High-Speed Software Implementation of the Prospective 128-bit Block Cipher and Streebog Hash-Function

Mikhail Borodin, Andrey Rybkin, Alexey Urivskiy

Abstract

We describe a method for high-speed software implementation of the LSX-type block cipher with block length of 128 bits and Streebog (GOST R 34.11-2012) hash-function. The method is based on combining a linear transformation L and a non-linear substitution S into a single operation, and implementing it via precomputed look-up tables. Implementation of the inverse of the iterative LSX-transformation, in which L-transformation precedes S-substitution, is also considered.

We implemented our method in the C++ programming language. Without any assembler code, special registers or SIMD instructions, our implementation encrypts at 26 and decrypts at 31 clock cycles per byte, and Streebog-512 hash-function hashes at 35 clock cycles per byte on a x64/Intel platform.

Keywords: iterative LSX-transformation, block cipher, Streebog, high-speed implementation.

1 Cipher description

A prospective block cipher presented at RusCrypto'2013 [1] (see also [2]) is a substitution-permutation network, in which three invertible transformations are applied iteratively to a 128-bit argument:

- $X[K](a) = K \oplus a$, $K, a \in V_{128}$, — a bitwise XOR with a round key K;
- $S(a) = S(a_{15}\| \dots \|a_0) = \pi(a_{15})\| \dots \|\pi(a_0)$, $a_j \in V_8$, — a bijective mapping based on a nonlinear byte substitution π;
- $L\colon V_{128} \to V_{128}$ — some linear transformation.

Here V_n is a set of all binary strings of length n. For our purposes we consider L to be a matrix multiplication

$$L(a) = (a_{15}, \ldots, a_0)C \tag{1}$$

for some 16×16 nonsingular matrix C over $GF(2^8)$ and $a_i \in V_8$.

Encryption $E_{K_1,\ldots,K_{10}}$ and decryption $D_{K_1,\ldots,K_{10}}$ procedures are $V_{128} \to V_{128}$ transformations defined as

$$\begin{aligned} E_{K_1,\ldots,K_{10}}(a) &= X[K_{10}]LSX[K_9]\ldots LSX[K_2]LSX[K_1](a); \\ D_{K_1,\ldots,K_{10}}(a) &= X[K_1]S^{-1}L^{-1}X[K_2]\ldots S^{-1}L^{-1}X[K_9]S^{-1}L^{-1}X[K_{10}](a) \end{aligned} \tag{2}$$

for round keys $K_1, \ldots, K_{10}$. In this paper, we do not consider any particular key scheduling to obtain round keys from a working key. We simply assume that $K_1, \ldots, K_{10}$ are available in computations.

2 Implementation

Fastest software implementations are usually obtained by representing the most resource consuming operations through look-up tables provided those tables require a reasonable amount of memory. The most resource intensive operation in the prospective cipher is the linear transformation L. In what follows we show how to make a tabular representation for L, LS and $S^{-1}L^{-1}$ transformations.

2.1 Tabular representation of linear transformation L

For $a = a_{15}\| \ldots \|a_0 \in V_{128}$, $a_j \in V_8$, $j = 0, \ldots, 15$, consider the linear transformation L:

$$L(a) = aC = (a_{15}, a_{14}, \ldots, a_0) \cdot \begin{pmatrix} c_{15,15} & c_{15,14} & \ldots & c_{15,0} \\ c_{14,15} & c_{14,14} & \ldots & c_{14,0} \\ \vdots & \vdots & \ddots & \vdots \\ c_{0,15} & c_{0,14} & \ldots & c_{0,0} \end{pmatrix} \tag{3}$$

where $c_{i,j} \in GF(2^8)$, $i = 0, \ldots, 15$.

Now we associate with each row of matrix C in (3) some linear mapping, i.e. we define a mapping $L_i\colon V_8 \to V_{128}$ such that

$$\forall b \in V_8 : \ L_i(b) = c_{i,15} \cdot b \parallel c_{i,14} \cdot b \parallel \ldots \parallel c_{i,0} \cdot b. \tag{4}$$

Then from (3) and (4) it follows, that

$$\begin{gathered} L(a) = L_{15}(a_{15}) \oplus L_{14}(a_{14}) \oplus \ldots \oplus L_0(a_0) = \\ = \\ c_{15,15} \cdot a_{15} \ \parallel \ c_{15,14} \cdot a_{15} \ \parallel \ \ldots \ \parallel \ c_{15,0} \cdot a_{15} \\ \oplus \\ c_{14,15} \cdot a_{14} \ \parallel \ c_{14,14} \cdot a_{14} \ \parallel \ \ldots \ \parallel \ c_{14,0} \cdot a_{14} \\ \oplus \\ \ldots \\ \oplus \\ c_{0,15} \cdot a_0 \ \parallel \ c_{0,14} \cdot a_0 \ \parallel \ \ldots \ \parallel \ c_{0,0} \cdot a_0 \end{gathered}$$

Thus, the transformation L could be implemented through mappings $L_0, \ldots, L_{15}$. For every i we create a table of 256 sorted rows each of 128 bits:

$c_{i,15} \cdot (0\text{x}00)$	$\parallel$	$c_{i,14} \cdot (0\text{x}00)$	$\parallel \ \ldots \ \parallel$	$c_{i,0} \cdot (0\text{x}00)$
		$\ldots$		
$c_{i,15} \cdot (b)$	$\parallel$	$c_{i,14} \cdot (b)$	$\parallel \ \ldots \ \parallel$	$c_{i,0} \cdot (b)$
		$\ldots$		
$c_{i,15} \cdot (0\text{x}FF)$	$\parallel$	$c_{i,14} \cdot (0\text{x}FF)$	$\parallel \ \ldots \ \parallel$	$c_{i,0} \cdot (0\text{x}FF)$

The b-th row (for $b \in V_8$) of the i-th table represents exactly

$$L_i(b) = c_{i,15} \cdot (b) \| c_{i,14} \cdot (b) \| \ldots \| c_{i,0} \cdot (b).$$

The inverse transformation L^{-1} can be implemented in the same way using the elements of the inverse matrix C^{-1} for computing the tables.

2.2 Composition of S-substitution and L-transformation

Further consider mappings $L_i^S\colon V_8 \to V_{128}$, $i = 0, \ldots, 15$, defined as

$$\forall b \in V_8 : \ L_i^S(b) = c_{i,15} \cdot \pi(b) \| c_{i,14} \cdot \pi(b) \| \ldots \| c_{i,0} \cdot \pi(b). \tag{5}$$

Then from (5) it follows that

$$LS(a) = L_{15}^S(a_{15}) \oplus L_{14}^S(a_{14}) \oplus \ldots \oplus L_0^S(a_0) =$$
$$=$$
$$\begin{array}{ccccccc} c_{15,15} \cdot \pi(a_{15}) & \| & c_{15,14} \cdot \pi(a_{15}) & \| & \ldots & \| & c_{15,0} \cdot \pi(a_{15}) \\ & & \oplus & & & & \\ c_{14,15} \cdot \pi(a_{14}) & \| & c_{14,14} \cdot \pi(a_{14}) & \| & \ldots & \| & c_{14,0} \cdot \pi(a_{14}) \\ & & \oplus & & & & \\ & & \ldots & & & & \\ & & \oplus & & & & \\ c_{0,15} \cdot \pi(a_0) & \| & c_{0,14} \cdot \pi(a_0) & \| & \ldots & \| & c_{0,0} \cdot \pi(a_0) \end{array}$$

Thus, to implement the composition of nonlinear and linear transformations LS it is sufficient to implement mappings $L_0^S, \ldots, L_{15}^S$. Similarly to the previous section, for every i we create a table of 256 sorted rows each of 128 bits:

$c_{i,15} \cdot \pi(0\text{x}00)$	$\|$	$c_{i,14} \cdot \pi(0\text{x}00)$	$\|$	$\ldots$	$\|$	$c_{i,0} \cdot \pi(0\text{x}00)$
		$\ldots$				
$c_{i,15} \cdot \pi(b)$	$\|$	$c_{i,14} \cdot \pi(b)$	$\|$	$\ldots$	$\|$	$c_{i,0} \cdot \pi(b)$
		$\ldots$				
$c_{i,15} \cdot \pi(0\text{x}FF)$	$\|$	$c_{i,14} \cdot \pi(0\text{x}FF)$	$\|$	$\ldots$	$\|$	$c_{i,0} \cdot \pi(0\text{x}FF)$

The b-th row (for $b \in V_8$) of the i-th table represents exactly

$$L_i^S(b) = c_{i,15} \cdot \pi(b) \| c_{i,14} \cdot \pi(b) \| \ldots \| c_{i,0} \cdot \pi(b).$$

So we joined both S and L into a single operation. Note that the elements of all the tables do not depend on the data to be transformed, and therefore can be calculated in advance.

2.3 Implementation of $S^{-1}L^{-1}$

The inverse transformation $S^{-1}L^{-1}$ required for the decryption evidently cannot be implemented only via compact look-up tables. This happens because L^{-1} precedes S^{-1}, hence either we need to store huge look-up tables, or we need to apply S^{-1} after fetching data from the tables corresponding to L^{-1}. Both approaches are inefficient.

However, we may represent the decryption procedure so that only $L^{-1}S^{-1}$ transformation will be required. Indeed, from (2) we obtain

$$\begin{aligned} D(a) &= X[K_1]S^{-1}L^{-1}X[K_2]\dots S^{-1}L^{-1}X[K_9]S^{-1}L^{-1}X[K_{10}](a) \\ &= X[K_1]S^{-1}L^{-1}X[K_2]\dots S^{-1}L^{-1}X[K_9]S^{-1}L^{-1}X[K_{10}]S^{-1}S(a) \end{aligned} \tag{6}$$

Since L^{-1} is a linear transformation, then for any a it holds that

$$\begin{aligned} L^{-1}X[K_i]S^{-1}(a) &= L^{-1}X[K_i](S^{-1}(a)) = L^{-1}(S^{-1}(a) \oplus K_i) = \\ &= L^{-1}(S^{-1}(a)) \oplus L^{-1}(K_i) = L^{-1}S^{-1}(a) \oplus L^{-1}(K_i) = \\ &= X[L^{-1}(K_i)]L^{-1}S^{-1}(a). \end{aligned} \tag{7}$$

From (6) and (7) it follows that

$$\begin{aligned} D(a) &= X[K_1]S^{-1}L^{-1}X[K_2]S^{-1}\dots L^{-1}X[K_9]S^{-1}L^{-1}X[K_{10}]S^{-1}S(a) = \\ &= X[K_1]S^{-1}\big(L^{-1}X[K_2]S^{-1}\big)\dots\big(L^{-1}X[K_{10}]S^{-1}\big)S(a) = \\ &= X[K_1]S^{-1}\big(X[L^{-1}(K_2)]L^{-1}S^{-1}\big)\dots\big(X[L^{-1}(K_{10})]L^{-1}S^{-1}\big)S(a) \end{aligned} \tag{8}$$

Most of operations for decryption according to (8) are in $L^{-1}S^{-1}$ transformation except S at the beginning and S^{-1} at the end. Here we assume that decryption round keys $L^{-1}(K_i)$ are given. This assumption is valid when quite a large volume of data are processed on a single working key.

The transformation $L^{-1}S^{-1}$ can obviously be implemented just as LS by the precomputed look-up tables $L_i''(b)$:

$$L^{-1}S^{-1}(a) = L_{15}''(a_{15}) \oplus L_{14}''(a_{14}) \oplus \dots \oplus L_0''(a_0).$$

3 Analysis

3.1 Theoretical performance

To estimate theoretically the performance of the encryption and decryption procedures, we compute how many operations of every kind are required at each round of the iterative LSX-transformation, and the amount of

memory required to store the look-up tables. The operations we took into account are bitwise additions modulo 2 of 128-bit vectors and fetches from the look-up tables of 128- or 8-bit values. All other operations are ignored.

3.1.1 Encryption

The input to the encryption procedure is a plaintext $a = a_{15}\| \ldots \|a_0 \in V_{128}$, $a_j \in V_8,\ j = 0, \ldots, 15$. According to the results of section 2.2, we obtain the following performance estimates.

#	Transformation	Implementation	$\oplus$	Fetches		Memory KBytes
				128	8	
1	$a := X[K_1](a)$	$a := a \oplus K_1$	1	0	0	0
2–10	$a := X[K_i]LS(a)$ $i = 2, \ldots, 10$	$a := L_{15}^S(a_{15}) \oplus \ldots \oplus L_0^S(a_0) \oplus K_i$	9×16	9×16	0	64
		Total:	145	144	0	64

Table 1: Encryption performance

3.1.2 Decryption

The input to the decryption procedure is a ciphertext $a = a_{15}\| \ldots \|a_0 \in V_{128}$, $a_j \in V_8,\ j = 0, \ldots, 15$. According to the results of section 2.3, we obtain the following performance estimates.

We obtain the following implementation decryption algorithm scheme:

#	Transformation	Implementation	$\oplus$	Fetches		Memory KBytes
				128	8	
1	$a := S(a)$	$a := \pi(a_{15})\| \ldots \|\pi(a_0)$	0	0	16	0.25
2–10	$a := L^{-1}X[K_i]S^{-1}(a)$ $i = 10, \ldots, 2$	$a := L''_{15}(a_{15}) \oplus L''_{14}(a_{14}) \oplus \ldots$ $\ldots \oplus L''_0(a_0) \oplus L^{-1}(K_i)$	9×16	9×16	0	64
11	$a := S^{-1}(a)$	$a := \pi^{-1}(a_{15})\| \ldots \|\pi^{-1}(a_0)$	0	0	16	0.25
12	$a := X[K_1](a)$	$a := a \oplus K_1$	1	0	0	0
		Total:	145	144	32	64.5

Table 2: Decryption performance

3.2 Software implementation

We implemented the described method in software. The source code is in the C++ programming language and optimised for 64-bit hardware platforms. The code does not contain any assembler instructions. It neither uses any special registers or invocation of SIMD commands. So it is cross-platform and can easily be ported to any 64-bit platform.

The code was compiled in Visual Studio 2008 environment for x64/Intel platform. The measurements were performed on a single core of Intel i7-2600 @ 3,4GHz processor in Windows 7 OS. The size of the look-up tables is twice as large as the data part of Level 1 cache of the Intel processor. However, the tables are compact enough to definitely fit into Level 2 cache.

The cipher was running in ECB mode of operation. The results are given in table 3.2. The measured speed is given both in Megabytes per second and clock cycles per byte.

The decryption is evidently slower because of the initial S and the pre-final S^{-1} transformations.

Streebog hash-function defined in GOST R 34.11-2012 [3] has a structure similar to prospective block cipher: It is also based on an iterative LSX-transformation. However, particular transformations and the block size are different from those of the cipher. We implemented Streebog-512 using our method. The measured performance is given in table 3.2. There we also gathered known implementation results for Streebog.

	Speed		Platform	Features
	MB/s	cpb		
Prospective 128-bit block cipher				
Encryption	125	26	i7-2600 @ 3.4GHz, Win7	
Decryption	105	31	i7-2600 @ 3.4GHz, Win7	
GOST R 34.11-2012 (Streebog-512)				
Kazimirov [4]	38	67		
This paper	92	35	i7-2600 @ 3.4GHz, Win7	
Lebedev [5]	94	27	i7-920 @ 2.67GHz	SSE4, ASM
Degtyarev [6]	121	28	i7-2600 @ 3.4GHz	SSE4

Table 3: Performance of the prospective cipher and Streebog-512

Conclusion

We presented a method for a high-speed software implementation of the prospective LSX-type 128-bit block cipher. The main approach is to combine the nonlinear substitution S and the linear transformation L into a single operation and to implement it via compact look-up tables. The efficiency of the method depends on the order in which S and L transformations are applied. Using the iterative structure of the cipher, we show how to implement the case when L precedes S almost as efficiently as the case when S precedes L. Our implementation in the C++ language encrypts at 26 and decrypts at 31 clock cycles per byte on a x64/Intel platform.

This method is applicable to Streebog has-function as well. Implemented on the mentioned platform Streebog-512 hash-function hashes data at 35 clock cycles per byte. Moreover, this is the fastest known implementation which does not use any assembler code or SIMD commands.

References

[1] V.A. Shishkin. Design principles of a prospective block cipher with 128 bit block length. — Presentation at RusCrypto'2013 (In Russian). — http://www.ruscrypto.ru/resource/summary/rc2013.

[2] Shishkin V., Dygin D., Lavrikov I., Marshalko G., Rudskoy V., Trifonov D. Low-Weight and Hi-End: Draft Russian Encryption Standard. — In: these pre-proceedings. — P 180–185.

[3] GOST R 34.11-2012. — Information technology. Cryptographic data security. Hash-function. — Moscow, Standartinform, 2012. http://www.tc26.ru/en/GOSTR3411-2012/

[4] Kazymyrov O., Kazymyrova V. Algebraic Aspects of the Russian Hash Standard GOST R 34.11-2012. — In: Proc. of CTCrypt 2013. — Ekaterinburg, 2013. — P. 160–176.

[5] Lebedev P.A. Comparison of old and new cryptographic hash function national standards of Russian Federation on CPUs and NVIDIA

GPUs // Mathematical Aspects of Cryptography, 2013. — Vol. 4., No. 2. — P. 73–80.

[6] https://www.streebog.net/.