The TFi Family of Stream Ciphers

Alexander Klimov and Adi Shamir

Computer Science department,
The Weizmann Institute of Science
Rehovot 76100, Israel

{ask,shamir}@weizmann.ac.il

October 2004

Notations

- $\mathbb{B}$ denotes $\{0,1\}$.
- $\mathbb{B}^n$ denotes n-bit words (or integers in $\{0,\ldots,2^n-1\}$).
- $[x]_i$ denotes bit number i of x (where $[x]_0$ is the least significant bit of x).
- All arithmetic operations (addition $(+)$, subtraction $(-)$ and multiplication $(\times)$) are implicitly reduced modulo 2^n. The standard word size is $n = 64$.
- Bitwise operations on n-bit words are denoted by $\vee$ (or), $\wedge$ (and) and $\oplus$ (xor).
- Circular bit-shifts of x by i positions are denoted by $x \circlearrowleft i$ (left) and $x \circlearrowright i$ (right).

TF-0

This is the simplest non-linear T-function with a single cycle of maximal length. It is an excellent substitute for LFSRs and linear congruential generators in non-cryptographic `rand()` functions.

State: $x \in \mathbb{B}^{64}$,
Update: $x \to x + (x^2 \vee C)$,
Output: $[x]_{63\ldots64-k}$,

where C is a constant such that $C < 2^{32}$ and $C \bmod 8 = 5{,}7$ (e.g., $C = 5$), $k = 1, 8, 16, 32$.

The update function is a nonlinear mapping which can be evaluated using only three operations. There are only two such families of mappings [3]: $x \to x \pm (x^2 \vee C)$, where $C \bmod 8 = 5{,}7$, and $x \to x \pm (x \vee C)^2$, where C is odd. Intuitively, the former provides better mixing (each bit of the state is used in multiplication), whereas the latter is more resistant to "zero-tail" attacks (if x ends with p zeros then x^2 ends with $2p$ zeros). If C is small then for each fixed $x \bmod 2^{|C|}$ the mapping is equivalent to some polynomial. If $|C| > 64 - k$ then there will be a bit "visible through the output window" which is flipped with biased probability. The generator is not secure by itself since it is vulnerable to a brute force attack ($O(2^{64-k})$), to a "zero-tail" attack ($O(2^{\frac{64-k}{2}})$) and even faster more sophisticated attacks.

TF-1

This generator is better suited for cryptographic applications (with intended security 2^{128}) and has a guaranteed period of 2^{256}.

State: $x_0, x_1, x_2, x_3 \in \mathbb{B}^{64}$,

Update:
$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \to \begin{pmatrix} x_0 \oplus s \oplus (2(x_1 \vee C_1)x_2) \\ x_1 \oplus (s \wedge a_0) \oplus (2x_2(x_3 \vee C_3)) \\ x_2 \oplus (s \wedge a_1) \oplus (2(x_3 \vee C_3)x_0) \\ x_3 \oplus (s \wedge a_2) \oplus (2x_0(x_1 \vee C_1)) \end{pmatrix},$$

Output: $\big((x_0 + x_2) \circlearrowleft 32\big) \times \big(((x_1 + x_3) \circlearrowleft 32) \vee 1\big)$,

The update function was constructed in the following way: We start from a single-cycle mapping with the smallest (as far as we know) number of primitive operations [4]:

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \rightarrow \begin{pmatrix} x_0 & \oplus & s \\ x_1 & \oplus & (s \wedge a_0) \\ x_2 & \oplus & (s \wedge a_1) \\ x_3 & \oplus & (s \wedge a_2) \end{pmatrix},$$

where $a_0 = x_0$, $a_1 = a_0 \wedge x_1$, $a_2 = a_1 \wedge x_2$, $a_3 = a_2 \wedge x_3$, $s = (a_3 + C) \oplus a_3$ and C is an odd constant. The particular value of the constant was chosen to maximize the number of ones in s (note that a_3 is sparse since each its bit is one with probability $\frac{1}{16}$). Even parameters were added to make each variable non-linearly depend on the values of two other variables:

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \rightarrow \begin{pmatrix} x_0 & \oplus & s & \oplus & (2x_1x_2) \\ x_1 & \oplus & (s \wedge a_0) & \oplus & (2x_2x_3) \\ x_2 & \oplus & (s \wedge a_1) & \oplus & (2x_3x_0) \\ x_3 & \oplus & (s \wedge a_2) & \oplus & (2x_0x_1) \end{pmatrix}.$$

Finally, "$\vee$" was added to avoid "zero-tail" attacks with the smallest number of additional operations. Note that since the constants are odd each bit of the product is one with probability $\frac{1}{2}$, and so each bit (except the least significant one) of each variable is flipped with probability $\frac{1}{2}$.

We want the output function to be non-linear and to depend on the whole state: $(x_0 + x_2)(x_1 + x_3)$ (we do not use $(x_0 + x_1)(x_2 + x_3)$, because we want each factor to non-linearly depend on the whole previous state). To make the distribution of each bit of the product uniform we or one of the factors with an odd constant: $(x_0 + x_2)((x_1 + x_3) \vee 1)$. Since the lower bits of the state are weak we rotate the factors: $\big((x_0 + x_2) \circlearrowleft 32\big) \times \big(((x_1 + x_3) \circlearrowleft 32) \vee 1\big)$.

We tested the actual execution speed of our mappings on an IA-32 machine. We used a standard PC with a 2.66 GHz Pentium 4 processor with 512KB of cache. In each experiment we encrypted one gigabyte of data by encrypting 10^4 times a buffer which is 10^5 bytes long (this ensured that the data was prefetched into L2 cache). The tests were done on an otherwise idle Linux system. To take advantage of the SSE2 instruction set of Pentium 4 processors (which contains 128-bit integer instructions that allow us to operate on two 64-bit integers simultaneously[1]), we run two generators in parallel. With this optimization, the complete encryption operation (including the state update function, the generation of the output, the xor with the data, the read/write of the data, and the loop instructions) required only 2.19 seconds per gigabyte, and thus the experimentally verified encryption speed was approximately 3.65 gigabits per second. To put our results in perspective, the reader should consider the RC4 stream cipher, which is one of the fastest software oriented ciphers available today. The web site of Crypto++ contains benchmarks for highly optimized implementations of many well known cryptographic algorithms [1]. For RC4 it quotes a speed of 113 megabyte per second on a 2.1 GHz Pentium 4 processor. This can be scaled to $\frac{1000}{113} \times \frac{2.1}{2.66} = 7.2$ seconds per gigabyte on our 2.66GHz processor. Note that, although the performance of our scheme is quite impressive on a 32-bit processor, it will be much higher on a 64-bit one.

The best attack we are aware of has complexity $O(2^{128})$: Guess the least significant half of all variables (128 bits in total). For each possible value of an additional bit-slice check if it conforms to the output for few iterations.

References

[1] Crypto++ 5.1 Benchmarks: `http://www.eskimo.com/~weidai/benchmarks.html`

[2] A. Klimov and A. Shamir, "A New Class of Invertible Mappings", CHES 2002.

[3] A. Klimov and A. Shamir, "Cryptographic Applications of T-functions", SAC 2003.

[4] A. Klimov and A. Shamir, "New Cryptographic Primitives Based on Multiword T-functions", FSE 2004.

[1] Although the SSE2 instruction set contains instructions for bitwise and arithmetic operations with 64-bit integers, there is no instruction to multiply two such integers and so each such multiplication have to be emulated with several shifts, additions and 32-bit multiplications.