A New Class of Single Cycle T-functions

Jin Hong, Dong Hoon Lee, Yongjin Yeom, and Daewan Han

National Security Research Institute
161 Gajeong-dong, Yuseong-gu
Daejeon, 305-350, Korea
{jinhong,dlee,yjyeom,dwh}@etri.re.kr

Abstract. T-function is a relatively new cryptographic building block suitable for streamciphers. It has the potential of becoming a substitute for LFSRs, and those that correspond to maximum length LFSRs are called single cycle T-functions. We present a family of single cycle T-functions, previously unknown. An attempt at building a hardware oriented streamcipher based on this new T-function is given.

Keywords: T-function, single cycle, streamcipher

1 Introduction

The appearance of algebraic attack on streamciphers[5, 11–13] has certainly made the designing of streamciphers a more difficult task. At the same time, as presentations[7, 28] and discussions during a recent streamcipher workshop has shown, the demand for streamciphers is declining. But, we have also seen at the same workshop that at the very extremes, there are still genuine needs for streamciphers. One case is when the cipher has to be ultra-fast(Gbps) in software (on relatively good platforms), as in software routers. The other extreme, namely where efficient hardware implemented ciphers for resource constrained environment is needed, could also benefit from a good streamcipher design.

Most of the recent attempts at streamcipher constructions[6, 9, 10, 14–17, 26, 29–31] are mainly focused on software, except for those based on LFSRs. In particular, most of them demand a very large memory space to store its internal state. If we turn to traditional designs that use bitwise LFSRs, which could have advantages in hardware, we find that large registers have to be used to counter algebraic attacks. In short, we have a lack of good hardware oriented streamciphers at the moment. This paper is an attempt at filling this gap.

Few years ago, Klimov and Shamir started developing the theory of T-functions[20–22]. A T-function is a function acting on a collection of memory words, with a weak one-wayness property. It started out as a tool for blockciphers, but now, its possibility as a building block for streamciphers is drawing attention.

An important class of T-functions consists of those that exhibit the *single cycle* property. This is the T-function equivalent of maximum length LFSRs and has potential to bring about a very fast streamcipher. Unfortunately only a very small family of single cycle T-functions is known to the crypto community currently.[1]

The main contribution of this work is to uncover a new class of single cycle T-functions. It is a generalization of a small subset of the previously known single cycle T-functions and it does show some good properties which the previous ones did not. We also give an example of how one might build a cipher on top of this new class of single cycle T-functions. Although previous T-functions targeted software implementations, our T-function based streamcipher is designed to be light and is suitable for constrained hardware environment.

The paper is organized as follows. We start by reviewing the basics of T-functions. In Section 3, we look into the existing single cycle T-functions and show that without the multiplicative part, which is not understood at all, it is a very simple object, far from a random function. With this new way of viewing existing T-functions, we give a new class of single cycle T-functions in Section 4. A streamcipher example built on top of the new T-functions is introduced in the following section. The last section concludes the paper.

2 Review of T-functions

We shall review the basics of multi-word T-functions in this section. Readers may refer to the original papers [20–22] for a more in-depth treatment.

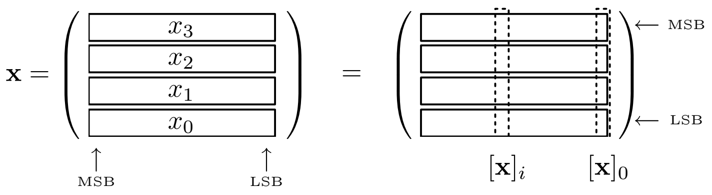
Let us consider a gathering of m-many of n-bit words, which we denote by x_i $(i = 0, \dots, m-1)$. Our main interest lies in the case $n = 32$ and $m = 4$. As a shorthand for multiple words, we shall often use the corresponding boldface letter. For example, $\mathbf{x} = (x_k)_{k=0}^{m-1}$. The i-th bit of a word x is denoted by $[x]_i$, where we always count starting from 0. Seen as an integer, we have

$$x = \sum_{i=0}^{n-1} [x]_i 2^i. \tag{1}$$

The i-th bits of the m-tuple of words $\mathbf{x}$ are denoted collectively as $[\mathbf{x}]_i$. We sometimes view $[\mathbf{x}]_i$ also as an m-bit integer by setting

$$[\mathbf{x}]_i = \sum_{k=0}^{m-1} [x_k]_i 2^k. \tag{2}$$

In reading the rest of this paper, it helps to view the various notations pictorially as follows.



[1] It seems T-function was already studied in the mathematics community under various different names[3, 4].

Accordingly, we shall sometimes refer to $[\mathbf{x}]_i$ as the i-th column.

Definition 1. *A (multi-word)* T-function *is a map*

$$\mathbf{T} : (\{0,1\}^n)^m \longrightarrow (\{0,1\}^n)^m, \qquad \mathbf{x} \mapsto \mathbf{T}(\mathbf{x}) = (T_k(\mathbf{x}))_{k=0}^{m-1}$$

sending an m-tuple of n-bit words to another m-tuple of n-bit words, where each resulting n-bit word is denoted as $T_k(\mathbf{x})$, such that for each $0 \le i < n$, the i-th bits of the resulting words $[\mathbf{T}(\mathbf{x})]_i$ are functions of just the lower input bits $[\mathbf{x}]_0, [\mathbf{x}]_1, \ldots, [\mathbf{x}]_i$.

We shall mainly be dealing with multi-word T-functions, as opposed to single-word T-functions, which is the $m = 1$ case, and hence shall mostly omit writing *multi-word.* Also, unless stated otherwise, we shall always assume a T-function to be acting on m words of n-bit size. The set of words a T-function is acting on is sometimes referred to as *memory* or *register* and the bit values it contains are said to form a *state* of the memory.

Given a T-function $\mathbf{T}$, one may fix an initial state $\mathbf{x}^0$ for the memory and iteratively act $\mathbf{T}$ on it to obtain a sequence defined by

$$\mathbf{x}^{t+1} = \mathbf{T}(\mathbf{x}^t). \tag{3}$$

Such a sequence will always be eventually periodic and if its periodic part passes through all of the 2^{nm} possible states the memory may take, the T-function is said to form a *single cycle.* A single cycle T-function may serve as a good building block for a streamcipher. To state results about single cycle T-functions, we need a few more definitions.

Definition 2. *A (multi-word)* parameter *is a map*

$$\alpha : (\{0,1\}^n)^m \longrightarrow \{0,1\}^n, \qquad \mathbf{x} \mapsto \alpha(\mathbf{x})$$

sending an m-tuple of n-bit words to a single n-bit word such that for each $0 \le i < n$, the i-th bit of the resulting word $[\alpha(\mathbf{x})]_i$ is a function of just the strictly lower input bits $[\mathbf{x}]_0, [\mathbf{x}]_1, \ldots, [\mathbf{x}]_{i-1}$.

In other words, a parameter is a sort of multi-word to single-word T-function for which the i-th output bit does not depend on the i-th input bits $[\mathbf{x}]_i$. When restricted to just linear functions acting on a single word, T-functions are exactly the upper triangular matrices and parameters correspond to the strictly upper triangular matrices.

Given a parameter α, and fixed $0 \le i < n$, we may consider the bit value

$$B[\alpha, i] = \bigoplus_{\mathbf{x}=(0,\ldots,0)}^{(2^i-1,\ldots,2^i-1)} [\alpha(\mathbf{x})]_i. \tag{4}$$

Notice that since $[\alpha(\mathbf{x})]_i$ does not depend on any of the high indexed input bits, going any higher in this direct sum would be meaningless. Also, note that $[\alpha(\mathbf{x})]_0$

is constant for all input $\mathbf{x}$ and the sum $B[\alpha, 0]$ is equal to this constant value. If the value $B[\alpha, i] = 0$ for all i, the parameter is said to be *even.* Likewise, if it is 1 for all i, the parameter is *odd.*

Currently, it seems that the only single cycle T-functions known to the crypto community are based on the following theorem from [22].

Theorem 1. *The T-function defined by setting* $\mathbf{T}(\mathbf{x}) = (T_k(\mathbf{x}))_{k=0}^{m-1}$*, with*

$$T_k(\mathbf{x}) = x_k \oplus (\alpha_k(\mathbf{x}) \wedge x_0 \wedge x_1 \wedge \cdots \wedge x_{k-1}) \tag{5}$$

for $k = 0, \ldots, m-1$*, exhibits the single cycle property, when each* α_k *is an odd parameter.*

3 Analysis of an example T-function

The following example may be found in [22]. We shall try to give a clearer view of the inner workings of this example.

Example 1. Consider the following T-function which acts on four 64-bit words. Fix any odd number `C0` and set `C1` = `0x12481248`, `C3` = `0x48124812`. Use the notation $a_0 = x_0$ and $a_{i+1} = a_i \wedge x_{i+1}$ for $i = 0, 1, 2$. Then,

$$\alpha = \alpha(\mathbf{x}) = (a_3 + \mathtt{C0}) \oplus a_3 \tag{6}$$

defines an odd parameter. Finally, the mapping

$$\begin{pmatrix} x_3 \\ x_2 \\ x_1 \\ x_0 \end{pmatrix} \mapsto \begin{pmatrix} x_3 \oplus (\alpha \wedge a_2) \oplus (2x_0(x_1 \vee \mathtt{C1})) \\ x_2 \oplus (\alpha \wedge a_1) \oplus (2x_0(x_3 \vee \mathtt{C3})) \\ x_1 \oplus (\alpha \wedge a_0) \oplus (2x_2(x_3 \vee \mathtt{C3})) \\ x_0 \oplus \ \alpha \qquad\ \ \oplus (2x_2(x_1 \vee \mathtt{C1})) \end{pmatrix} \tag{7}$$

gives a single cycle T-function.

This example is not a direct application of Theorem 1, because of the last term in each row that utilizes multiplications. But these last terms are even parameters and it is possible to follow through the proof of Theorem 1 given in [22] with them attached.

Let us have a closer look at this example. Without the even parameter part, it is almost identical to Theorem 1.

$$\begin{pmatrix} x_3 \\ x_2 \\ x_1 \\ x_0 \end{pmatrix} \mapsto \begin{pmatrix} x_3 \oplus (\alpha(\mathbf{x}) \wedge x_0 \wedge x_1 \wedge x_2) \\ x_2 \oplus (\alpha(\mathbf{x}) \wedge x_0 \wedge x_1) \\ x_1 \oplus (\alpha(\mathbf{x}) \wedge x_0) \\ x_0 \oplus \ \alpha(\mathbf{x}) \end{pmatrix}. \tag{8}$$

We will denote this simplified function by $\mathbf{T}$ for the moment. Now, let us look at just the 0-th column. We know $[\alpha(\mathbf{x})]_0 = 1$ for any odd parameter and this can also be checked directly from (6). Hence, as noted in [22], the map (8) restricted to the 0-th column is

$$[\mathbf{T}(\mathbf{x})]_0 = [\mathbf{x}]_0 + 1 \pmod{2^m}. \tag{9}$$

Here, we are using the notation of (2). Let us move onto the higher bits. Since all the odd parameters are set to be the same in (8), the mapping $\mathbf{T}$ may be described by

$$[\mathbf{T}(\mathbf{x})]_i = \begin{cases} [\mathbf{x}]_i & \text{if } [\alpha(\mathbf{x})]_i = 0, \\ [\mathbf{x}]_i + 1 \pmod{2^m} & \text{if } [\alpha(\mathbf{x})]_i = 1. \end{cases} \tag{10}$$

In other words, the role of the odd parameter α is to decide whether or not to apply the map

$$[\mathbf{x}]_i \mapsto [\mathbf{x}]_i + 1 \pmod{2^m} \tag{11}$$

to the i-th bit column.

In the extreme case when `C0` $= 1$ in the definition (6) for the odd parameter, we have $[\alpha(\mathbf{x})]_i = 1$ if and only if $[x_k]_j = 1$ for all k and all $j < i$. So the mapping (11) is applied to the i-th bit column if and only if all $4i$ of the strictly lower bits are filled with 1. Hence the map (8) literally defines a counter (in hexadecimal numbers). It runs through all the $2^{4\cdot 64}$ possible values, incrementing the memory by 1 at each application of $\mathbf{T}$.

The mapping (8) is more complex when the constant `C0` is bigger, but this does not seem to give a fundamental difference. Using a more complex odd parameter would make (8) a lot more random-like. Without this, the real reason for (7) producing a sequence which passes all the statistical tests lies in the even parameter part.

Remark 1. The paper [27] gives an attack on a (very basic) streamcipher base on (7). Essentially, they analyze the multiplicative part, and find a way to apply this technique to (7). From the viewpoint of their attack, the components of (7), excluding the multiplicative part, contribute very little to the security of the system. Arguments of this section show that this is a natural consequence of its inner workings.

4 A new class of T-functions

Arguments of the previous section lead us naturally to the following idea. What would happen if we replaced mapping (11) with a more general mapping?

Given an $m \times m$ S-box, $S : \{0,1\}^m \mapsto \{0,1\}^m$, define

$$\mathbf{S} : (\{0,1\}^n)^m \longrightarrow (\{0,1\}^n)^m, \qquad \mathbf{x} \mapsto \mathbf{S}(\mathbf{x})$$

by setting

$$[\mathbf{S}(\mathbf{x})]_i = S([\mathbf{x}]_i).$$

Here, we are using the notation of (2), so that the bold face $\mathbf{S}$ acts on each and every *column* of the registers. We say that an S-box has the *single cycle* property if its cycle decomposition gives a single cycle. That is, starting from any point, if we iteratively act S, we end up going through all possible elements of $\{0,1\}^m$.

Certainly, $\mathbf{S}$ will not define a single cycle T-function, even when S is of single cycle. So let us start by first defining some logical operations on multi-words. Let $\mathbf{x} = (x_k)_{k=0}^{m-1}$ and $\mathbf{y} = (y_k)_{k=0}^{m-1}$, be two multi-words. Define $\mathbf{x} \oplus \mathbf{y}$ by setting

$$\mathbf{x} \oplus \mathbf{y} = (x_k \oplus y_k)_{k=0}^{m-1}.$$

Also, for a (single) word α, define the multi-word

$$\alpha \cdot \mathbf{x} = (\alpha \wedge x_k)_{k=0}^{m-1}.$$

The notation $\sim \alpha$ will denote the bitwise complement of α.

Theorem 2. *Let S be a single cycle S-box and let α be an odd parameter. If S^o is an odd power of S and S^e is an even power of S, the mapping*

$$\mathbf{T}(\mathbf{x}) = \big(\alpha(\mathbf{x}) \cdot \mathbf{S^o}(\mathbf{x})\big) \oplus \big((\sim \alpha(\mathbf{x})) \cdot \mathbf{S^e}(\mathbf{x})\big).$$

defines a single cycle T-function.

Proof. That this is a T-function is easy to check. Notice that due to its definition, any T-function may be restricted to just the lower bits. It suffices to prove that, when restricted to the lower bits $[\mathbf{x}]_0, [\mathbf{x}]_1, \ldots, [\mathbf{x}]_{i-1}$, the period of the above map is $2^{m \cdot i}$. This will be shown by induction.

The mapping $\mathbf{T}$ can be better understood when it is written as

$$[\mathbf{T}(\mathbf{x})]_i = \begin{cases} S^e([\mathbf{x}]_i) & \text{if } [\alpha(\mathbf{x})]_i = 0, \\ S^o([\mathbf{x}]_i) & \text{if } [\alpha(\mathbf{x})]_i = 1. \end{cases} \tag{12}$$

In particular, since we always have $[\alpha(\mathbf{x})]_0 = 1$, if we restrict $\mathbf{T}$ to the 0-th column of the registers, it acts as just S^o regardless of the input $\mathbf{x}$. Notice that any odd power of S also has the single cycle property. Hence the period of $\mathbf{T}$ is 2^m, when restricted to $[\mathbf{x}]_0$. This gives us the starting point.

So suppose, as an induction hypothesis, that the period of $\mathbf{T}$, restricted to the lower bits $[\mathbf{x}]_0, \ldots, [\mathbf{x}]_{i-1}$, is $2^{m \cdot i}$. The period of $\mathbf{T}$, restricted to the next step $[\mathbf{x}]_0, \ldots, [\mathbf{x}]_i$, must be a multiple of $2^{m \cdot i}$. Now, with the parameter α being odd, (12) shows that when $\mathbf{T}$ is consecutively applied to the bits $[\mathbf{x}]_0, \ldots, [\mathbf{x}]_i$ exactly $2^{m \cdot i}$ times, S^o and S^e are both applied an odd number of times to $[\mathbf{x}]_i$. In all, this is equivalent to applying S to $[\mathbf{x}]_i$ an odd number of times. Since an odd number is relatively prime to the period 2^m of S, the period of $\mathbf{T}$ restricted to $[\mathbf{x}]_0, \ldots, [\mathbf{x}]_i$ must be $2^{m \cdot (i+1)}$. This completes the induction step and the proof.

Expression (12) shows that this new T-function may be viewed as a twisted product of small S-boxes, each acting on a single column of memory.

The reader may already have noticed that allowing for odd powers of single cycles S-boxes is not really any more general than allowing for just the (single power of) single cycle S-boxes. If we further restrict the above theorem to the case when the even power used is zero, we have the following corollary.

Corollary 1. *Given a single cycle S-box S and an odd parameter α, the following mapping defines a single cycle T-function.*

$$\mathbf{x} \mapsto \mathbf{x} \oplus \big(\alpha(\mathbf{x}) \cdot (\mathbf{x} \oplus \mathbf{S}(\mathbf{x}))\big).$$

5 T-function based streamcipher; TSC-1

In this section, we propose a very bare framework for a streamcipher based on Theorem 2. A distinguishing attack on this example of very low complexity is already known[19] and the authors no longer believe this cipher to be secure, but we include this as a reference for further developments in this direction.

Since the work [27] has shown that disclosing parts of the raw memory state could be fatal, we want to hide the memory while producing output from this T-function. So we shall use the T-function as a substitute for an LFSR in a filter model.

5.1 The specification

Specification of the cipher will be given by supplying a filter function in addition to fixing various components for the T-function.

Fix $n = 32$ and $m = 4$, that is, we work with four 32-bit words, for a total internal state of 128 bits. Define an odd parameter by setting

$$\alpha(\mathbf{x}) = (p + \mathtt{C}) \oplus p \oplus 2s, \tag{13}$$

where[2]

$$\mathtt{C} = \mathtt{0x12488421}, \quad p = x_0 \wedge x_1 \wedge x_2 \wedge x_3, \quad \text{and} \quad s = x_0 + x_1 + x_2 + x_3.$$

All additions are done modulo 2^{32}. This is equal to (6), except that we have added the even parameter $2s$ to allow for stronger inter-column effects. Define a 4×4 S-box S, as given by the following line written in C-style.

```
S[16] = {3,5,9,13,1,6,11,15,4,0,8,14,10,7,2,12};
```
(14)

One may check easily that this is a single cycle S-box. Using this S-box, let us set $S^o = S$ and $S^e = S^2$. We can now define a single cycle T-function $\mathbf{T}$, through the use of Theorem 2.

To actually obtain the keystream, use the filter

$$f(\mathbf{x}) = (x_{0\lll 9} + x_1)_{\lll 15} + (x_{2\lll 7} + x_3) \tag{15}$$

on the memory bits, after each application of $\mathbf{T}$. This will give us a single 32-bit block of keystream per action of the T-function. Here, the symbol $\lll$ denotes left rotation, and the additions should be done modulo 2^{32}. Going back to the notation of (3), the output word produced at time t may be written as $f(\mathbf{x}^t)$.

[2] The constant `0x12488421` was chosen so that 1s are denser at the higher bits than at the lower bits. This will help it quickly move away from the all-zero state, should it occur.

5.2 Naive security

Period We already know that the period of the state registers is 2^{128}, as guaranteed by the single cycle property. The output itself also has a period of 2^{128} words.

To see this, first note that the period has to be a divisor of 2^{128}. Now, initialize the register content with the all zero state and consider what the content of the registers would be after 2^{124} iterated applications of the T-function. Since the period of the T-function restricted to the lower 31 columns is 2^{124}, all columns except the most significant column will be zero. Furthermore, when observed every 2^{124} iterations apart, due to description (12) and the definition of an odd parameter, the change of the most significant column follows some fixed odd power of the S-box, which is of cycle length 16. Explicit calculation of the 16 keystream output words for each odd power of the S-box confirms that, in all odd power cases, one has to go through all 16 points before reaching the starting point. Hence the period of the cipher is $16 \cdot 2^{124} = 2^{128}$.

Actually, for a general single cycle T-function, one can always show that at least one bit position in the register will show period equal to the T-function. When any mildly complicated filter is attached to such a T-function, the output keystream has a high chance of inheriting this property and one should be able to show some result on the period of the whole filter generator. For example, the cipher TF-1[23], can be shown to have a period of at least 2^{254}.

Maximum security level Given a single word block of keystream, guessing any three words (96 bits) of memory determines the remaining word uniquely. And it suffices to look at the next three word blocks of keystream to check if the guess is correct. Hence, it is clear that this proposal cannot provide more than 96-bit security.

Bit-flip property In addition to imposing the single cycle property, we had chosen the S-box (14) to satisfy the following conditions.[3]

1. At the application of S, each of the four bits has bit-flip probability of $\frac{1}{2}$.
2. The same if true for S^2, the square of S.

In more exact terms, the first condition states that for each $i = 0, 1, 2, 3$,

$$\#\{\, 0 \leq t < 16 \mid \text{the } i\text{-th bit of } t \oplus S(t) \text{ is } 1\} = 8.$$

Due to this property, regardless of the behavior of the odd parameter α, every bit in the register is guaranteed a $\frac{1}{2}$ bit-flip probability. This is one thing that wasn't satisfied by (8) and which was only naively expected of (7).

[3] S-box (14) enjoys the added property that S^6, S^{10}, S^{14}, and all odd powers of S also exhibit the $\frac{1}{2}$ bit-flip probability.

Rotations Rotations used in the filter serve two main purposes. The first is to ensure that output from the same S-box, i.e., bits from the same column, do not contribute directly to the same output bit. We want contributions to any single output bit to come from bits that change independently of each other.

The other reason is to remove the possibility of relating a part of the output with a part of the memory that allows some sort of separate handling in view of the action of T-function. In particular, this stops the guess-then-determine attack. Difficulty of correlation attacks can also be understood from this viewpoint. In the last step of a correlation attack, one needs to guess a part of the state bits and compare calculated outputs with the keystream, checking for the occurrence of expected correlation. In our case, any correlation with a single output bit will involve multiple input bits and at least one of them will come near the high ends of the registers. This will force one to guess quite a large part of the registers to be able to apply T-function even once.

Misc We have done most of the tests presented in [2] and have verified that this proposal gives good statistical results. As the S-boxes are nonlinear, the most dangerous attack on streamciphers, the algebraic attack, seems to be out of the question.

With our T-function based filter model, one can view the randomness of keystream as originating from the T-function and as being amplified through the filter. Compared with LFSR based filter models, it seems fair to say that the randomness at the source is better with T-functions. In the Appendix, we present another design that shifts the burden of producing randomness more to the filter.

5.3 Implementation

Let us first consider the cipher's efficiency in hardware. Given a single 4-bit input $t = t_0 + 2t_1 + 4t_2 + 8t_3$, the output $u = u_0 + 2u_1 + 4u_2 + 8u_3$ of $S(t)$ for the S-box (14) may be written as follows. Each line represents a single output bit as a function of four input bits.

$$\begin{aligned} u_3 &= t_1 \oplus (t_3 \wedge t_2 \wedge \bar{t}_0) \\ u_2 &= t_0 \oplus (t_3 \wedge \bar{t}_2 \wedge \bar{t}_1) \\ u_1 &= t_2 \oplus (t_3 \wedge t_1 \wedge t_0) \oplus (\bar{t}_3 \wedge \bar{t}_1 \wedge \bar{t}_0) \\ u_0 &= \bar{t}_3 \oplus (t_2 \wedge \bar{t}_1 \wedge t_0) \end{aligned} \tag{16}$$

Here, the bar denotes bit complement. Similarly, the bits of $S^2(t) = v_0 + 2v_1 + 4v_2 + 8v_3$ may be calculated as follows.

$$\begin{aligned} v_3 &= t_2 \oplus (\bar{t}_3 \wedge \bar{t}_1 \wedge \bar{t}_0) \\ v_2 &= \bar{t}_3 \oplus (t_2 \wedge \bar{t}_1 \wedge t_0) \oplus (\bar{t}_2 \wedge t_1 \wedge \bar{t}_0) \\ v_1 &= t_0 \oplus (\bar{t}_3 \wedge t_2 \wedge t_1) \\ v_0 &= \bar{t}_1 \oplus (t_3 \wedge t_2 \wedge \bar{t}_0) \oplus (\bar{t}_3 \wedge \bar{t}_2 \wedge t_0) \end{aligned} \tag{17}$$

We had deliberately chosen the S-box so that these expressions are simple.[4]

For sake of simplicity, let us assume that the logical operations NOT, AND, and XOR all take the same time in a hardware implementation. Even for a very straightforward implementation of (16) and (17), the critical path for the simultaneous calculation of $S(t)$ and $S^2(t)$ contains only 4 logical operations.

In most hardware implementations, this takes a lot shorter than the time required for a single 32-bit addition, an algebraic operation. Hence the calculation of α, given by (13), whose critical path consists of two 32-bit additions and a single XOR, will take longer than the S-box calculation. In all, the total time cost of the T-function given by Theorem 2 is two 32-bit additions and four logical bit operations.

The filter (15), taking two 32-bit additions, may be run in parallel to the T-function, so our cipher will produce 32-bits of keystream for every clock tick that allows for two 32-bit additions and four logical bit operations.

Thus a straightforward approach will give us a very fast hardware implementation. For example, in an ASIC implementation that uses a 32-bit adder of modest delay time 0.4ns, the cipher will run at somewhere around 32 Gbps. The total cost for such a rough implementation is given in Table 1.

register	128×(flip-flop)
S-box	32×(11 XOR, 22 AND, 20 NOT)
odd parameter	4×(32-bit addition), 32×(2 XOR, 3 AND)
filter	3×(32-bit addition)
the rest	32×(1 XOR, 8 AND, 1 NOT)

Table 1. Implementation const of TSC-1

For lack of a good hardware oriented streamcipher, let us try to compare this implementation cost with that of a summation generator on four 256-bit LFSRs. Results of [24] show that this may be broken within time complexity of $2^{77} \sim \binom{4\cdot 256}{3}^{\log_2 7}$. Even this weak summation generator needs 1024 flip-flops, just to get started on the four LFSRs. This will already be larger than what we have in Table 1.

Actually, some tricks may be used to reduce the gates needed for S-box implementation without impacting speed. For example, if we use the expression

$$\begin{aligned} y &:= (\bar{t}_1 \wedge (t_2 \wedge t_0)) \oplus t_3 \\ z &:= \ \bar{t}_1 \vee (t_2 \vee t_0) \\ u_0 &= \bar{y} \\ v_2 &= y \oplus z, \end{aligned}$$

[4] We have rejected S-boxes that contained wholly linear expressions. But even such S-boxes might be used when this cipher is better understood.

calculation of u_0 and v_2 may be done in 2 XOR, 2 AND, 2 OR, and 2 NOT, whereas, it was carelessly counted as 3 XOR, 6 AND, and 6 NOT in Table 1.

A very small but slower implementation might use just one or two 4×4 S-boxes, and implementations that come somewhere in between are also possible, allowing for a very wide range of implementation choices.

Although we have designed this cipher mainly for hardware, its performance in software is not bad. Using the standard bit-slice technique for S-boxes with the above polynomial expressions (16) and (17), we achieve speeds of up to 1.25 Gbps on a Pentium IV 2.4GHz, Windows XP (SP1) platform using Visual C++ 6.0(SP6). In comparison, the Crypto++ Library[1] cites a speed of 113 MBps for RC4 on a Pentium IV 2.1GHz machine. Scaled up for 2.4GHz, this is only 1.03 Gbps.

6 Conclusion

We have given an analysis of the generic single cycle T-function previously known. With a better understanding of this T-function, we were able to present a new class of single cycle T-functions.

Compared to the old T-function (Theorem 1), our T-function (Theorem 2) certainly gives a better *column mixing.* Also, unlike (8), which is based on the old T-function, the bit-flip probability of the register bits under the action of our new T-function construction can be manipulated, and even made equal to $\frac{1}{2}$, through proper selection of S and α.

On the other hand, unlike previous T-functions, our new T-function does not allow for the addition of an even parameter. This, we admit, is a very disappointing characteristic. But we would like to take the position that the multiplicative even parameter is the less understood part of previous T-function(Example 1), while being at the very core of its randomness. And as we saw in our example ciphers, the reduced randomness of the register contents can be compensated for by an appropriate use of the filter function.

We have also presented an example cipher which shows the possibility of using T-functions to build hardware oriented streamciphers. Our T-function allows for a wide range of implementation choices, so that the final cipher could be either fast or of small footprint in hardware.

Acknowledgments

We would like to thank Alexander Klimov and Adi Shamir for their valuable comments on an earlier version of this paper.

References

1. Crypto++ 5.2.1 benchmarks. Available from
`http://www.eskimo.com/~weidai/benchmarks.html`

2. NIST. A statistical test suite for random and psedorandom number generators for cryptographic applications. NIST Special Publication 800-22.
3. V. S. Anashin, Uniformly distributed sequences over p-adic integers. Proceedings of the Intl Conference on Number Theoretic and Algebraic Methods in Computer Science (A. J. van der Poorten, I. Shparlinsky and H. G. Zimmer, eds.), World Scientific, 1995.
4. V. S. Anashin, private communication.
5. F. Armknecht, M. Krause, Algebraic attacks on combiners with memory, *Advances in Cryptology - Crypto 2003*, LNCS 2729, Springer-Verlag, pp.162–175, 2003.
6. M. Boesgaard, M. Besterager, T. Pedersen, J. Christiansen, and O. Scavenius, Rabbit: A new high-performance stream cipher. *FSE 2003*, LNCS 2887, pp.307–329, 2003.
7. S. Babbage, Stream ciphers: What does the industry want? Presented at *State of the Art of Stream Ciphers* workshop, Brugge, 2004.
8. A. Biryukov, A. Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. *Asiacrypt 2000,* LNCS 1976, pp. 1–13, Springer-Verlag, 2000.
9. K. Chen, M. Henricksen, W. Millan, J. Fuller, L. Simpson, E. Dawson, H. Lee, and S. Moon, Dragon: A fast word based stream cipher. To be presented at *ICISC 2004*.
10. A. Clark, E. Dawson, J. Fuller, J. Golić, H-J. Lee, W. Miller, S-J. Moon, and L. Simpson, The LILI-II Keystream Generator. *ACISP 2002*.
11. N. Courtois, Algebraic attacks on combiners with memory and several outputs, E-print archive, 2003/125. To be presented at ICISC 2004.
12. N. Courtois, Higher order correlation attacks, XL algorithm and Cryptanalysis of Toyocrypt, *ICISC 2002*, LNCS 2587, Springer-Verlag, pp.182–199, 2002.
13. N. Courtois, W. Meier, Algebraic attacks on stream ciphers with linear feedback, *Advances in Cryptology - Eurocrypt 2003*, LNCS 2656, Springer-Verlag, pp.345–359, 2003.
14. P. Ekdahl and T. Johansson, A new version of the stream cipher SNOW, *SAC 2002*, pp.47–61, 2002.
15. N. Ferguson, D. Whiting, B. Schneier, J. Kelsey, S. Lucks, and T. Kohno, Helix, fast encryption and authentication in a single cryptographic primitive, *FSE 2003*, 2003.
16. P. Hawkes and G. Rose, Primitive specification and supporting documentation for SOBER-t32, *NESSIE Submission*, 2000.
17. S. Halevi, D. Coppersmith, and C. Jutla, Scream: A software-efficient stream cipher, *FSE2002*, volume 2365 of LNCS, pages 195–209, Springer-Verlag, 2002.
18. J. Hong, D. H. Lee, Y. Yeom, and D. Han, A new class of single cycle T-functions and a stream cipher proposal. *SASC*(State of the Art of Stream Ciphers, Brugge, Belgium, Oct. 2004) workshop record. Available from `http://www.isg.rhul.ac.uk/research/projects/ecrypt/stvl/sasc.html`
19. P. Junod, S. Kuenzlie, and W. Meier, Attacks on TSC. FSE 2005 rump session presentation.
20. A. Klimov and A. Shamir, A new class of invertible mappings. *CHES 2002*, LNCS 2523, Springer-Verlag, pp.470–483, 2003.
21. A. Klimov and A. Shamir, Cryptographic application of T-functions. *SAC 2003*, LNCS 3006, Springer-Verlag, pp.248–261, 2004.
22. A. Klimov and A. Shamir, New cryptographic primitives based on multiword T-functions. *FSE 2004*, LNCS 3017, Springer-Verlag, pp.1–15, 2004.
23. A. Klimov and A. Shamir, The TFi family of stream ciphers. Handout at the *State of the Art of Stream Ciphers* workshop, Brugge, 2004.

24. D. H. Lee, J. Kim, J. Hong, J. W. Han, and D. Moon, Algebraic attacks on summation generators. *FSE 2004*, LNCS 3017, Springer-Verlag, pp.34–48, 2004
25. M. Matsui, Linear cryptanalysis method for DES cipher. *Eurocrypt '93*, LNCS 765, Springer-Verlag, pp.386–397, 1994.
26. D. McGrew and S. Fluhrer, The stream cipher LEVIATHAN. *NESSIE Submission*, 2000.
27. J. Mitra and P. Sarkar, Time-memory trade-off attacks on multiplications and T-functions. *Asiacrypt 2004*, LNCS 3329, Springer-Verlag, pp.468–482, 2004.
28. A. Shamir, Stream ciphers: Dead or alive? Invited talk presented at *State of the Art of Stream Ciphers* workshop, Brugge, 2004 and *Asiacrypt 2004.*
29. K. Sugimoto, T. Chikaraishi, and T. Morizumi, Design criteria and security evaluations on certain stream ciphers. *IEICE Technical Report*, ISEC20000-69, Sept. 2000.
30. D. Watanabe, S. Furuya, H. Yoshida, K. Takaragi, and B. Preneel, A new key stream generator MUGI, *FSE 2002*, pp.179–194, 2002.
31. H. Wu, A new stream cipher HC-256, *FSE 2004*, LNCS 3017, Springer-Verlag, pp.226–244, 2004.

A T-function based streamcipher; TSC-2

A streamcipher based on Corollary 1 will be given in this section.[5] At the time of this writing, this example cipher is known to be insecure[19]. It is included in this paper for reference purposes only.

Compared to TSC-1, this cipher example will be lighter in hardware at slightly reduced speed and faster in software. As with TSC-1, we will provide a filter function in addition to fixing various components for the T-function.

A.1 The specification

Fix $n = 32$ and $m = 4$. Define an odd parameter by setting

$$\alpha(\mathbf{x}) = (p+1) \oplus p \oplus 2s, \tag{18}$$

where

$$p = x_0 \wedge x_1 \wedge x_2 \wedge x_3 \quad \text{and} \quad s = x_0 + x_1 + x_2 + x_3.$$

Define a 4×4 S-box S as follows.

$$\texttt{S[16] = \{5,2,11,12,13,4,3,14,15,8,1,6,7,10,9,0\};} \tag{19}$$

One may check easily that this is a single cycle S-box. We can now define a single cycle T-function $\mathbf{T}$, through the use of Corollary 1.

To actually obtain the keystream, use the filter

$$f(\mathbf{x}) = (x_{0\lll 11} + x_1)_{\lll 14} + (x_{0\lll 13} + x_2)_{\lll 22} + (x_{0\lll 12} + x_3) \tag{20}$$

on the memory bits, after each application of $\mathbf{T}$. This will give us a single 32-bit block of keystream per action of the T-function.

[5] This example cipher was presented at the SASC workshop. Readers may find more detail, including an example C-code, in [18].

A.2 Naive security

Most of the arguments of Section 5.2 carry over to TSC-2, word for word. But arguments concerning the bit-flip probability needs to be redone.

With a T-function following the construction of Corollary 1, it is difficult to obtain a $\frac{1}{2}$ bit-flip probability for all the memory bits. If this non-randomly characteristic were to show unfiltered in the final keystream, we could obtain a distinguishing attack. So we took care to make sure one word of memory displayed the $\frac{1}{2}$ bit-flip probability and used it to ensure that the final keystream showed the same characteristic. Let us explain this in more detail.

The S-box (19), in addition to meeting the single cycle property, satisfies the following.

– An even number is sent to an odd number and vice versa.

In other words, the LSB (among the four bits in a single column of memory) is flipped on every application of S. To see the bit-flip probability of $\mathbf{T}$ itself, we should next look at how often S is applied to each column.

Lemma 1. *The odd parameter* (18) *satisfies*

$$\mid \frac{1}{2} - \mathrm{prob}_{\mathbf{x}}([\alpha(\mathbf{x})]_i = 1) \mid = \frac{1}{2^{4i}}$$

for all $i > 0$.

This lemma, which may be proved directly, tells us that except for in the lower few bits, each output bit of α is equal to 1 almost half of the time. Recalling description (12) of $\mathbf{T}$ together with the bit-flip characteristic of S, we conclude that bits of memory x_0 has bit-flip probability close to $\frac{1}{2}$ except at the lower few bits.

Now, the 32-bit addition operation, seen at each bit position, is an XOR of two input bits and a carry bit, so we may apply the Piling-up Lemma[25] to argue that for each $i = 1, 2, 3$, the bits of

$$x_{0 \lll k} + x_i$$

will have bit-flip probability very close to $\frac{1}{2}$, except maybe at the points where lower bits of x_0 was used. What discrepancy these bits may show from changing one half of the time disappears, once again through the use of Piling-up Lemma, when these values are rotated relative to each other and added together to form the final output (20). The rotation, while allowing the mixing of lower bits of x_0 with higher bits, also gains independence of the XORed bits needed in applying the Piling-up Lemma.

Using the explicit probability stated by Lemma 1, we have checked that a straightforward distinguishing attack based on the bit-flip probability of (single $\mathbf{T}$ action on) register contents is not possible.

A.3 Implementation

As in TSC-1, the S-box (19) was chosen with its efficient implementation in mind. The mapping $t \mapsto t \oplus S(t)$ allows for an efficient bit slice realization. Also note that because of the even-odd exchange condition, the LSB of $t \oplus S(t)$ will always be 1, leaving only 3 bits to be calculated.

Hardware implementation of TSC-2 will be slightly slower compared to that of TSC-1, because the output filter now exhibits the critical path of three 32-bit additions. But we have halved the count of S-boxes to obtain a lower total implementation cost.

In software, TSC-2 runs on a Pentium-IV 2.4 GHz machine with code compiled using Visual C++ 6.0 (SP6) at speeds over 1.6 Gbps. This is over 1.6 times faster than the speed for RC4 given in [1].