be used to produce more than 2^{64} keystream bits. Thus the cipher must be re-initialized (potentially by only modifying the IV) after at most 2^{64} cycles.

4.2 Initialisation Process

Before start producing the keystream, RAKAPOSHI goes through an initialisation process, in which the secret key and IV are loaded into the registers and mixed.

The secret key $\{k_0, k_1, ..., k_{127}\}$ and IV $\{iv_0, iv_1, ..., iv_{191}\}$ are loaded into the NLFSR and DLFSR, respectively, as follows:

$$\begin{aligned} a_0, a_1, ..., a_{127} &\leftarrow k_0, k_1, ..., k_{127}, \\ b_0, b_1, ..., b_{191} &\leftarrow iv_0, iv_1, ..., iv_{191}. \end{aligned}$$

The cipher then clocks 448 times with the output of the filter function $v(\cdot)$ (that is, s_t rather than z_t) being fed back into the cipher state. This process is divided into two stages:

– Stage 1

  In the first stage of the initialisation, the cipher runs for 320 cycles, with the output of the non-linear filter function $v(\cdot)$ being fed back into the register $\mathcal{B}$. Thus during this stage the LFSR $\mathcal{B}$ uses the following recurrence relation:

$$\begin{aligned} b_{t+192} = {} & b_t \oplus b_{t+14} \oplus b_{t+37} \oplus b_{t+41} \oplus b_{t+49} \oplus b_{t+51} \oplus b_{t+93} \oplus \\ & \overline{c_0} \cdot \overline{c_1} \cdot b_{t+107} \oplus \overline{c_0} \cdot c_1 \cdot b_{t+120} \oplus c_0 \cdot \overline{c_1} \cdot b_{t+134} \oplus \\ & c_0 \cdot c_1 \cdot b_{t+136} \oplus \overline{c_0} \cdot b_{t+155} \oplus c_0 \cdot b_{t+158} \oplus b_{176} \oplus s_t. \end{aligned}$$

– Stage 2

In the second stage of the initialisation, the cipher runs for further 128 cycles, with the output of the non-linear filter function $v(\cdot)$ being fed back into the register $\mathcal{A}$. Thus during this stage the NLFSR $\mathcal{A}$ uses the following recurrence relation:

$$\begin{aligned} a_{t+128} = 1 &\oplus a_t \oplus a_{t+6} \oplus a_{t+7} \oplus a_{t+11} \oplus a_{t+16} \oplus a_{t+28} \oplus a_{t+36} \oplus a_{t+45} \oplus \\ &a_{t+55} \oplus a_{t+62} \oplus a_{t+7}a_{t+45} \oplus a_{t+11}a_{t+55} \oplus a_{t+7}a_{t+28} \oplus \\ &a_{t+28}a_{t+55} \oplus a_{t+6}a_{t+45}a_{t+62} \oplus a_{t+6}a_{t+11}a_{t+62} \oplus s_t. \end{aligned}$$

At the end of the initialisation, the cipher internal state is $\mathcal{S}^0 = (\mathcal{A}^0, \mathcal{B}^0)$, and it is ready to produce the first output bit of the keystream z_0, as specified in Section 4.1.

5 Security Evaluation

In this section we provide a provisional security analysis of the RAKAPOSHI cipher, taking into account the most common cryptanalytic methods against stream ciphers.

5.1 Time-Memory Trade-Off Attacks

Time-Memory Trade-off attacks are generic attacks against several cryptographic constructions, consisting of a precomputation phase and an online phase. In a typical attack scenario, one would perform extensive offline computation and store the results, with the goal of reducing the time complexity of the online attack. Time-Memory Trade-off attacks were originally proposed by Hellman [18] for attacking the DES block cipher.

In the context of stream ciphers, TMTO attacks were first proposed by Babbage [2], and later improved by Biryukov and Shamir [7]. In these attacks, one tries to invert the function mapping the cipher internal state to a segment of the keystream output. As a result, to prevent against such attacks a cipher should have its internal state at least twice as long as its key length.

On a different attack scenario, proposed by Hong and Sarkar [19,20], the function inverted maps the key/IV to a keystream segment. In their original approach, the IV is treated as part of the secret key, and as a result, prevention against such attack scenario requires stream ciphers to have the IV at least as long as the key. More recently Dunkelman and Keller [11] proposed a new approach, in which an attacker selects in advance several IVs, and mounts the attack to invert the function mapping the key to a keystream segment (with the chosen IVs). As a result, it follows that if n is the cipher key length, then the IV length must be at least $\frac{3}{2}n$ to withstand this form of TMTO attack.

The RAKAPOSHI stream cipher uses 128-bit keys and 192-bit IVs, with internal state with 320 bits. Therefore it withstands the currently known TMTO attacks against stream ciphers.

5.2 Guess-and-Determine Attacks

In guess-and-determine attacks, the attacker guesses a subset of the cipher internal state, and recovers further bits from the state based on the observed keystream and guessed values. This procedure may be repeated for other values until the full state and/or the secret key is recovered.

For the RAKAPOSHI stream cipher, one may guess a selected subset of the state of register $\mathcal{A}$ and hope to recover bits from $\mathcal{B}$ (since $\mathcal{A}$ is the source of non-linearity to both registers). A naïve attack would be to guess all bits of the NLFSR $\mathcal{A}$, and then recover the state of the DLFSR $\mathcal{B}$. This attack is however essentially equivalent to a brute force attack on the cipher key, and is thus considered infeasible.

We have considered a number of alternative scenarios, in which an attacker guesses a number of bits of the cipher's internal state, and tries to recover further bits from the observed keystream. However due to the RAKAPOSHI internal structure (its registers' taps and input to non-linear filtering function), we have not been able to come up with an efficient attack. Thus we believe that RAKAPOSHI is not susceptible to guess-and-determine attacks.

5.3 Distinguishing Attacks

In distinguishing attacks, one attempts to find ways to distinguish the stream cipher (considered as a bit generator) from a purely random source of binary digits. It is notoriously difficult (if not impossible) to provide assurance that a cipher is not susceptible to distinguishing attacks. In fact some *secure* constructions (for instance, AES in counter mode) can be trivially distinguished from a random source [16], and as a result several researchers dispute the general applicability of some forms of distinguishing attacks against stream ciphers.

For RAKAPOSHI, we tried to construct linear distinguishers, by considering linear approximations of functions used in the cipher. For instance, the best affine approximation of the nonlinear filter has bias $\epsilon = 2^{-6}$, and the function uses as input 6 bits from register $\mathcal{B}$. For fixed values of c_0, c_1, the number of terms in the register's recurrence function f is 11. Now, if we omit the 2-bit input from the NLFSR for the nonlinear filter, we can construct a linear distinguisher using 11 keystream bits as follows:

$$D : z_t \oplus z_{t+14} \oplus z_{t+37} \oplus z_{t+41} \oplus z_{t+49} \oplus z_{t+51} \oplus z_{t+93} \oplus z_{t+107} \oplus z_{t+155} \oplus z_{t+176} \oplus z_{t+192} = 0,$$

where we assume two clock bits c_0, c_1 from the NLFSR are $(c_0, c_1) = (0, 0)$ for each feedback function accompanying the distinguisher D. Using the pilling-up lemma [26], the bias of the linear distinguisher is estimated as $2^{10} * (2^{-6})^{11} = 2^{-56}$. We note however that the two clock control bits in the linear recurrence f have to be determined to obtain the correct equation. As a result, the data complexity of this particular distinguishing attack for the cipher increases to $((1/2^{-56}) * 2^{2*6})^2 = 2^{136}$. Thus even by ignoring the effects of the NLFSR $\mathcal{A}$, this distinguishing attack can be considered infeasible.

We have also considered several other linear approximations, which are better than the above scenario; however, due to effects of the NLFSR, we have not been able to find a more efficient linear approximation. Thus, we expect that the cipher is secure against distinguishing attacks.

5.4 Algebraic Attacks

Algebraic attacks against stream ciphers were originally proposed in 2003 by Courtois and Meier [9]. The attack is a powerful cryptanalytic technique against some LFSR-based stream cipher constructions (e.g. the filter generator). When mounting an algebraic attack, one attempts to construct a system of equations derived from the cipher operation, which can then be solved using a choice of methods. A stream cipher designer can protect the algorithm against algebraic attacks by using to compute the keystream, a non-linear filter function $z(\cdot)$ of high degree, for which the annihilating sets of both z or its complement $z+1$ contain no low degree polynomials (the lowest degree of polynomials in either of these sets is called the *algebraic immunity* of z).

For the RAKAPOSHI stream cipher, the keystream output z_t is computed by means of a Boolean function over $(\mathbb{F}_2)^{10}$ given by

$$z_t = s_t + a_t + b_t = v(a_{t+67}, a_{t+127}, b_{t+23}, b_{t+53}, b_{t+77}, b_{t+81}, b_{t+103}, b_{t+128}) + a_t + b_t.$$

This function $z(\cdot)$ has degree 7 and algebraic immunity 4 (which is the same of the function v). Indeed, there are 21 linearly independent functions of degree 4 in the annihilating set of both z and $z+1$. However both registers $\mathcal{A}$ and $\mathcal{B}$ are updated non-linearly (by functions g and f respectively). This increases very rapidly the degree and the complexity of the polynomial expression of the keystream output bits; this was verified experimentally for several clocks of the cipher. Thus conventional algebraic attacks do not seem to work against RAKAPOSHI.

We note an alternative to the approach above. One could guess several bits of the register $\mathcal{A}$, in the hope of keeping the resulting degrees at a reasonably low value (since $\mathcal{A}$ is the source of non-linearity to both registers). A trivial, somewhat naïve algebraic attack against RAKAPOSHI would thus be to guess all the 128 values for the register $\mathcal{A}$, and generate 21 equations of degree 4 for each output bit of the cipher. If the state of register $\mathcal{A}$ is fully known, then the complexity of an algebraic attack would be on the order of

$$\left(\sum_{k=0}^{4} \binom{192}{k}\right)^3 \approx 2^{77}$$

operations, requiring approximately 2^{21} output bits. The total complexity of the attack would thus be on the order of 2^{205} operations. One could instead guess fewer bits from $\mathcal{A}$; however the results above indicate that the overall complexity of the attack would remain well above the one required for exhaustive key search. Thus we conclude that RAKAPOSHI is secure against algebraic attacks.

5.5 Analysis of the Initialisation Process

The RAKAPOSHI initialisation process consists of 448 steps. The first 320 steps affect the subregister $\mathcal{B}$, ensuring that after this initial stage, all bits of the

Table 1. Diffusion of the Initialisation Process

	Stage 1 cycles			Stage 2 cycles	
	0	293	319	105	128
All registers of NLFSR	- -	- -	- -	All KEY -	All KEY All IV
All registers of DLFSR	- -	All KEY -	All KEY All IV	All KEY All IV	All KEY All IV

DLFSR $\mathcal{B}$ have been influenced by all bits of the secret key (that is, the initial state of NLFSR $\mathcal{A}$) and all bits of the IV (that is, the initial state of DLFSR $\mathcal{B}$). The second stage consists of further 128 steps, which affects the subregister $\mathcal{A}$, and as a result all bits of the NLFSR $\mathcal{A}$ would have been influenced by all bits of the IV and the secret key at the end of this stage. This is summarised in Table 1. We have also analysed how complex the relationship between the state bits and the secret key / IV are, by computing the algebraic expression of the state bits during several steps of the initialisation. Our experiments indicate that these are dense, high-degree polynomials involving a large number of internal state bits. Thus, the number of cycles in the initialisation process seems to be sufficient for diffusing the bits of the initial state in a very complex way, and we believe that RAKAPOSHI is secure against the most recent attacks on the initialisation process [10,25,32,35].

5.6 Statistical Tests

The statistical properties of the cipher depend on the properties of the output sequences of the NLFSR and DLFSR; given their construction, we expect the keystream of the cipher to have good statistical properties. We have evaluated the statistical properties for the cipher keystream, as well as the output sequences of the NLFSR and DLFSR using the NIST Test Suite [30]. The results indicated that the statistical properties of the RAKAPOSHI output sequence are good.

6 Implementation Aspects

In this section we discuss various aspects related to the implementation and performance of the RAKAPOSHI stream cipher.

6.1 Parallelization

The throughput of RAKAPOSHI can be increased by applying parallelization techniques. It is in fact possible to increase the cipher output to 64 bits per clock cycle by adding some additional circuits. To realise n times parallelization ($1 \leq n \leq 64$), we add circuits for n feedback functions (for $f(\cdot)$ and $g(\cdot)$), n keystream generation functions (for $v(\cdot)$), and add $n-1$ bits to the NLFSR $\mathcal{A}$. The increase of registers for the DLFSR is not required for parallelization up to

Table 2. Evaluation Results of RAKAPOSHI

	Cir. Size (Slices)	Max. Cl. Freq. (MHz)	Throughput (Mbps)	Throughput / Slices
Spartan-II	199	111.6	111.6	0.56
Spartan-3	194	120.1	120.1	0.62
Spartan-3 (Opt.)	63	155.9	155.9	2.47
Vertex-II	193	218.9	218.9	1.13
Spartan-3 (×8)	342	125.5	1004.0	2.94
Spartan-3 (×16)	445	124.5	1992.0	4.48
Spartan-3 (×32)	849	125.0	4000.0	4.71
Spartan-3 (×64)	1302	95.0	6080.0	4.67

64 times, since the input for the nonlinear filter does not include any of the last 63 bits of DLFSR.

The design of RAKAPOSHI has been optimised for 32 times parallelization: the last 15 bits of the DLFSR are not used in the feedback function $f(\cdot)$. We note however that 64 times parallelization is also possible by a design criterion for NLFSR: the last 65 bits of NLFSR are not used for execution of the feedback function $g(\cdot)$.

6.2 Circuit Size and Performance

We present here the evaluation results of hardware implementations using an FPGA simulator. We implemented RAKAPOSHI targeted towards the Xilinx Spartan-II, Spartan-3, and Virtex-II FPGAs. We used Xilinx ISE 9.1 for post-place and route simulation and static timing analysis. Circuit sizes of the DLFSR, NLFSR, Nonlinear Filter function on Spartan-II are 1575 gates, 1060 gates, and 147 gates respectively. The circuit size of the whole algorithm is 3130 gates[2].

We have also implemented parallelized versions of the algorithm that produce 8-bit, 16-bit, 32-bit, and 64-bit outputs for each clock cycle. Table 2 shows evaluation results of circuit size, maximum clock frequency, throughput, and efficiency for each implementation. The efficiency is computed as throughput per slice. The efficiency improves due to circuit parallelization and it reaches its maximum by 32-times parallelization. The maximum throughput is approximately 6 Gbps using the 64-times parallelized circuit on Spartan-3.

6.3 Comparison with Other Stream Ciphers

As stated in Section 3, one of the design goals of the RAKAPOSHI stream cipher is to complement the eSTREAM portfolio in profile 2. As such, we present a comparison between various aspects of the RAKAPOSHI implementation and some of the eSTREAM ciphers in Table 3. We use in this comparison the evaluation results of circuit sizes and maximum clock frequency presented in [13,22].

[2] This corresponds to approximately one tenth of the circuit size of the K2 stream cipher [23], a predecessor of RAKAPOSHI which targets however software applications with high throughput requirements.

Table 3. Comparison on Spartan-3

Algorithm	Key Len	IV Len	Internal State	Cir. Size (Slices)	Max. Cl. Freq(MHz)	Thr/ Slices	# Cyc. for Init.
Grain [13]	80	64	160	122	193	1.58	160
Grain (opt.) [22]	80	64	160	44	196	4.45	160
Trivium [13]	80	80	288	188	206	1.10	1152
Trivium (opt.) [22]	80	80	288	50	240	4.80	1152
MICKEY-128 [13]	128	128	320	261	156	0.60	416
MICKEY-128(opt.)[22]	128	128	320	176	223	1.27	416
Grain-128(opt.)[22]	128	96	256	50	196	3.92	256
DECIM-128(opt.)[22]	128	128	352	89	43.5	0.49	1584
Rakaposhi	128	192	320	194	120.1	0.62	448
Rakaposhi (opt.)	128	192	320	63	155.9	2.47	448

We can note that RAKAPOSHI presents a competitive performance profile when compared with MICKEY-128[5] with regards to throughput per slice. Grain v1 and Trivium are however more efficient than RAKAPOSHI. We note however that these algorithms do not provide 128-bit security level, which as noted earlier, we consider a particularly attractive feature when aiming for long-term deployments. Furthermore, given those ciphers' state/key/IV sizes, they would appear to be susceptible to more recently proposed TMTO attacks (see Section 5.1). We note that the latter remark is also applicable to Grain-128 [17].

7 Conclusion

In this paper we propose a new stream cipher, called RAKAPOSHI. The algorithm presents a simple and elegant design, and is particularly suitable for hardware applications with restricted resources. We have presented a provisional security evaluation of the cipher, which indicates that it is secure against the most common attacks against stream ciphers. Furthermore, evaluation of several implementation and performance aspects shows that RAKAPOSHI is a competitive alternative to ciphers in the final eSTREAM portfolio. We believe therefore that RAKAPOSHI can complement the current eSTREAM portfolio in profile 2, while presenting some attractive additional features, increasing thus the choice of secure lightweight stream ciphers suitable for hardware applications with restricted resources.

References

1. Arnault, F., Berger, T., Lauradoux, C.: F-FCSR Stream Ciphers. In: Robshaw, M.J.B., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 170–178. Springer, Heidelberg (2008)
2. Babbage, S.: Improved "exhaustive search" attacks on stream ciphers. In: IEE European Convention on Security and Detection, vol. 408, pp. 161–165 (1995)

3. Babbage, S., De Canniere, C., Canteaut, A., Cid, C., Gilbert, H., Johansson, T., Parker, M., Preneel, B., Rijmen, V., Robshaw, M.: The eSTREAM Portfolio (rev.1), September 08 (2008), http://www.ecrypt.eu.org/stream/portfolio_revision1.pdf
4. Babbage, S., De Canniere, C., Canteaut, A., Cid, C., Gilbert, H., Johansson, T., Parker, M., Preneel, B., Rijmen, V., Robshaw, M.: The eSTREAM Portfolio, April 15 (2008), http://www.ecrypt.eu.org/stream/portfolio.pdf
5. Babbage, S., Dodd, M.: The MICKEY Stream Ciphers. In: Robshaw, M.J.B., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 191–209. Springer, Heidelberg (2008)
6. Beth, T., Piper, F.C.: The stop-and-go-generator. In: Beth, T., Cot, N., Ingemarsson, I. (eds.) EUROCRYPT 1984. LNCS, vol. 209, pp. 88–92. Springer, Heidelberg (1985)
7. Biryukov, A., Shamir, A.: Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 1–13. Springer, Heidelberg (2000)
8. Cid, C., Robshaw, M.: The eSTREAM Portfolio 2009 Annual Update, July 31 (2009), http://www.ecrypt.eu.org/stream/eStream_reportJul09.pdf
9. Courtois, N., Meier, W.: Algebraic Attacks on Stream Ciphers with Linear Feedback. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 345–359. Springer, Heidelberg (2003)
10. Dinur, I., Shamir, A.: Cube Attacks on Tweakable Black Box Polynomials. In: Joux, A. (ed.) FroCoS 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2009)
11. Dunkelman, O., Keller, N.: Treatment of the Initial Value in Time-Memory-Data Tradeoff Attacks on Stream Ciphers. Information Processing Letters 107, 133–137 (2008)
12. eSTREAM, the ECRYPT Stream Cipher Project, http://www.ecrypt.eu.org/stream/
13. Gaj, K., Southern, G., Bachimanchi, R.: Comparison of hardware performance of selected Phase II eSTREAM candidates. In: Proceedings of SASC 2007, Bochum (2007)
14. Golic, J.D.: On Security of Nonlinear Filter Generators. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 173–188. Springer, Heidelberg (1996)
15. Hell, M., Johansson, T.: Breaking the F-FCSR-H Stream Cipher in Real Time. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 557–569. Springer, Heidelberg (2008)
16. Hell, M., Johansson, T., Brynielsson, L.: An overview of distinguishing attacks on stream ciphers. Cryptography and Communications 1(1), 71–94 (2009)
17. Hell, M., Johansson, T., Maximov, A., Meier, W.: A stream cipher proposal: Grain-128. In: Proceedings of 2006 IEEE International Symposium on Information Theory, pp. 1614–1618. IEEE, Los Alamitos (2006)
18. Hellman, M.E.: A Cryptanalytic Time-Memory Tradeoff. IEEE Transactions on Information Theory 26(4), 401–406 (1980)
19. Hong, J., Sarkar, P.: New Applications of Time Memory Data Tradeoffs. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 353–372. Springer, Heidelberg (2005)
20. Hong, J., Sarkar, P.: Rediscovery of Time Memory Tradeoffs. Cryptology ePrint Archive, Report 2005/090 (2005), http://eprint.iacr.org/

21. Horan, D., Guinee, R.: A Novel Keystream Generator using Pseudo Random Binary Sequences for Cryptographic Applications. In: Irish Signals and Systems Conference 2006, pp. 451–456. IEEE, Los Alamitos (2006)
22. Hwang, D., Chaney, M., Karanam, S., Ton, N., Gaj, K.: Comparison of FPGA-Targeted Hardware Implementations of eSTREAM Stream Cipher Candidates. In: Proceedings of SASC (2008); Lausanne
23. Kiyomoto, S., Tanaka, T., Sakurai, K.: FPGA-Targeted Hardware Implementations of K2. In: Proceedings of SECRYPT 2008, pp. 270–277 (2008)
24. Kiyomoto, S., Tanaka, T., Sakurai, K.: K2: A Stream Cipher Algorithm Using Dynamic Feedback Control. In: Proceedings of SECRYPT 2007, pp. 204–213 (2008)
25. Lee, Y., Jeong, K., Sung, J., Hong, S.: Related-Key Chosen IV Attacks on Grain-v1 and Grain-128. In: Mu, Y., Susilo, W., Seberry, J. (eds.) ACISP 2008. LNCS, vol. 5107, pp. 321–335. Springer, Heidelberg (2008)
26. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
27. Medina, M., Domínguez, A.: Caracterización de Secuencias Binarias Pseudoaleatorias generadas mediante LFSR con Realimentación Dinámica (DLFSR). In: Proceedings of XVIII Simposium Nacional de la URSI, A Coruña, Spain (2003)
28. Menezes, A.J., Van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1996)
29. Mita, R., Palumbo, G., Pennisi, S., Poli, M.: Pseudorandom bit generator based on dynamic linear feedback topology. Electronic Letters 28(19), 1097–1098 (2002)
30. National Institute of Standards and Technology. NIST Statistical Test, `http://csrc.nist.gov/rng/`
31. Robshaw, M., Billet, O.: New Stream Cipher Designs. LNCS, vol. 4986. Springer, Heidelberg (2008)
32. Khazaei, S., Fischer, S., Meier, W.: Chosen IV Statistical Analysis for Key Recovery Attacks on Stream Ciphers. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 236–245. Springer, Heidelberg (2008)
33. Simpson, L.R., Dawson, E., Golic, J., Millan, W.: LILI Keystream Generator. In: Stinson, D.R., Tavares, S. (eds.) SAC 2000. LNCS, vol. 2012, pp. 248–261. Springer, Heidelberg (2001)
34. Soriano, M.: Stream ciphers based on NLFSR. In: Proceedings of SBT/IEEE International Telecommunications Symposium 1998, pp. 528–533. IEEE, Los Alamitos (1998)
35. Vielhaber, M.: Breaking ONE.FIVIUM by AIDA an Algebraic IV Differential Attack, Cryptology ePrint archive, report 2007/413 (2007)

Appendix

A Rakaposhi Non-linear Function

The RAKAPOSHI stream cipher uses the following Boolean function to produce the keystream output.

$$\begin{gathered}
v(x_0,x_1,x_2,x_3,x_4,x_5,x_6,x_7) = \\
x_0x_1x_2x_3x_4x_5x_6 + x_0x_1x_2x_3x_4x_5 + x_0x_1x_2x_3x_4x_6 + x_0x_1x_2x_3x_5x_6x_7 + \\
x_0x_1x_2x_3x_5x_6 + x_0x_1x_2x_3x_5x_7 + x_0x_1x_2x_3x_5 + x_0x_1x_2x_3x_6x_7 + \\
x_0x_1x_2x_4x_5x_6 + x_0x_1x_2x_4 + x_0x_1x_2x_5x_6 + x_0x_1x_2x_5x_7 + x_0x_1x_2x_7 + \\
x_0x_1x_2 + x_0x_1x_3x_4x_5x_6x_7 + x_0x_1x_3x_4x_5x_7 + x_0x_1x_3x_4x_5 + x_0x_1x_3x_4x_7 + \\
x_0x_1x_3x_4 + x_0x_1x_3x_6 + x_0x_1x_4x_5x_6x_7 + x_0x_1x_4x_5x_6 + x_0x_1x_4x_5x_7 + \\
x_0x_1x_4x_6x_7 + x_0x_1x_4x_7 + x_0x_1x_5x_6x_7 + x_0x_1x_5x_6 + x_0x_1x_5 + x_0x_1x_6 + \\
x_0x_1 + x_0x_2x_3x_4x_5x_6 + x_0x_2x_3x_4x_5x_7 + x_0x_2x_3x_4 + x_0x_2x_3x_5x_6x_7 + \\
x_0x_2x_3x_5x_6 + x_0x_2x_3x_5x_7 + x_0x_2x_3x_6 + x_0x_2x_4x_5x_6x_7 + x_0x_2x_5x_6 + \\
x_0x_2x_5 + x_0x_2x_6x_7 + x_0x_2x_7 + x_0x_3x_4x_5x_6x_7 + x_0x_3x_4x_5x_6 + x_0x_3x_4x_5x_7 + \\
x_0x_3x_4x_5 + x_0x_3x_4x_7 + x_0x_3x_5x_6x_7 + x_0x_3x_5 + x_0x_3x_6 + x_0x_3 + x_0x_4x_5x_6 + \\
x_0x_4x_6x_7 + x_0x_5x_6 + x_0x_6 + x_0 + x_1x_2x_3x_4 + x_1x_2x_3x_5x_6 + x_1x_2x_3x_5x_7 + \\
x_1x_2x_3x_5 + x_1x_2x_3 + x_1x_2x_4x_5x_6 + x_1x_2x_4x_6 + x_1x_2x_4 + x_1x_2x_5 + x_1x_2 + \\
x_1x_3x_4x_5x_6x_7 + x_1x_3x_4x_5x_7 + x_1x_3x_4x_6x_7 + x_1x_3x_4x_6 + x_1x_3x_4 + \\
x_1x_3x_5x_6 + x_1x_3x_5 + x_1x_3x_6 + x_1x_3x_7 + x_1x_4x_5x_6x_7 + x_1x_4x_5x_7 + \\
x_1x_5x_6 + x_1x_5x_7 + x_1x_5 + x_1x_6x_7 + x_1x_6 + x_1 + x_2x_3x_4x_5x_6 + x_2x_3x_4x_5x_7 + \\
x_2x_3x_4x_5 + x_2x_3x_4x_6x_7 + x_2x_3x_4 + x_2x_3x_5x_7 + x_2x_3x_6x_7 + x_2x_3x_6 + \\
x_2x_4x_5x_6 + x_2x_4x_5x_7 + x_2x_4x_5 + x_2x_4x_6x_7 + x_2x_4x_6 + x_2x_4x_7 + x_2x_4 + \\
x_2x_5x_6x_7 + x_2x_6x_7 + x_2x_6 + x_2x_7 + x_3x_4x_5x_6x_7 + x_3x_4x_5 + x_3x_4x_6x_7 + \\
x_3x_4x_6 + x_3x_4x_7 + x_3x_5x_6x_7 + x_3x_6x_7 + x_3x_6 + x_3x_7 + x_4x_5x_6 + x_4x_5 + \\
x_5x_6x_7 + x_5x_6 + x_5 + x_6 + x_7.
\end{gathered}$$

B Test Vector

```
Key:    00000000000000000000000000000000
IV:     000000000000000000000000000000000000000000000000
Internal State Bits after the Initialization:
(NLFSR-A) 3c12b227eccb28a0baf327a7d42a51e5
(DLFSR-B) 619344585ae94087412e9863bd028f18f42eefe6378c5011
Keystream:
7a72bd702002121880960ed4ae0c054ecad09b0459c334866fbbd8
84aa0ff5585497943c6095d427c96eeb8719f87a02761465d0f62a
1e0faad849302104827e6db2e0b81e49a7b81ce170e4cf261468d6
6b2e6e13cfcabca1073f2077298b2c0fe0da1feb8c1e20b27f5907
b883eb17c5165113acfb2a7ca7a0c6cf3578f87c
```