Software Implementation of eSTREAM Profile I Ciphers on Embedded 8-bit AVR Microcontrollers*

Gordon Meiser, Thomas Eisenbarth, Kerstin Lemke-Rust, and Christof Paar

Horst Görtz Institute for IT Security
Ruhr University Bochum
44780 Bochum, Germany
{gordon.meiser,eisenbarth,lemke,cpaar}@crypto.rub.de

Abstract. This work is motivated by the question of how efficient candidates in the focus of eSTREAM Profile I can be implemented on small embedded microprocessors that are constrained in memory resources. For the evaluation of the ciphers, the C code implementation provided by the designers is ported to an 8-bit AVR microcontroller and compared with a byte-oriented AES implementation. Our comparison metrics are memory usage in flash and SRAM and performance of keystream generation, key setup, and IV setup. As an overall summary, Salsa20 and LEX turn out to be the most promising candidates. In particular, Salsa20 is attractive because of its compactness in terms of memory. Even though throughput yields are high, SOSEMANUK, Dragon, and HC-128 seem sub-optimum for embedded applications as they require much resources.

Keywords: eSTREAM, stream cipher, software performance, AVR microcontroller, embedded security, Dragon, HC-256, HC-128, LEX, Salsa20, SOSEMANUK, AES.

1 Introduction

In 2005, the European Network of Excellence in Cryptology (ECRYPT) launched a call for stream cipher primitives [6] to identify new stream ciphers suitable for widespread adoption that may also serve as an alternative for the AES [8]. Profile I of this call asked for stream ciphers for software applications with high throughput requirements, while Profile II aims at identifying stream ciphers suitable for hardware applications with restricted resources such as limited storage, gate count, or power consumption.

The original call says that performance benchmarking for Profile I "may include 8-bit processors (as found in inexpensive smart cards), 32-bit processors (e.g., the Pentium family) to the modern 64-bit processors". However, the current testing framework of the eSTREAM project [7, 13, 10] exclusively targets

* Supported by the European Commission through the IST Contract IST-2002-507932 ECRYPT, the European Network of Excellence in Cryptology.

general-purpose 32-bit and 64-bit CPUs for Profile I candidates. Given the great importance of small embedded controllers in the real world (the market share of embedded processors is more than 99%), we feel that such an evaluation is of value.

This paper is driven by the question of how efficient candidates in the current focus of Profile I can be implemented on small 8-bit embedded microprocessors. Small 8-bit microprocessors are constrained in resources such as flash memory and RAM. Besides throughput, efficiency has an additional meaning in this context: resources needed by an implementation of a stream cipher should be kept small, since embedded applications are very often cost constrained. In fact, in many situations cost (given by memory consumption) is more crucial than throughput, in particular since many embedded applications only crypt small payloads.

8-bit embedded microprocessors are widely used in various applications, including smart cards, household appliances, industrial control, cars and many more systems. Modern cars, for instance, are equipped with more than fifty microcontrollers. In embedded systems, cryptography is often needed for authentication, secure messaging, and software download.

Though an 8-bit microprocessor may not be the mainstream target platform of Profile I we are confident that there is also a wide public and industrial interest in finding out whether candidates of Profile I can also serve as possible secure alternative of AES on small 8-bit embedded microprocessors.

In this work, we evaluate performance aspects of stream ciphers that are in the focus of Profile 1 and not broken yet. In detail, this work covers Dragon, HC-256/HC-128, LEX, Salsa20, and SOSEMANUK. Because of recently reported key recovery attacks we have not included Py/Pypy [18] and Phelix [17] in our testing framework.

For the evaluation of Profile I candidates, the C code implementation provided by the designers is ported to an 8-bit AVR microcontroller. For comparison, we also implemented the byte-oriented AES taken from Gladman [15]. Our comparison metric includes (i) throughput of keystream, (ii) time needed for key setup, (iii) time needed for IV setup, (iv) memory allocation in flash (program code), and (v) memory allocation in SRAM (variables).

2 Focused Stream Ciphers in eSTREAM Profile I

This section provides a short description of each cipher. An overview of the ciphers' parameters is given in Table 1.

2.1 Dragon

Dragon is a word-based stream cipher using 32-bit words [14]. There are two proposed modes of operation, either using a 128-bit or a 256-bit key and initialization vector (IV). The cipher is based on a word-oriented non-linear feedback shift register (NLFSR), utilizing an update function and a 64 bit memory, acting

Table 1. Characteristic sizes of the focused ciphers in bits

Cipher	Key size	IV size	State size
Dragon128	128	128	1088
Dragon256	256	256	1088
HC-128	128	128	32768
HC-256	256	256	65536
LEX	128	128	256
Salsa20-128	128	128	512
Salsa20-256	256	128	512
SOSEMANUK	128	128	384

as a counter during keystream generation. The update function is used during key initialization phase as well as during keystream generation phase. It bijectively maps 192 bit values. It uses XOR addition, addition modulo 2^{32} for mixing and six 32×32 s-boxes for adding non-linearity. The s-boxes are deduced from two 8×32 bit s-boxes. The update function is used as a feedback generator for the NLFSR. During keystream generation phase it is also used as a non-linear output function.

Storage requirements include 2048 bytes to store Dragon's two 8×32 s-boxes. The size of the NLFSR is chosen quite big (128 byte) for an implementation on 8-bit microcontrollers because many low cost microcontrollers have RAM in the same order of magnitude.

2.2 HC-128 and HC-256

HC-128 is another 32-bit word-based stream cipher [16]. It uses a 128-bit key and initialization vector. There is also a 256-bit key and IV variant, the HC-256. The main part of the HC-128 are two huge secret tables, each consisting of 512 32-bit elements. The algorithm makes use of different non-linear functions during initialization phase and during key stream generation phase. During initialization phase the secret key and IV are expanded into the two tables. Then the cipher is executed and its output is used to replace all table entries once. During each step of the keystream generation phase, one table element is updated using a non-linear function. Hence the whole table is updated after 1024 steps. Each step one 32-bit element is produced as an output.

The two secret tables of the HC-128 have a total size of 4096 bytes. This exceeds the RAM of many 8-bit microcontrollers by far. Consequently we did not take a closer look at the HC-256 that even has twice the table size. The 32 bit design as well as the possibility of parallelization do not yield an advantage on a microcontroller.

2.3 LEX

The LEX (Leak EXtraction) stream cipher is a modification of the AES Block Cipher [12]. It uses internal states of each round to create the key stream. Amongst

the candidates of this framework, it is the only non-32-bit word oriented cipher. For further information regarding AES, refer to [8]. The standard version of LEX uses a 128-bit key and a 128-bit initialization vector. The setup phase is a simple AES encryption of the IV with the secret key K. From now on the encrypted IV is repeatedly re-encrypted. During each of these encryptions 320 bits of keystream are extracted, 32 bits from each round of the AES. The 32 bits are extracted right after the ShiftRows operation at different locations for even and odd rounds.

By using AES as a base, the designer used many of the assumptions made about the AES. Efficient implementations of the AES are available in hardware as well as in software [1, 15]. Because of the higher output of the LEX (320 bits instead of 128 bits for each full AES encryption), a speed up of a factor of roughly 2.5 can be expected.

2.4 Salsa20

Another 32-bit word-based stream cipher is the Salsa-20 [11]. It uses a hash function with a 64-byte input and a 64-byte output to generate the key stream. Its input consists of an 8-byte nonce and an 8-byte counter, together composing an 128-bit IV, and either a 256-bit or a 128-bit key. Key and IV are padded with a standard sequence to reach the required input length of 64 bytes. Since Salsa20 does not need to prepare an internal state, there exists no setup phase. Due to the counter's length a re-keying is required every 2^{70} bytes. The hash function is built up of word mixing, addition modulo 2^{32}, xor-addition and word-wise rotation.

The lack of a huge state and the simplicity of the operations favors the cipher's implementation on an 8-bit microcontroller.

2.5 SOSEMANUK

The SOSEMANUK stream cipher is based on the SNOW 2.0 stream cipher and uses parts of the SERPENT block cipher as well [9]. Like most of the other ciphers it is based on 32-bit words. It uses a 128-bit key and IV. The key can be bigger, up to a length of 256 bit, but no security increase is claimed for this by the authors. The cipher consists of a 10-word LFSR and a 2-word finite state machine (FSM) to produce the output. For the setup phase a modified SERPENT, reduced to 24 rounds, is used to fill the LFSR as well as the FSM. During key stream generation four consecutive output words of the FSM are fed through the SERPENT's third s-box and then xor-ed to the output words of the LFSR to produce the keystream.

Like LEX, SOSEMANUK may benefit from efficient existing implementations, SNOW and SERPENT in this case.

3 AVR Microcontroller Family

AVR microprocessors are a family of 8-bit RISC microcontrollers. The individual device classes differ in SRAM and flash memory size, as listed in Table 2. Its memory is organized as a Harvard architecture with a 16-bit word program memory and an 8-bit word data memory. Most of the microcontroller's instructions are one-cycle. All of the microcontrollers listed in Table 2 can be clocked at up to 16 MHz.

Due to its easy usage, its low power consumption and its comparatively low price, the AVR microcontrollers have reached a high popularity in embedded system design.

Table 2. Specification of the most popular AVR devices (ATmega family)

Device	Flash [kbyte]	SRAM [byte]
ATmega8	8	1024
ATmega16	16	1024
ATmega32	32	2048
ATmega64	64	4096
ATmega128	128	4096
ATmega1281	128	8192

4 Framework Set-Up and Tools

In this section we describe how the API-conform eSTREAM implementations are ported to AVR microcontrollers. We also give an introduction to the software development tools and how we measure clock cycles, flash size and SRAM size.

4.1 Porting to AVR Microcontrollers

The ciphers come with a set of associated files according to the eSTREAM API. In order to reduce the size of the code and to solve the dependencies we move only the parts of each file that are required for execution into one AVR file. One problem in porting code to an AVR microcontroller is the limited amount of SRAM. A solution for saving valuable SRAM lies in moving S-boxes or comparable big static data arrays into flash memory. Another issue are the different sizes of integer variables in a 32-bit-oriented environment and an 8-bit-oriented environment of an AVR. Thus all variables used have to be adapted to the standard integer sizes of the AVR microcontroller.

4.2 Development Tools

For the software development we used the tool 'WinAVR' [5]. This is a suite of executable, open source software development tools for the Atmel [2] AVR series

of RISC microprocessors hosted on the Windows platform. WinAVR contains avr-gcc (compiler), avrdude (programmer), avr-gdb (debugger) and a tool for automatic makefile generation. Once the code compiles without errors we used the output files from WinAVR to execute the code in 'AVR Studio 4' [4] and simulate it on a chosen AVR device. AVR Studio 4 is an Integrated Development Environment (IDE) for writing and debugging AVR applications on the Windows platform. We are able to use all the functions as known by common debugging tools such as watching registers and variables. At every state we can obtain the number of CPU cycle counts, which enables us to measure clock cycles for benchmarking throughput.

Code size and SRAM size are measured in WinAVR. After compilation, WinAVR shows a summary of the used flash and SRAM size. We use the tool avr-sizex [3] to display these values in percentage. It has to be pointed out that SRAM size as provided by WinAVR includes only static variables that are initialized at the beginning. Because of that, WinAVR returns a smaller value than the actual required SRAM size. Especially, the cipher specific structure ECRYPT_ctx is not included in this value because it is initialized during runtime and is non-static. To provide a better statement on actual SRAM size the byte size of the cipher specific structure ECRYPT_ctx is calculated manually and added to the size of the static variables.

Remark 1. We can not acknowledge all given test vectors of LEX. The test vector with IV = {0x1000..} still delivers a wrong keystream.

4.3 Configuration for Testing

Here we explain the general structure of the performance evaluation based on the ECRYPT API functions. The test sequence is the following (all data in pseudo code):

```
ECRYPT_init()
ECRYPT_keysetup(key)
ECRYPT_ivsetup(iv)
ECRYPT_process_bytes(encryption, blocksize)
ECRYPT_ivsetup(iv)
ECRYPT_process_bytes(decryption, blocksize)
```

The parameter `blocksize` is adapted for each cipher. After each call of a function the CPU cycles needed are recorded by using the AVR Studio 4.

5 Results

This section provides the results on efficiency.

5.1 Memory Usage

A microcontroller holds restrictions in the size of available flash memory and SRAM. Flash memory is used to store static information like program code or huge look-up tables. The usually even smaller SRAM is used for dynamic access during program execution. The memory requirements of each cipher's implementation determine the smallest possible AVR device.

Table 3 shows the memory allocation in flash memory. The second column provides the smallest device on which the implementation of the cipher can be executed without errors. The third column displays the used size of the flash memory in bytes and the last column translates this value to percentage in relation to the maximum size of the flash memory of the associated device in the second column.

Table 3. Memory allocation in flash

Cipher	AVR device	Flash size [byte]	Flash usage on device [%]
AES	ATmega16	6664	40
Dragon	ATmega128	57434	43
HC-128	ATmega1281	23100	18
LEX	ATmega32	21312	65
Salsa20	ATmega8	4478	54
Salsa20 improved	ATmega8	3842	46
SOSEMANUK	ATmega64	44704	68

Remark 2. In the tables there exist a cipher named 'Salsa20 improved'. This is a slightly modified version of the original Salsa20 implementation. We merely substitute the original rotation macro with four improved rotation functions for left rotation of 7, 9, 13 and 18 bits. As shown in Table 5 with our improved version of Salsa20, there is a great saving of cycles when replacing the rotation macro with a rotation function that uses permutation of bytes prior to rotations and adapts better to an 8-bit microcontroller. The original macro does 32 shifts, no matter how many bits should be rotated. This improvement saves nearly 75% of cycles needed by the original macro.

Remark 3. On an 8-bit microcontroller, LEX can be assumed to result in the same code size of the byte-oriented AES. Consequently, it can be assumed that a byte-oriented LEX can fit in an ATmega16 device.

AES, Dragon, LEX, Salsa20, Salsa20 improved and SOSEMANUK[1] can not be reduced to smaller AVR devices because of their consumption of flash memory[2]. HC-128 instead has to use the ATmega1281 device because of its immense usage of the SRAM. This will be shown in Table 4.

Table 4 has nearly the same structure as Table 3, but focuses on the amount of SRAM needed by the ciphers. Column 4 shows the requirement of the cipher specific structure ECRYPT_ctx in bytes, which represents the internal state of the cipher. This value is important because WinAVR disregards dynamic variables in the value given in column 3 as discussed in Section 4.2. Column 5 provides the percentage of SRAM used in the associated AVR device based on the sum of column 3 and 4.

Table 4. Memory allocation in SRAM

Cipher	AVR device	Static variables [byte]	ECRYPT_ctx [byte]	SRAM usage on device [%]
AES	ATmega16	88	241	32
Dragon	ATmega128	424	405	20
HC-128	ATmega1281	232	4324	56
LEX	ATmega32	200	232	21
Salsa20	ATmega8	258	64	31
Salsa20 improved	ATmega8	258	64	31
SOSEMANUK	ATmega64	192	448	16

5.2 Performance

In the following performance benchmarks we use a key and IV size of 128 bit. Input and output arrays are of the size of the block size of each cipher. This means that we encrypt one block with each cipher.

Table 5 shows the number of cycles for key setup, IV setup, encryption and decryption for each cipher.

As seen in Table 5 HC-128 has an immense cycle count in the iv_setup function. However, HC-128 achieves the lowest number of cycles for encryption of one block of data.

Table 6 focuses on the throughput of the encryption function for each cipher while Table 5 gives the number of cycles for one block size. Column 2 in Table 6 shows the corresponding block size in bytes and column 3 reprints the count of

[1] By replacing all existing macros with functions this results in a flash size of 24648 bytes for SOSEMANUK, i.e., SOSEMANUK fits in an ATmega32 device. As a compactness similar to AES is not achieved, this implementation variant of SOSEMANUK is not considered further in this paper.

[2] Though the flash size entries of Table 3 seem to indicate that AES, Dragon and Salsa20 improved can be implemented on a smaller AVR device, actually, this is not possible.

Table 5. Performance of key setup, IV setup, encryption and decryption. All numbers given are measured CPU cycles.

Cipher	Key setup	IV setup	Encryption	Decryption
AES	6953	196	12574	14815
Dragon	2136	24052	24227	24222
HC-128	460	2082876	10804	10799
LEX	2619	7288	8061	8056
Salsa20	249	71	90802	90826
Salsa20 improved	248	70	48942	48965
SOSEMANUK	32851	33972	14134	14129

cycles from Table 5. Column 4 is the quotient of column 3 and 2 and column 5 shows the throughput of the encryption function. The throughput is computed by dividing the CPU clock (assuming 8 MHz) by the value in column 4.

Table 6. Throughput of Encryption

Cipher	Block size [byte]	Encryption [cycles]	Encryption [cycles/byte]	Throughput [bytes/sec]
AES	16	12574	785.88	10179
Dragon	128	24227	189.27	42267
HC-128	64	10804	168.81	47390
LEX	40	8061	201.53	39696
Salsa20	64	90802	1418.78	5638
Salsa20 improved	64	48942	764.72	10461
SOSEMANUK	80	14134	176.68	45279

As shown in Table 6 as well as in Fig. 1 and Fig. 2, the ciphers can be classified into two groups regarding throughput. HC-128, SOSEMANUK, Dragon and LEX are in the fast group. Salsa20 improved, AES and Salsa20 reside in the slow group. Notable is the fact that the improved Salsa20 implementation nearly produces twice the output of the original Salsa20 implementation. Note further that for long keystreams the encryption is the dominant factor (but remember the huge time for IV setup of HC-128).

When only a small amount of keystream has to be generated, it can be seen from Table 6 and Fig. 1 that LEX is the most efficient. Until 48 byte keystream, AES is the second most efficient that is then passed by Dragon.

6 Perspectives and Further Work

Most ciphers are already strongly optimized for the C language and 32 bit CPUs. So it is only feasible to improve the source code at particular positions. We expect that certain improvements – similar to the ones for Salsa20 – are still feasible

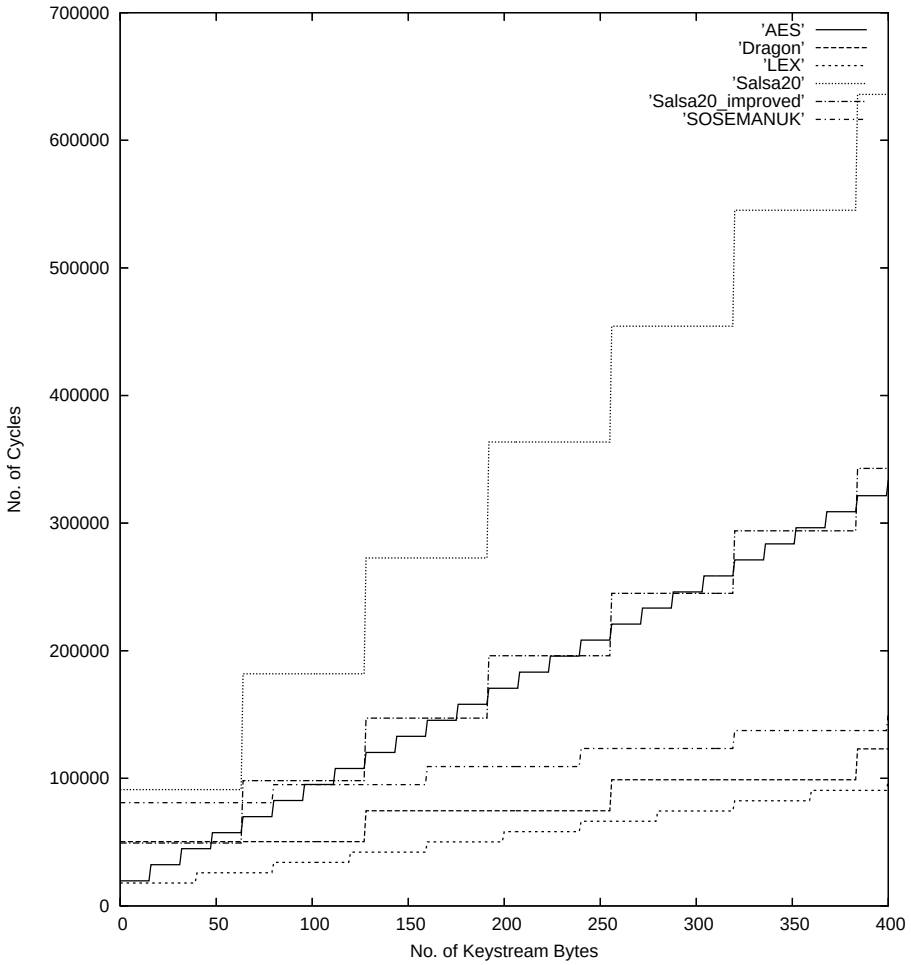

Fig. 1. Cycles versus number of keystream bytes for up to 400 bytes of keystream. Note that HC-128 is not printed here as it is by far the least efficient cipher in this domain. LEX turns out to be most efficient.

at the other ciphers. Disintegration of the given eSTREAM API structure can save another few cycles.

Programming in assembly language offers significant advances both in code size and throughput (for example, AVR instructions are available to rotate registers). Further work will deal with providing a compact implementation of the most promising ciphers in Assembly language.

Moreover, one may also include ciphers that are not in the focus of Profile I in this testing framework. We encourage to consider performance on 8-bit pro-

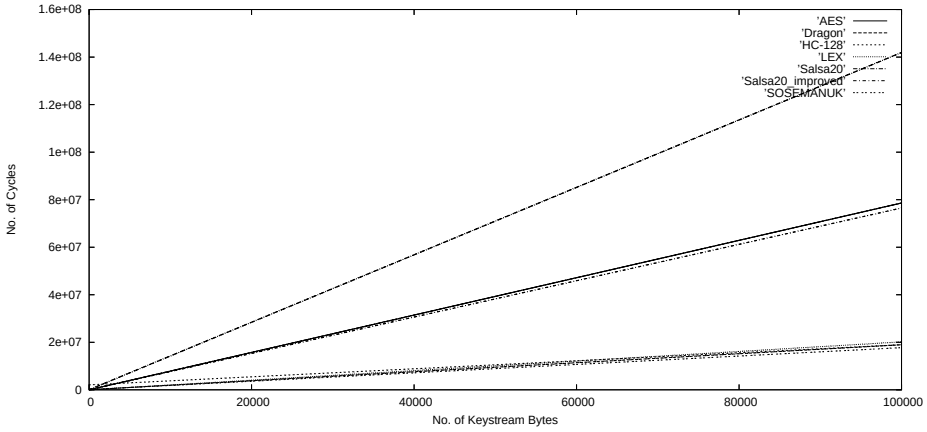

Fig. 2. Cycles versus number of keystream bytes for up to 100,000 bytes of keystream. In this domain three groups are clearly separated. The group of the most efficient ciphers includes SOSEMANUK, HC-128, Dragon, and LEX (from the most efficient to the least efficient at 100,000 bytes of keystream).

cessors as one further criterion for the suitability of a cipher in the eSTREAM project.

7 Conclusion

A performance analysis of eSTREAM Profile I candidates on 8-bit AVR microcontrollers has been applied. If considering memory requirements Salsa20 is the most promising candidate, especially, as it achieves a better compactness in code and data size than AES and LEX and fits in the smallest AVR device. SOSEMANUK, Dragon, and HC-128 consume much memory and do not fit in small 8-bit AVR devices. In terms of throughput the ranking depends on the number of keystream bytes. For a small amount of keystream bytes, the best results are obtained for LEX, AES, and Dragon. HC-128 needs extremely long for IV setup and is not of interest in this domain. For long keystreams, ciphers with the best throughput are HC-128, SOSEMANUK, Dragon, and LEX. In this domain, Salsa20 is slightly faster than AES.

As an overall summary, Salsa20 and LEX turn out as the most promising candidates for small embedded 8-bit microcontrollers. In particular, Salsa20 is attractive because of its compactness in terms of memory. SOSEMANUK and Dragon seem sub-optimum for embedded applications. Because of its immense use of SRAM memory, HC-128 cannot be recommended for small embedded processors.

References

1. AES Lounge. Available from: http://www.iaik.tugraz.at/research/krypto/AES/.
2. ATMEL Corporation. Available from: http://www.atmel.com.
3. AVR-sizex. Available from: http://alt.kreatives-chaos.com/download/avr-sizex.exe.
4. AVR Studio 4.12, Atmel Corporation, Build 460, November 2005. Available from: http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725.
5. WinAVR : avr-gcc for Windows, Release 20060421, 21th April 2006. Available from: http://winavr.sourceforge.net/.
6. Call for Stream Cipher Primitives, Version 1.3, 12th April 2005. eSTREAM, ECRYPT Stream Cipher Project, 2005. Available from: http://www.ecrypt.eu.org/stream/call/.
7. eSTREAM – Update 1, September 2, 2005. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/057, 2005. Available from: http://www.ecrypt.eu.org/stream/papersdir/057.pdf.
8. FIPS 197. Announcing the AES, November 2001.
9. C. Berbain, O. Billet, A. Canteaut, N. Courtois, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin, and H. Sibert. SOSEMANUK, a fast software-oriented stream cipher. Available from: http://www.ecrypt.eu.org/stream/p2ciphers/sosemanuk/sosemanuk_p2.pdf.
10. D. J. Bernstein. Notes on the ECRYPT Stream Cipher project (eSTREAM). Available from: http://cr.yp.to/streamciphers.html.
11. D. J. Bernstein. Salsa20. Available from: http://www.ecrypt.eu.org/stream/p2ciphers/salsa20/salsa20_p2.zip.
12. Alex Biryukov. A new 128-bit key stream cipher LEX. Available from: http://www.ecrypt.eu.org/stream/p2ciphers/lex/lex_p2.zip.
13. Christophe De Cannière. eSTREAM testing framework. eSTREAM, ECRYPT Stream Cipher Project. Available from: http://www.ecrypt.eu.org/stream/perf/.
14. K. Chen, M. Henricksen, W. Millan, J. Fuller, L. Simpson, E. Dawson, H. Lee, and S. Moon. Dragon: A fast word based stream cipher. Available from: http://www.ecrypt.eu.org/stream/p2ciphers/dragon/dragon_p2.pdf.
15. Brian Gladman. Brian Gladman's Home Page. Available from: http://fp.gladman.plus.com/AES/index.htm.
16. Hongjun Wu. The stream cipher HC-128. Available from: http://www.ecrypt.eu.org/stream/p2ciphers/hc256/hc128_p2.pdf.
17. Hongjun Wu and Bart Preneel. Differential-Linear Attacks against the Stream Cipher Phelix. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/056. Available from: http://www.ecrypt.eu.org/stream/papersdir/2006/056.pdf.
18. Hongjun Wu and Bart Preneel. Key Recovery Attack on Py and Pypy with Chosen IVs. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/052. Available from: http://www.ecrypt.eu.org/stream/papersdir/2006/052.pdf.