Security and Implementation Properties of ABC v.2

Vladimir Anashin[1], Andrey Bogdanov[2], and Ilya Kizhvatov[1]

[1] Russian State University for the Humanities,
Institute for Information Sciences and Security Technologies,
Faculty of Information Security,
Kirovogradskaya Str. 25/2, 117534 Moscow, Russia
{anashin,kizhvatov}@rsuh.ru

[2] escrypt GmbH – Embedded Security
Lise-Meitner-Allee 4, D-44801 Bochum, Germany
abogdanov@escrypt.com

Abstract. ABC is a synchronous stream cipher submitted to eSTREAM. Here we describe ABC v.2 – a tweaked version of ABC. The tweaks made ABC v.2 resistant to certain attacks, including the ones presented by Berbain and Gilbert and by Khazaei. We give a design rationale and a brief security analysis of ABC v.2. Also it is shown that the distinguishing attacks against ABC v.2 like the one suggested by Khazaei and Kiaei are totally impractical. ABC v.2 is extremely fast in software often heading the eSTREAM benchmark list. Further we define informal requirements for an industrial software stream cipher and show that ABC v.2 meets them. Moreover, we demonstrate that ABC v.2 is also suitable for embedded security applications demanding high performance.

Keywords: cryptography, stream cipher, ABC, eSTREAM, ECRYPT, distinguishing attack, stream cipher performance

1 Introduction

ABC is a synchronous stream cipher optimized for software applications which was submitted to eSTREAM [7]. ABC v.2 [8] with a 128-bit key and 32-bit internal variables, offers 128-bit security and is extremely fast in software often heading the eSTREAM performance benchmark list and ranking first in packet encryption [2].

This paper first outlines the tweaks to the original ABC that lead to ABC v.2. Then the attacks and the way the tweaks make ABC v.2 resistant to these attacks are described. Another possible tweak is discussed. We also show that 'Theorem 1' from the paper [12] by S. Khazaei describing an attack on ABC is wrong.

It is shown that the paper [13] by S. Khazaei and M. Kiaei does not present any distinguishing attack both on ABC v.1 and ABC v.2. The results of experiments are presented, indicating that the distinguisher for ABC v.2 has a complexity greater than that of a brute force attack.

Apart from its security properties, ABC v.2 meets a set of requirements which distinguish a stream cipher well suited for the real-world applications according to a number of features. We call these *industrial software implementation requirements* which are the following:

– High generic performance for all software platforms including embedded ones (at least twice as fast as AES on the same platform),
– Low memory consumption,
– Low costs of IV and key setup procedures.

Since these properties are mutually contradictory (e.g. more precomputations allow as a rule a faster implementation which leads, however, to a higher memory consumption), the latter two of them can be substituted for *flexibility* which means that a good industrial cipher should be capable of an efficient throughput/memory trade-off. ABC v.2 meets these requirements which is shown in the paper.

Actually ABC is a family of stream ciphers. This implies not only the flexibility of ABC implementation, but also the natural flexibility of the ABC design, which enabled us in [6] to suggest the tweaks raising its keystream period from $2^{32} \cdot (2^{63} - 1)$ 32-bit words to $2^{32} \cdot (2^{127} - 1)$ 32-bit words while keeping all the other properties of ABC stated in [7], including guaranteed uniform distribution and high linear complexity of the keystream.

Moreover, the ABC stream cipher is highly scalable which gives a possibility of natural extension of the cipher to a larger computational base (e.g. 64-bit version of ABC) and to exchange its separate components with very low overhead. This was done in ABC v.2 and can be further extended to create a version of ABC providing 256-bit security with a negligible performance overhead.

The paper is organized as follows. In Section 2 ABC v.2 is introduced and its differences from ABC v.1 are discussed. Section 3 describes a class of distinguishing and correlation attacks which could be applicable to ABC v.1 and ABC v.2. In Section 4 a number of ways avoiding this attack possibilities are suggested and the remedy selection for ABC v.2 is motivated. Section 5 provides experimental evidence demonstrating that ABC v.2 is robust to the distinguishing attack. In Section 6 we consider the industrial software implementation requirements, show that ABC v.2 meets them, discuss in what way ABC v.2 is superior to the other eSTREAM ciphers and demonstrate that ABC v.2 clearly outperforms AES on embedded platforms. We conclude in Section 7.

2 Moving from ABC v.1 to ABC v.2

Here the tweaks in the ABC keystream generator making ABC v.2 out of ABC v.1 are briefly outlined. The adjusted setup procedures described in [6,8] are not discussed here, we just note that some inaccuracy concerning the initialization routine mentioned in [9] was corrected. The following notation is used in the description of the cipher.

$x, y \in \mathbb{Z}/2^{32}\mathbb{Z}$ denote the state of the function B and the output of the keystream generator respectively;

z is a 128-bit integer value for ABC v.2 and a 64-bit integer value for ABC v.1 denoting the state of the transform A; it can also be represented as $z = (\bar{z}_3, \bar{z}_2, \bar{z}_1, \bar{z}_0) \in (\mathbb{Z}/2^{32}\mathbb{Z})^4$ for ABC v.2 and $z = (\bar{z}_1, \bar{z}_0) \in (\mathbb{Z}/2^{32}\mathbb{Z})^2$ for ABC v.1, $\bar{z}_3, \bar{z}_2, \bar{z}_1, \bar{z}_0 \in \mathbb{Z}/2^{32}\mathbb{Z}$;

$d_0, d_1, d_2, e, e_0, e_1, \ldots, e_{31} \in \mathbb{Z}/2^{32}\mathbb{Z}$ denote the coefficients of the transforms B and C respectively;

$w \in \mathbb{Z}/2^5\mathbb{Z}$ denotes the length in bits of the optimization window used in computation of the transform C;

$\delta_i(\cdot)$ is the i-th bit selection operator returning the value of the i-th bit of an integer, e.g. $\delta_0(x)$ is the least significant bit of x;

$\oplus$ is the bitwise modulo 2 addition ('XOR') operation;

$\ll, \gg, \ggg$ denote correspondingly left (zero-fill) bit shift, right (zero-fill) bit shift and right rotation of binary expansion of a 32-bit integer.

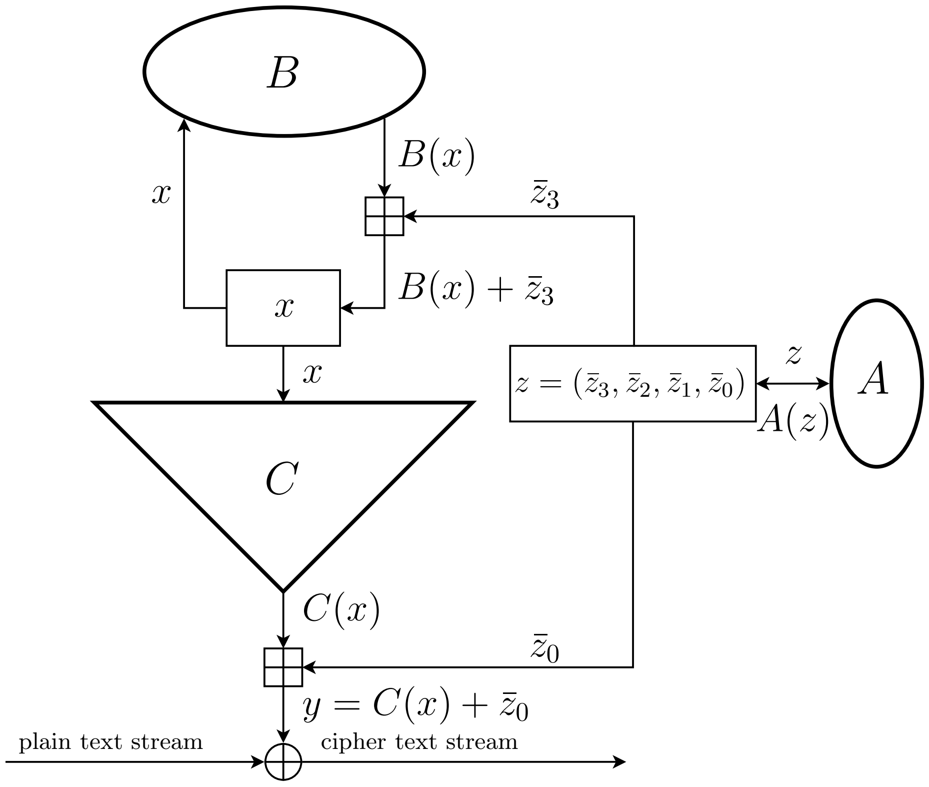

Fig. 1. ABC v.2 keystream generator

The keystream generator of ABC v.2 is illustrated in Fig. 1. In both versions of ABC A is a linear transformation of the vector space $\mathbb{V}_n = \mathrm{GF}(2)^n$ with a cycle of length $2^{n-1}-1$ (where $n = 128$ for ABC v.2, and $n = 64$ for ABC v.1), B is a single cycle T-function on 32-bit words, and $C\colon \mathbb{Z}/2^{32}\mathbb{Z} \to \mathbb{Z}/2^{32}\mathbb{Z}$ is a filter function: C takes x as argument and produces y in the following way:

$$\begin{aligned} \zeta &= S(x), \\ y &= \zeta \ggg 16, \end{aligned} \tag{1}$$

where $\zeta \in \mathbb{Z}/2^{32}\mathbb{Z}$ and $S : \mathbb{Z}/2^{32}\mathbb{Z} \to \mathbb{Z}/2^{32}\mathbb{Z}$ is a mapping defined by

$$S(x) = e + \sum_{i=0}^{31} e_i \delta_i(x) \bmod 2^{32}, \tag{2}$$

$e_{31} \equiv 2^{16} \pmod{2^{17}}$. Coefficients $e, e_0, \ldots, e_{31} \in \mathbb{Z}/2^{32}\mathbb{Z}$ are obtained from the key during the initialization procedure.

The single cycle function B used in the ABC v.2 cipher can be specified through the following equation:

$$B(x) = ((x \oplus d_0) + d_1) \oplus d_2 \bmod 2^{32}, \tag{3}$$

where $d_0 \equiv 0 \pmod 4$, $d_1 \equiv 1 \pmod 4$, $d_2 \equiv 0 \pmod 4$. In the non-modified ABC v.1 the function B was of the form

$$B(x) = d_0 + 5(x \oplus d_1) \bmod 2^{32}, \tag{4}$$

with $d_0 \equiv 1 \pmod 2$, $d_1 \equiv 0 \pmod 4$.

Under the restrictions mentioned above the following properties of the keystream produced by the ABC v.2 keystream generator are proved:

- The length P of the shortest period of the keystream sequence of 32-bit words is $P = 2^{32} \cdot (2^{127} - 1)$.
- The distribution of the keystream sequence of 32-bit words is uniform in the following sense: For each 32-bit word a the number $\mu(a)$ of occurrences of a at the period of the keystream satisfies the following inequality:

$$\left| \frac{\mu(a)}{P} - \frac{1}{2^{32}} \right| < \frac{1}{\sqrt{P}}.$$

- The linear complexity λ of the keystream bit sequence satisfies the inequality $2^{31} \cdot (2^{127} - 1) + 1 \geq \lambda \geq 2^{31} + 1$.

Proofs are based on the results presented in [4] and can be found in the updated ABC specification [8].

3 Attack Possibilities

In this section we describe some attacks that lead to recovering the internal state of the (non-modified) ABC v.1, and which are more efficient than a brute force attack. The corresponding remedies are discussed in Section 4.

Suppose that one has a statistical test $\mathcal{T}$ (which is further called a *distinguisher*) that could tell the keystream sequence $Y = \{y_j \in \mathbb{Z}/2^{32}\mathbb{Z}\}_{j=0}^{\infty}$ from the intermediate sequence $C(X) = \{C(x_j) \in \mathbb{Z}/2^{32}\mathbb{Z}\}_{j=0}^{\infty}$, which is the output of the function C. Then trying different initial states $\hat{z}$ of the LFSR A and testing the sequences $\overline{C(X)}(z) = \{y_j - \bar{z}_{0,j}(\hat{z}) \bmod 2^{32}\}$ with $\mathcal{T}$, where $\bar{z}_{0,j}(\hat{z})$ is the the 32 low order bits of the output of the LFSR A at the j-th step, one finds $\bar{z}$.

In other words, if the guess for LFSR state is correct, subtracting the LFSR sequence from the keystream sequence results in bare C output. If the guess for LFSR state is incorrect, the subtracting leads to some other sequence $\overline{C}$. Now, if we distinguish C from $\overline{C}$, we determine the correct guess. Actually, the awaited statistical properties of $\overline{C}$ are as good as those of the keystream sequence Y. So from the point of view of simplest and effective distinguishers $\overline{C}$ and Y are the same. That is why C can be distinguished from $\overline{C}$ by such a distinguisher that can tell C from Y.

Under the assumption that $\mathcal{T}$ makes no errors in distinguishing, the computational cost of finding the true initial state of the LFSR is $(2^n - 1)T$ computations of AB, where T is the computational cost of testing one sequence with the test $\mathcal{T}$, and n is the length of the LFSR registry (i.e., $n = 63$ in non-modified ABC, and $n = 127$ in the modified one). After finding the true initial state $\hat{z}$ of the LFSR, one tests coefficients of the function B and then, solving the corresponding congruences modulo 2^{32} with respect to the unknown values of $e, e_0, \ldots, e_{31}$, totally recovers the internal state of the ABC.

Attacks of this kind were mounted by Berbain and Gilbert in [9], and by Shahram Khazaei in [12]. They were successfully thwarted (actually prior to their publishing) by the ABC v.2 update, containing the remedies described in the next section.

4 Remedies

We need only those remedies that do not worsen the important properties of ABC (long period, uniform distribution and high linear complexity of the keystream) and/or significantly reduce its performance. There are several such remedies; two of them are described below.

4.1 Remedy 1: Special Coefficients

Since the coefficients $e, e_0, \ldots, e_{31}$ of the function S of (2) are produced in a pseudorandom way during the initialization stage, the probability the mapping C of (1) is bijective is too small; see Corollary 1 below for the exact value of that probability (the estimate of [9] is just an empirical conjecture and the one of [12] is based on the erroneous 'Theorem 1' of [12]). Hence, with high probability the distribution of the sequence $C(X) = \{C(x_j) \in \mathbb{Z}/2^{32}\mathbb{Z}\}_{j=0}^{\infty}$, is not uniform since the distribution of the sequence $X = \{x_j \in \mathbb{Z}/2^{32}\mathbb{Z}\}_{j=0}^{\infty}$, which is the output if the function B, is uniform. This follows from the results stated [4] and can be found in [8]. At the same time, the distribution of the keystream sequence $Y = \{y_j \in \mathbb{Z}/2^{32}\mathbb{Z}\}_{j=0}^{\infty}$ is uniform (see Section 2). Hence, the distribution of the sequence $\overline{C(X)}(\hat{z})$ is not uniform in case of the right guess of the initial state $\hat{z}$ of the LFSR A, since the distribution of the output sequence of the LFSR A is uniform.

Thus, a distinguisher $\mathcal{T}$ just tests the uniformity of distribution of the sequence $\overline{C(X)}(z)$ for various z; in case the distribution is not uniform, the corresponding $z = \hat{z}$ is accepted as a true one. Distinguishers of [9] and of [12] are exactly of this sort.

To make sequences $\overline{C(X)}(\hat{z})$ indistinguishable one from another with respect to the test $\mathcal{T}$ for all the choices of $\hat{z}$ it suffices to choose coefficients of S in some special way to ensure that S is bijective.

Thus one needs criteria the coefficients should satisfy to make S bijective. In [12, Theorem 1] the following 'criterion' is stated: The function

$$\begin{array}{l} S(x) = e + \sum_{i=0}^{k-1} e_i\delta_i(x) \pmod{2^k}, \\ x, e, e_i \in \mathbb{Z}/2^k\mathbb{Z}, i = 0, \ldots, k-1, \end{array} \tag{5}$$

induces a permutation of the residue ring $\mathbb{Z}/2^k\mathbb{Z}$ iff for each non-empty subset $M \subset \{0, 1, \ldots, k-1\}$

$$\sum_{i \in M} e_i \not\equiv 0 \pmod{2^k}.$$

However, it could be immediately shown that the above 'criterion' (as well as the whole 'Theorem' 1 of [12]) *are merely wrong*: Take $k = 3$, put $e_0 = 1$, $e_1 = 2$, $e_2 = 3$ and verify that the mapping $x \mapsto \delta_0(x) + 2 \cdot \delta_1(x) + 3 \cdot \delta_2(x)$ is *not* a permutation of the residue ring modulo 8.

The right criterion reads the following.

Theorem 1. *The function* (5) *induces a permutation on the ring* $\mathbb{Z}/2^k\mathbb{Z}$ *if and only if*

$$e_{j_0} \equiv 1 \pmod 2, \quad e_{j_1} \equiv 2 \pmod 4, \ldots, e_{j_{k-1}} \equiv 2^{k-1} \pmod{2^k},$$

for some permutation $(j_0, j_1, \ldots, j_{k-1})$ *of* $(0, 1, \ldots, k-1)$.

Corollary 1. *There are exactly* $k! \cdot 2^{\frac{k(k+1)}{2}}$ *permutations among all* $2^{k(k+1)}$ *pairwise distinct transformations of the form* (5) *of the residue ring* $\mathbb{Z}/2^k\mathbb{Z}$. *Hence, the probability that* S *is a permutation is* $k! \cdot 2^{-\frac{k(k+1)}{2}}$.

In other words, S of (2) is a permutation iff $e_0, \ldots, e_{31}$ could be reordered so that $e_i = 2^i \cdot e_i'$, where e_i' are odd, $i = 0, 1, \ldots, 31$. Note that our condition $e_{31} \equiv 2^{16} \pmod{2^{17}}$ is in a certain sense a 'remnant' of our Theorem 1.

Theorem 1 follows immediately from a (more than 10 year old) result of one of us, see [3, Proposition 4.8]. Also, it could be easily deduced from the older result of DeBruijn, see [15, Section 4.1, Exercise 30]. Of course, it is not difficult to prove this theorem directly.

Thus, just to avoid the kind of attack described in [9] and [12] it is sufficient *only to make minor modifications to the initialization procedure* so that one of $e_0, \ldots, e_{31}$ always has 1 in the least significant bit position, another has 01 in its two rightmost bit positions, a further one has 001 in the three rightmost bit positions, etc. *The modification does not change the ABC keystream generation routine at all*, leaving both the performance and other properties (period length, uniform distribution, linear complexity) unchanged.

So the assumption of [12] by S. Khazaei that 'The designers of ABC have not neither evaluated C function theoretically nor using statistical simulations and just have designed C function to provide a provably minimum period for its output sequences' is just not true. We certainly could make S (whence, C) balanced (that is, bijective) at the very first stage of the ABC design procedure: We had mathematical tools to construct balanced mappings. These tools have been developed long before (see e.g. the bibliography in [7] and [8]) and are more effective than the ones of paper [14]. However, the *arbitrary* choice of coefficients in accordance with our Theorem 1 *might lead to some attacks* unless some special countermeasures are undertaken.

4.2 Remedy 2: Long LFSR

This solution is based on the usage of LFSR with period $2^{127} - 1$ instead of the LFSR with period $2^{63} - 1$ in the keystream generator, see Fig. 1. In spite of the fact that it implies modification of the keystream routine (we had also to modify the B function to compensate some speed reduction), the solution makes the ABC resistant to *all possible* attacks of the described kind *independently* of concrete distinguishers $\mathcal{T}$ they are based on: The computational cost is then $(2^{127} - 1) \cdot T \approx 2^{127} \cdot T \geq 2^{128}$, since we could hardly imagine a distinguisher with computational cost $T = 1$, under every reasonable definition of what the computational cost is. Thus, every attack of the described type becomes less effective than a brute force attack. As a bonus we obtain certain increase of security of the function B, since some extra bits of security are added (cf. (3) and (4)).

4.3 Scalability of ABC Design

The architecture of ABC stream cipher is highly scalable. This provides one with a possibility of natural extension of the cipher.

First, the extension can aim at a larger computational base, e.g. 64-bit version of ABC for 64-bit platforms, such as Intel Itanium or PowerPC G5. Second, the separate components of ABC, namely, A, B and C transforms, can be exchanged with a very low overhead.

Moreover, a natural extension of the digit capacity of the components of ABC can lead to a more secure and efficient cipher. This was done in ABC v.2 and can be further extended to create a version of ABC providing 256-bit security with a negligible performance overhead. Such a version of ABC with a 256-bit word-oriented LFSR, 256-bit key and 256-bit IV encrypts at 4.21 processor cycles per byte (the measurements were performed on a 1.73 GHz Intel Pentium M processor using the eSTREAM testing framework), which is only 4 percent more than for ABC v.2 with a 128-bit LFSR.

5 The Impracticability of Some Distinguishing Attacks

In their paper [13] Shahram Khazaei and Mohammad Kiaei claimed that there is a distinguisher on both versions of ABC with the complexity of about 2^{32}. The claim was supported by the empirical results of computer experiments with a set of reduced versions of ABC. However, the authors of [13] have made multiple errors, see [5] for details.

The idea of the possible attack is to reduce the influence of the LFSR A on the keystream and then detect a bias originating from the non-balanced filter function C. One can imagine that the LFSR influence can be reduced by applying an annihilator of the LFSR sequence to the keystream sequence. The word-oriented recurrence relation of A for ABC v.2 [8] induces the word-oriented annihilator

$$\bar{z}_{0,i} \oplus \bar{z}_{0,i-2} \oplus (\bar{z}_{0,i-3} \lll 31) \oplus (\bar{z}_{0,i-4} \ggg 1) = 0, \tag{6}$$

where $\bar{z}_{0,i-4}, \ldots, \bar{z}_{0,i}$ are the successive states of the word $\bar{z}_0$ of the LFSR A.

The application of (6) to the keystream sequence $\{y_n\} = \{c_n\}+\{z_{0,n}\}$ formed by the output of function C $\{c_n\}$ and the output of LFSR $\{z_{0,n}\}$ results in the sequence $\{u_n\}$ where

$$u_i = y_i \oplus y_{i-2} \oplus (y_{i-3} \lll 31) \oplus (y_{i-4} \ggg 1), \quad u_i \in \mathbb{Z}/2^{32}\mathbb{Z}. \tag{7}$$

According to the idea of approximating the arithmetic addition modulo 2^{32} with the bitwise exclusive OR exploited in [13] the keystream word can be seen as $y_i = c_i \oplus z_{0,i} \oplus w_i$. The term $w_i \in \mathbb{Z}/2^{32}\mathbb{Z}$ is the noise induced by carries of the arithmetic addition. Hence from (7)we have

$$u_i = c_i \oplus c_{i-2} \oplus (c_{i-3} \lll 31) \oplus (c_{i-4} \ggg 1) \oplus W_i, \tag{8}$$

where $W_i = w_i \oplus w_{i-2} \oplus (w_{i-3} \lll 31) \oplus (w_{i-4} \ggg 1)$. The idea of [13] now suggests to detect the bias of u_n as both w_i and c_i are biased.

For the observation of the assumed bias in the sequence $\{u_n\}_{n=0}^{N-1}$ of length N the word frequency statistic

$$\bar{\chi}^2 = \sum_{a=0}^{2^{32}-1} \frac{(\bar{\mu}[a] - \lambda)}{\lambda} \tag{9}$$

is used, where $\bar{\mu}[a]$ is the number of occurrences of the 32-bit word a in the sequence $\{u_n\}$ and λ is the awaited number of occurrences of each word for a random sequence ($\lambda = 1$ for $N = 2^{32}$). The values of $\bar{\chi}^2$ are supposed to be biased for the keystream of ABC when compared to the random sequence. The error probability of distinguishing can be estimated empirically by comparing the two sets of $\bar{\chi}^2$ values, one for the ABC keystream and another for a good (pseudo)random sequence.

Our computer experiments with the reduced versions of ABC v.2 [1] modeled the same distinguishing algorithm and employed good truly random sequences. The latter were obtained from a physical source of randomness. The experiments showed that distinguishing is completely impossible with time and data complexities of about 2^m for ABC with m-bit words.

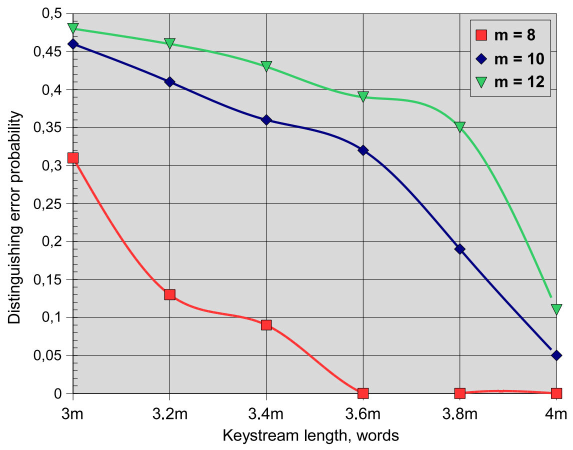

Fig. 2. Error probability for m-bit ABC distinguishers

To ensure that distinguishing attacks of this kind on ABC v.2 are impractical, extensive simulations on a high-performance computing cluster were performed. The results presented in Fig. 2 indicate that distinguishing of m-bit ABC for $m \geqslant 12$ in the way suggested in [13] with a negligible error probability cannot be carried out with time and data complexity less or equal to 2^{4m} and with memory complexity less or equal to 2^m (note that $m = 32$ for the full-size ABC v.2). Note that 2^{4m} the size of the corresponding key space. Therefore, our experiments suggest that efficiently distinguishing the full-scaled ABC v.2 from the random sequence requires more time resources than the 2^{128} brute force key search. This gives us grounds to conjecture that the application of distinguisher from [13] against ABC v.2 is nonsensical and totally impractical.

6 ABC v.2 and Stream Cipher Implementation Issues

In this section it is shown that ABC v.2 meets all the industrial implementation requirements mentioned above which make it perfectly suitable for various real-world applications including some embedded security systems.

6.1 ABC v.2 and Generic Performance

In many industrial applications it is difficult to optimize cryptographical algorithms for all concrete computer architectures. This is due to the following problems:

– High costs of assembly language implementations,
– Code portability requirement.

In practice even rather large firms cannot afford to pay for the optimized implementation of every cipher from the cryptographical library (the number of symmetrical cryptographical algorithms that are to be implemented within one library is often over 10) for every computer platform (the number of the platforms on which some consumers want to run their cryptographical libraries is often considerable and can exceed $10-20$). Such an implementation would require over $100 = 10 \cdot 10$ optimized realizations of individual cryptographical algorithms for specific computer platforms. This indicates that even inline assembly language sections can be not allowed. Another reason is that the industry wants to have the algorithms only once implemented and does not want to spend money every 6 months or 1 year (as new platforms demand immediate action rather frequently) for the same library again.

Thus, we consider the generic performance of a cipher an extremely important property for industrial applications. That is, a good stream cipher should be not only secure and at the least twice faster than AES. It should also provide the possibility of an easy and very efficient implementation in ANSI C using generic compilers (maybe with specific options).

We treat the generic implementation performance property as one of the most important stream cipher properties. For this reason we are not going to provide assembly language implementations of the ABC v.2 for the eSTREAM benchmarks since ABC v.2 holds its leading performance positions, even when implemented in ANSI C.

Here we present the results of the performance evaluation of ABC v.2. All the throughput values are gained for a generic implementation. Throughput values and costs of the setup routines for the reference implementation can be found in Table 1. The table contains results for different optimization window sizes obtained on a 3.2 GHz Intel Pentium 4 Northwood processor under the same measurement conditions as described in [7].

According to the performance benchmark tables published at the eSTREAM web site [2] ABC v.2 performance is very high. ABC v.2 is the fastest candidate at plain encryption on AMD64, PowerPC G4, and UltraSPARC-III processors,

Table 1. ABC v.2 performance for Intel Pentium 4

w	Speed, Gbps	Cycles per byte	Lookup tables, bytes	Key setup, cycles	IV setup, cycles
2	2.19	11.68	256	2056	372
4	3.36	7.65	512	4792	259
8	6.91	3.70	4096	90519	207

occupying the third place for HP9000 and second place (after Py or TRIVIUM) in the list for the rest of reported CPUs. Due to the rapid IV setup ABC v.2 is uneclipsed among Profile II (software ciphers) candidates at the encryption of short packets on all of the reported CPUs.

6.2 ABC v.2 Key and IV Setup Flexibility

ABC possesses a bundle of properties that make it fit well in various real-life applications and satisfy the industrial demands. One of these properties is the fast IV setup, which leads to the very effective packet encryption with a per-packet nonce. The ABC v.2 performance at packet encryption for various optimization window sizes (measured on an Intel Pentium M processor) is showed in Fig. 3.

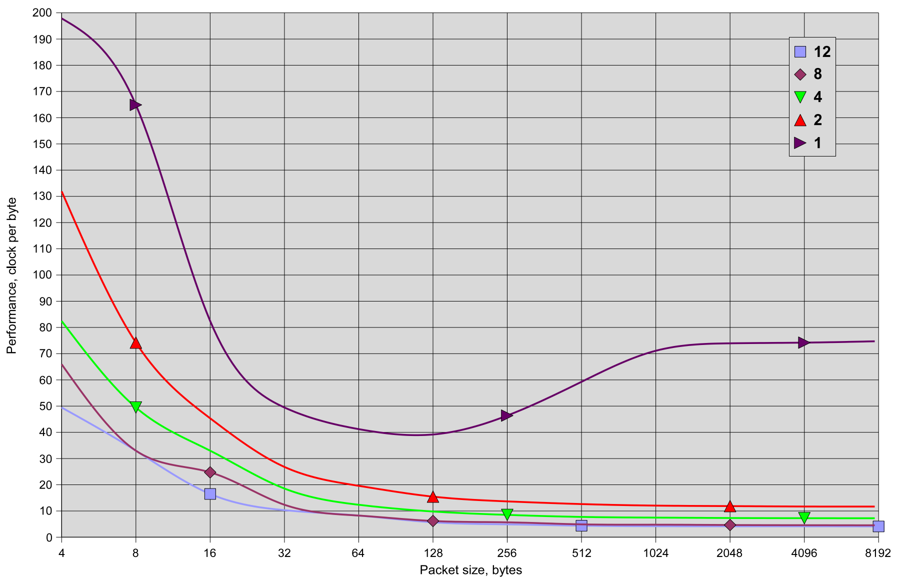

Fig. 3. ABC v.2 performance at packet encryption with IV setup

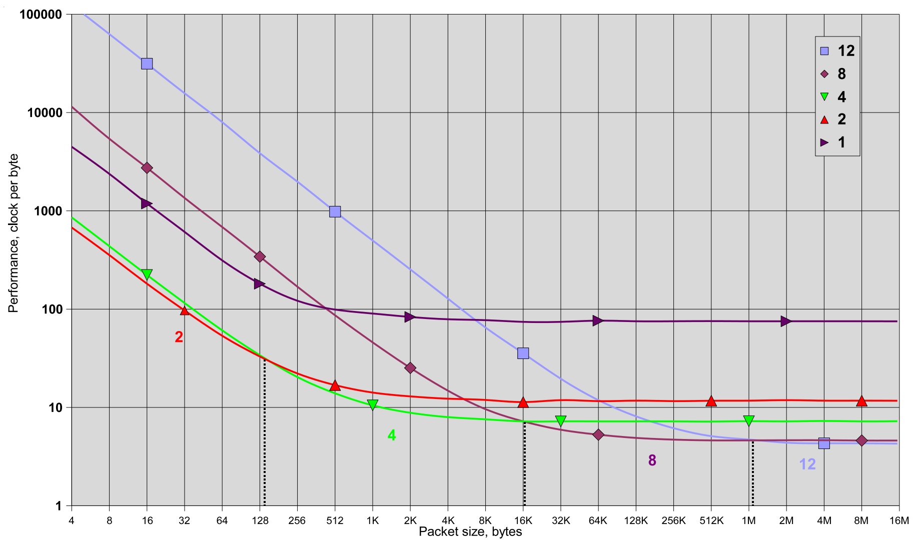

Fig. 4. ABC v.2 performance at packet encryption with key and IV setup

Another property is the natural flexibility of the ABC design, which allows one to choose the optimization parameters suitable for a given application. In practice a high demand for frequent key reinitializations means that short data segments are processed. The relatively costly key setup procedure of ABC v.2 for long optimization windows (e.g $w = 8$) is compensated by extremely high keystream generation performance of ABC v.2 with this parameter. If a less expensive key setup procedure is required, then a lower value of parameter w can be selected which results in a higher overall performance of the ABC v.2 key setup, IV setup and keystream generation routine. That is, the effect of a time-consuming key setup is easily leveled by choosing the appropriate optimization window size depending on the size of data blocks being processed. Note that the cipher remains in all the cases the same, the implementation parameters changing only. Figure 4 presents in what way the choice can be done, showing the performance of ABC v.2 at packet encryption with per-packet key and IV setup for various optimization window sizes. The relative cost of key and IV setup procedures at packet encryption is shown in Fig. 5. The measurements were performed on an Intel Pentium M processor.

6.3 ABC v.2 for Embedded Security Systems

The variable length of ABC v.2 optimization tables enables implementations with rather low memory consumption starting from 256 byte. ABC is the only cipher in the eSTREAM project providing a working 8-bit implementation [1].

The performance of ABC v.2 for a standard i8051 controller (Philips 80/87C51 microcontroller belonging to the MCS-51 family) can be found in Table 2

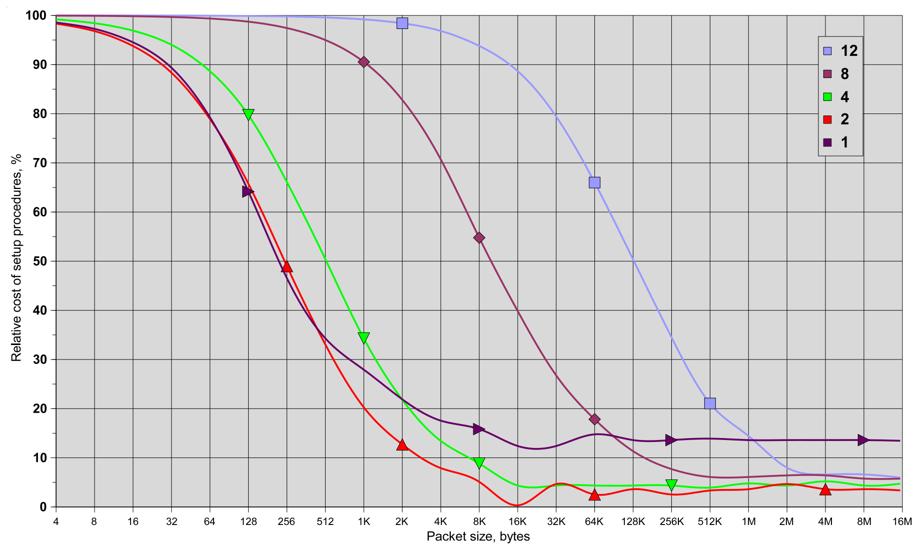

Fig. 5. Relative cost of ABC v.2 setup procedures at packet encryption

Table 2. Comparison of 8-bit ABC v.2, AES and RC6 implementations

Implementation	Code Size byte	Const byte	Ram Size IDATA+XRAM,byte	Clock cycles per byte	Key setup, clock cycles
ABC, w=2 [8]	1649	256	79+452	253	52562
ABC, w=4 [8]	1493	512	79+708	174	59854
AES [10]	760	256	65	198	
RC6 [10]	596	0	221	900	43200

which compares the implementation[3] of ABC v.2 with that for AES [10] and RC6 [11]. The performance of ABC v.2 was measured by encrypting 16-byte blocks fitting in the IRAM (internal RAM) of the microcontroller.

Contemporary smart cards possess as a rule several KByte RAM (1-4 KByte XRAM) which makes the implementation of ABC v.2 with $w = 4$ rather practicable. So, being primarily a 32-bit oriented cipher, ABC v.2 also performs well on constraint platforms. This indicates that the scope of application of ABC v.2 is not restricted by 32-bit platforms. Thus, the architecture of ABC v.2 is universal with respect to software implementations on numerous platforms.

A further RISC processor which is often applied in embedded systems and was not included in the eSTREAM benchmarks (as of January 2006) is the ARM microprocessor. The performance figures of ABC v.2 for ARM7 in comparison with those for AES and RC6 [11] can be found in Table 3.

[3] The implementation and also performance figures for ARM [8] are kindly provided by S. Kumar, COSY RUB, Germany.

Table 3. Comparison of ABC v.2, AES and RC6 on ARM

Implementation	Cycles per byte
ABC, w=2 [8]	97
ABC, w=4 [8]	55
ABC, w=8 [8]	35
AES [11]	91
RC6 [11]	49

7 Conclusion

In this paper we have presented ABC v.2 – a tweaked modification of the original ABC stream cipher. The tweaks increase the period of the ABC stream cipher in a simple way and also enlarge it's secret state. They totally eliminate the attacks described in [9] and [12]. ABC v.2 performance evaluation showed that the tweaks do not lead to significant overhead and that ABC v.2 is extremely fast in software often heading the eSTREAM benchmark list [2].

Also another way of thwarting these attacks was studied. It was explained why we preferred the way of [6]. It was also noted that the results stated in [13] are erroneous. Our computer experiments indicate that distinguishing ABC v.2 from the random sequence as suggested seems unreasonable as compared to the exhaustive key search.

The natural scalability of the ABC design was emphasized. It was shown that a version of ABC with a 256-bit key and a 256-bit IV offering 256-bit security can be made out of ABC v.2 at a very low performance cost.

Apart from its leading positions concerning software performance ABC v.2 meets a number of industrial software implementation properties such as generic performance property, flexible storage consumption and flexible cost of IV/key setup procedures. This makes ABC v.2 applicable not only on standard 32-bit platforms, but in some embedded security systems with high performance requirements as well. This was exhibited by the eSTREAM benchmarks: among eSTREAM candidates of software profile ABC v.2 is uneclipsed at such a real-life task as the encryption of short packets.

References

1. The ABC stream cipher page. `http://crypto.rsuh.ru`. 9, 12
2. eSTREAM optimized code HOWTO. `http://www.ecrypt.eu.org/stream/perf`. 1, 10, 14
3. Vladimir Anashin. Uniformly distributed sequences over p-adic integers. In I. Shparlinsky A. J. van der Poorten and H. G. Zimmer, editors, *Number theoretic and algebraic methods in computer science. Proceedings of the Int'l Conference (Moscow, June–July, 1993)*, pages 1–18. World Scientific, 1995. 7
4. Vladimir Anashin. Pseudorandom number generation by p-adic ergodic transformations, 2004. Available from `http://arXiv.org/abs/cs.CR/0401030`. 4, 6
5. Vladimir Anashin, Andrey Bogdanov, and Ilya Kizhvatov. ABC is safe and sound. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/079, 2005. `http://www.ecrypt.eu.org/stream`. 8
6. Vladimir Anashin, Andrey Bogdanov, and Ilya Kizhvatov. Increasing the ABC stream cipher period. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/050, 2005. `http://www.ecrypt.eu.org/stream`. 2, 14
7. Vladimir Anashin, Andrey Bogdanov, Ilya Kizhvatov, and Sandeep Kumar. ABC: A new fast flexible stream cipher. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/001, 2005. `http://www.ecrypt.eu.org/stream`. 1, 2, 7, 10
8. Vladimir Anashin, Andrey Bogdanov, Ilya Kizhvatov, and Sandeep Kumar. ABC: A new fast flexible stream cipher. Version 2, 2005. `http://crypto.rsuh.ru/papers/abc-spec-v2.pdf`. 1, 2, 4, 6, 7, 8, 13, 14
9. Côme Berbain and Henry Gilbert. Cryptanalysis of ABC. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/048, 2005. `http://www.ecrypt.eu.org/stream`. 2, 5, 6, 7, 14
10. Joan Daemen and Vincent Rijmen. The Rijndael block cipher. NIST, 1999. `http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf`. 13
11. G. Hachez, F. Koeune, and J. Quisquater. cAESar results: Implementation of four AES candidates on two smart cards. In *Second Advanced Encryption Standard Candidate Conference*, pages 95–108, 1999. 13, 14
12. Shahram Khazaei. Divide and conquer attack on ABC stream cipher. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/052, 2005. `http://www.ecrypt.eu.org/stream`. 1, 5, 6, 7, 14
13. Shahram Khazaei and Mohammad Kiaei. Distinguishing attack on the ABC v.1 and v.2. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/061, 2005. `http://www.ecrypt.eu.org/stream`. 1, 8, 9, 14
14. Alexander Klimov and Adi Shamir. A new class of invertible mappings. In B.S.Kaliski Jr.et al., editor, *Cryptographic Hardware and Embedded Systems 2002*, volume 2523 of *Lect. Notes in Comp. Sci*, pages 470–483. Springer-Verlag, 2003. 7
15. Donald Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, third edition, 1998. 7