

Elastic Block Ciphers in Practice: Constructions and Modes of Encryption

Debra L. Cook^{*}, Moti Yung^{**}, Angelos D. Keromytis^{**}

^{*} dcook@cs.columbia.edu¹

^{**}Department of Computer Science, Columbia University, New York, NY,
USA

{moti,angelos}@cs.columbia.edu

We demonstrate the general applicability of the elastic block cipher method by constructing examples from existing block ciphers: AES, Camellia, MISTY1 and RC6. An elastic block cipher is a variable-length block cipher created from an existing fixed-length block cipher. The elastic version supports any block size between one and two times that of the original block size. We compare the performance of the elastic versions to that of the original versions and evaluate the elastic versions using statistical tests measuring the randomness of the ciphertext. The benefit, in terms of an increased rate of encryption, of using an elastic block cipher varies based on the specific block cipher and implementation. In most cases, there is an advantage to using an elastic block cipher to encrypt blocks that are a few bytes longer than the original block length. The statistical test results indicate no obvious flaws in the method for constructing elastic block ciphers. We also use our examples to demonstrate the concept of a generic key schedule for block ciphers. In addition, we present ideas for new modes of encryption using the elastic block cipher construction.

keywords: elastic block ciphers, block cipher constructions, modes of encryption

1 Introduction

We illustrate the method for creating elastic block ciphers with four constructions. Elastic block ciphers are variable-length block ciphers created from existing block ciphers [5]. The elastic version of a block cipher supports any block size between one and two times that of the original block

¹ This work was completed while the author was at Columbia University.

size. The method consists of a substitution-permutation network that uses the round function from the existing fixed-length block cipher. In this work, we construct elastic block ciphers from AES [13], Camellia [1], MISTY1 [8] and RC6 [17], to serve as examples of the general applicability of the method. We analyze the randomness of the cipher's output using standard statistical tests and evaluate the performance of the elastic versions. We also use our constructions to illustrate the use of a generic key schedule for block ciphers. Additionally, we propose how the method can be used to create new modes of encryption.

Our performance tests demonstrate that the benefit of using an elastic block cipher varies based on the specific block cipher and implementation. In most cases, there is an increased rate of encryption when using an elastic block cipher to encrypt blocks a few bytes longer than the original block length as opposed to padding the data to two full blocks. The statistical tests applied to the block ciphers do not prove a cipher is secure but instead serve as a sanity check to determine if there are design flaws in the cipher. The test results for the elastic versions are consistent with those of the original ciphers and indicate no obvious flaws in the method for constructing elastic block ciphers.

The remainder of this paper is organized as follows. In Section 2, we describe our four constructions, including the use of a generic key schedule. In Section 3, we propose ideas for new modes of encryption. Section 4 concludes the paper.

2 Elastic Block Cipher Examples

2.1 Overview

We briefly review our method for creating elastic block ciphers [5]. Our method converts the encryption and decryption functions of any existing block cipher, G , that accepts blocks of size b bits to a variable-length block cipher, G' , that accepts block sizes of $b+y$ bits, where $0 \leq y \leq b$. Figure 1 shows the general structure of an elastic block cipher. The round function of G' is a cycle of G , where a cycle is the sequence in which all b bits have been processed by the round function of G . For example, in AES the round function is a cycle. If G is a Feistel network, a cycle is the sequence of applying the round function of G to the left and right halves of the b bit block. In each round of G' , the leftmost b bits are processed by the round function and the rightmost y bits are omitted from the round function. Af-

terwards, the rightmost y bits are XORed with a subset of y bits from the leftmost b bits and the results swapped. What y bits are chosen from the leftmost b bits for use in the swap step may vary per round. The swap step is omitted after the last round. The number of rounds in G' is $r' = r + \lceil (ry)/b \rceil$ where r is the number of cycles in G . The elastic version also includes initial and end-of-round whitening on all $b+y$ bits, and an initial and final key-dependent permutation that processes all $b+y$ bits.

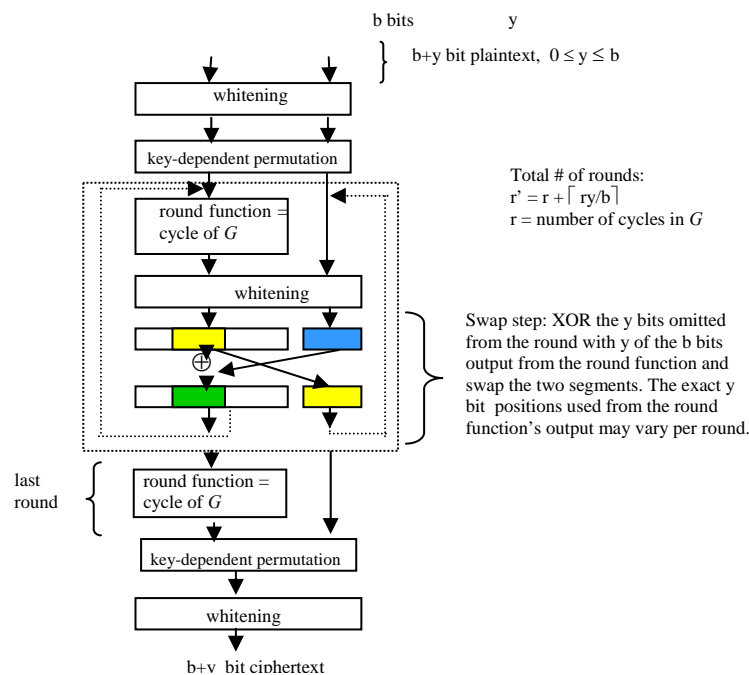


Figure 1: Elastic Block Cipher Structure

In the remainder of this section we describe the elastic versions of AES, Camellia, MISTY1 and RC6. We choose these particular block ciphers because they were finalists in standards competitions that represent different methods for how the round function process bits. AES serves as the simplest example for creating an elastic block cipher because its round function processes the entire 128-bit block in each application. Camellia, one of the recommended 128-bit block ciphers from NESSIE's competition for cryptographic algorithms [10], is a Feistel network with an additional function applied after certain cycles. MISTY1, the recommended 64-bit block cipher from NESSIE, is also structured as a Feistel network. Its elastic ver-

sion provides an example of a cipher covering blocks in the range of 64 to 128 bits. RC6, a finalist from the AES competition, breaks the data block into quarters and the round function updates two of the quarters using the values of the other two quarters. We use a 128-bit version of RC6.

2.2 Common Items

We first describe implementation details shared by the four examples. In the elastic versions of block ciphers, the bits in a block of data are numbered from the most significant (leftmost) to the least significant (rightmost). Bits 1 to b become the b -bit portion and bits $b+1$ to $b+y$ become the y -bit portion. The initial and final key-dependent permutations perform a byte or word level rotation combined with a swapping of any fractional byte of data. Two expanded-key bytes are utilized by each of the permutations. The amount of the rotation depends on an expanded-key byte. When the block size is not an integral number of bytes or words, the rightmost fractional byte or word is omitted from the rotation and swapped with bits from the rotation's result. A second expanded-key byte determines the byte or word from which bits are swapped with the fractional byte. If the block size is an integral number of bytes or words, this second expanded-key byte is unused. RC4 [18] was used for the key schedule. The first 512 bytes of RC4's output are discarded [9], then RC4 is run until the required amount of expanded key bytes are obtained. How the bits are selected for the swap steps varies slightly among our constructions. In all cases, the bits swapped out of the b -bit portion at the end of the round are y sequential bits (circling back to the leftmost bit after reaching the rightmost bit), but the starting position of this sequence varies per cipher. As shown in [4], the exact positions of the bits swapped does not matter in the sense that the elastic version will be secure against any attack that works by recovering key or round key bits if the original cipher is secure against the attack regardless of the bit positions chosen for each swap step.

For each cipher, we compared the performance of the elastic version to the original version with padding. We measured the rate of encryption for each block size that is an integral number of bytes. This excludes the time to expand the key. In the elastic implementations, when the block size is not an integral number of bytes, the fractional byte is stored in a byte and the processing time is the same as if a full byte of data is present; therefore, the time to encrypt $b+y$ bits is the time to encrypt $\lceil (b+y)/8 \rceil$ bytes. It is possible for the computational workload to vary at a more granular level, such as in a hardware implementation. The time for the fixed-length ver-

sion to encrypt a $(b+y)$ -bit block is the time to encrypt $2b$ bits in order to represent the padding required when using a b -bit block cipher. We measured the time to encrypt one million $(b+y)$ -bit blocks, where $0 \leq y \leq b$ and y is an integer multiple of 8, using the elastic version and two million b -bit blocks using the fixed-length version. The time to pad the data was not included when measuring the performance of the original cipher. We implemented all the ciphers in *C*. All tests were conducted on a 2.8Ghz Pentium 4 processor with 1GB RAM running Redhat Linux 2.4.22, unless otherwise noted.

We also compared the performance of the elastic versions to the performance of two previous proposals for variable-length block ciphers. The first proposal is by Bellare and Rogaway [2]. Their method involves running an existing block cipher, G , in CBC mode under one key, encrypting the last block of output from the CBC mode with G using a second key and using its output as an IV into G run in counter mode using third key. The ciphertext is the IV for the counter mode concatenated with the result of XORing the output from counter mode with the plaintext minus the last block. The second proposal is a modification by Patel, Ramzan and Sundarama to the first method that replaces the CBC portion with a hash function [15]. We used SHA-256 [14] as the hash function. Both proposals are less efficient than padding the plaintext to two full blocks and encrypting with a fixed-length block cipher, and both do not vary the workload for plaintext that is between one and two blocks in length. Bellare and Rogaway's method requires slightly more than twice the work of using fixed-sized, b -bit blocks for any $(b+y)$ -bit block, where $0 < y \leq b$. Patel's method requires two full applications of the block cipher plus the cost of a hash function to encrypt $b+y$ bits.

2.3 Elastic AES

We created the elastic version of AES by adding the swap step between rounds of AES, expanding AES's whitening steps (`AddRoundKey`) from $b = 128$ bits to $128+y$ bits, and adding the initial and final key-dependent permutations. The round function consists of AES's `SubBytes`, `Shiftrows` and `MixColumns` steps, with the `MixColumns` step omitted in the last round to be consistent with the fixed-length version of AES [13]. The number of rounds ranges from 10 when $y = 0$ to 20 when $116 \leq y \leq 128$. We implemented the swap step by selecting y sequential bits from the leftmost b bits, wrapping around from the right to the left as needed. The starting position is varied by moving one byte to the right each round to

avoid using the same bit positions in each swap. This avoids any complex selection process for choosing the y bits that would decrease performance.

We implemented two elastic versions of AES that differed in how the round function was implemented. In Version I, we implemented the round function as a straightforward sequence of the SubBytes, Shiftrows and MixColumns steps as defined in [13]. In Version II, we combined these steps into a table lookup. This results in the round function being a series of byte-level table lookups and XORs. Version II requires fewer CPU cycles than Version I, at the cost of an increase in memory usage. The round function can also be implemented to process the data as 32-bit words, in which case the table entries are 32-bit words. We kept table lookups at the byte level because we chose to implement the key-dependent permutations and swap step at the byte level.

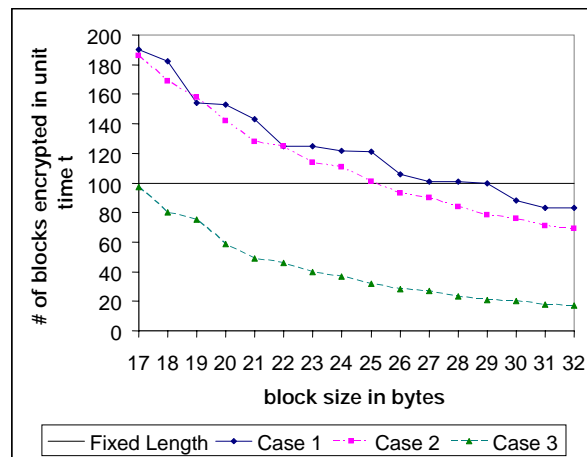


Figure 2: Normalized # of Blocks Encrypted by Elastic AES in Unit Time (Regular AES = 100)}

The elastic versions increase the number of operations beyond the 128-bit versions due to the swap steps, the two key-dependent permutations and the expansion of whitening to cover $128+y$ bits. In Version I, the elastic version saves processing time over padding. Obviously, as the block size approaches two full blocks, 20 rounds of AES are incurred in the elastic version along with the added steps, which increases the number of operations beyond the 20 rounds of AES that are required when padding the data to two full blocks. Therefore, it is expected that there is no performance benefit when encrypting blocks just under 32 bytes. In Version II, the

elastic version does not offer a performance benefit compared to padding. This is because of the simplistic nature of the operations involved (table lookups and XORs) for the round function. Even though there are fewer rounds in the elastic version than with padding, the operations for the swap step and the two key-dependent permutations consume any savings gained from having fewer rounds. However, Version II offers a performance benefit over the variable-length block cipher construction by Bellare and Rogaway, and its modification by Patel, *et al.*

Figure 2 summarizes the results from the following three cases: Case 1: Version I tested on a 1.3 Ghz Pentium 4 processor with 512MB RAM running Windows XP, Case 2: Version I tested in the Linux environment described in Section 2.2. Case 3: Version II tested in the Linux environment described in Section 2.2. In the first trial, the number of $(b+y)$ -bit blocks the elastic version can encrypt per second ranges from 190% of the number of $2b$ -bit blocks AES can encrypt per second when $y=1$ to 100% when $y = 97$. Then the elastic version's performance decreased gradually to a low of 83% of AES's rate. In the second trial, the values ranged from 186% to 69% of AES's rate, with the elastic version becoming slower than the fixed-length version when $y = 73$. In the third trial, the elastic version was slower than the fixed-sized version with padding for all block sizes.

We compared Bellare and Rogaway's method and Patel's method to AES with padding on the Pentium 4 processor used in cases 2 and 3. Bellare and Rogaway's method encrypted between 49 and 50 $(b+y)$ -bit blocks in the same amount of time AES with padding encrypted 100 blocks, for both Version I and II of AES. Patel's method encrypted 96 $(b+y)$ -bit blocks in the time it took Version I of AES to encrypt 100 blocks, and encrypted 18 $(b+y)$ -bit blocks in the time it took Version II of AES to encrypt 100 blocks. When using Version I, elastic AES is computationally more efficient than both Bellare and Rogaway's method and Patel's method for all block sizes. When using Version II, elastic AES is computationally more efficient than Bellare and Rogaway's method for block sizes up to 21 bytes in length, and is more efficient than Patel's method for block sizes less than 31 bytes and is as efficient as Patel's method for block sizes between 31 and 32 bytes.

2.4 Elastic Camellia

Camellia processes 128-bit blocks and is a Feistel network with additional steps. A function, referred to as the *FL* function, is applied after every

three cycles in the Feistel network, except after the last three cycles. FL is applied to the left half and its inverse is applied to right half of the $b=128$ bits. Camellia contains initial and final whitening steps, but not end-of-round whitening. Creating the elastic version involved using a cycle from the Feistel network as the round function, expanding the two existing whitening steps to cover $128+y$ bits and adding end-of-round whitening steps to all the other rounds, and adding the same initial and final key-dependent permutations that we used in elastic AES. We apply the FL function after every three rounds, except for the last round. A round of the elastic version is shown in Figure 3. The data is processed as bytes. The swap step was implemented by altering the starting positions between the left and right halves of the b -bit portion then rotating it one byte to the right within the half. Camellia has 9 cycles. The number of rounds in the elastic version ranges from 9 when $y=0$ to 18 when $114 \leq y \leq 128$.

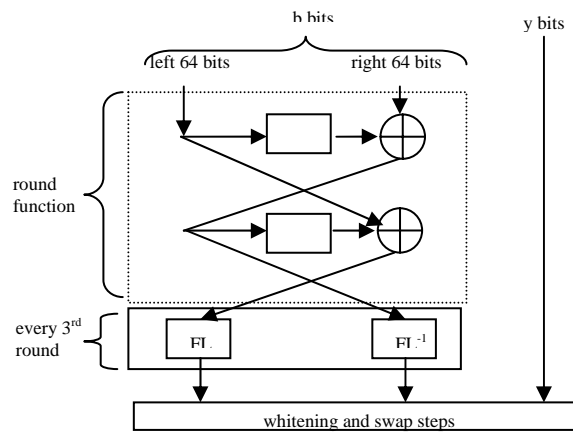


Figure 3: Round Function for Elastic Camellia

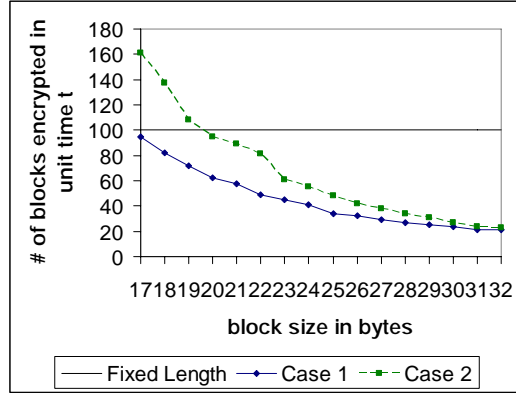


Figure 4: Normalized #of Blocks Encrypted by Elastic Camellia in Unit Time (Regular Camellia = 100)}

The elastic version offered no performance gain over the fixed-length version with padding. We also measured the performance of the elastic version without the initial and final permutations. Removing these two steps results in the elastic version offering a performance benefit when encrypting blocks that are one to three bytes over the normal 16-byte block size. Results for the following two cases are shown in Figure 4: Case 1: elastic Camellia with all steps, Case 2: elastic Camellia without the initial and final key-dependent permutations. By using a lower bound of twice the work of padding for Bellare and Rogaway's method, elastic Camellia with the key-dependent permutations provides a performance benefit for block sizes up to 22 bytes and the version without the key-dependent permutations provides a performance benefit for block sizes in the range of 9 to 25 bytes compared to Bellare and Rogaway's method. Patel's method encrypted $61(b+y)$ -bit blocks, $0 < y \leq b$, in the time it took Camellia with padding to encrypt 100 blocks. Elastic Camellia is more efficient than Patel's method for block sizes up to 21 bytes and 23 bytes, respectively, for the two cases.

2.5 Elastic MISTY1

MISTY1 is a 64-bit block cipher structured as a Feistel network with an additional function, called the *FL* function (not to be confused with the *FL* function from Camellia), applied once per cycle. While the number of cycles is not fixed, four cycles are recommended [10] and is the number upon which we base the number of rounds in the elastic version. MISTY1

does not contain whitening steps. A cycle from MISTY1 is used as the round function in the elastic version, shown in Figure 5. Creating the elastic version involved adding the whitening steps, the initial and final key-dependent permutations and the swapping of bits after each cycle. The data is processed as 32-bit words. The key-dependent permutations are of the same form (a rotation and swap) as those used in the other three elastic block cipher examples. We alternate the starting position for the swap between the left and right halves of the round's output and, within each half, rotate the starting position one word each time.

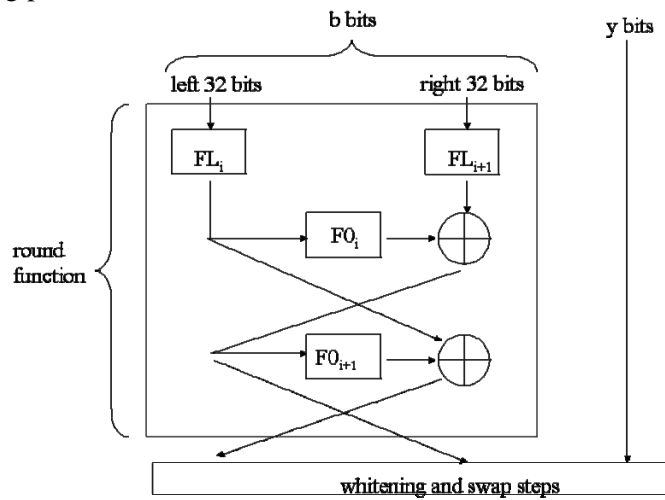


Figure 5: Round Function for Elastic MISTY1

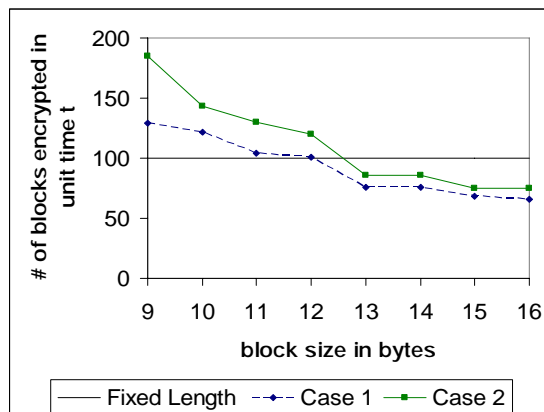


Figure 6: Normalized # of Blocks Encrypted by Elastic MISTY1 in Unit Time (Regular MISTY1 = 100)

We implemented elastic versions, with and without the key-dependent permutations, and the regular version of MISTY1. The performance results are shown in Figure 6. Case 1 refers to the version with the key-dependent permutations and Case 2 refers to the version without the key-dependent permutations. The elastic versions increased the number of operations beyond the 64-bit version of MISTY1 due to the whitening, the swap steps and, in one version, the key-dependent permutations. The elastic version of MISTY1 provides a performance benefit compared to padding for blocks that are one to four bytes over the 8-byte block size that MISTY1 processes. The benefit increases significantly in Case 2 compared to Case 1 for block sizes that are up to one additional byte over MISTY1's 8-byte block size. The performance benefit from removing the initial and final key permutations decreases as the block size increases because they represent an increasingly smaller portion of the operations as more rounds are added. In both cases, the elastic version provides a performance benefit when compared to Bellare and Rogaway's method based on a lower bound of twice the work of padding for their method. Patel's method encrypted 51 $(b+y)$ -bit blocks, $0 < y \leq b$, in the time it took MISTY1 with padding to encrypt 100 blocks using padding. Both cases of the elastic version of MISTY1 encrypt at a faster rate than Patel's method for all block sizes between 8 and 16 bytes.

2.6 Elastic RC6

RC6 is an example of a block cipher other than a Feistel network whose round function processes only a segment of the data block. RC6 divides a 128-bit data block into four 32-bit words, which we will refer to as ABCD. A and C are updated by the round function based on the values of B and D. At the end of the round, A and C have expanded-key bits added to them then all the words are rotated to the left one word. B and D have expanded-key bits added to them before the first round, and A and C have expanded-key bits added to them after the last round. The addition of expanded-key bits to a word is a type of whitening. Since this "whitening" does not cover the entire data block and is not the same as performing whitening by XORing data with expanded-key bits, we view this addition as a step in the round function and not as whitening that should be expanded to all $b+y$ bits when forming the elastic version. A sequence of four applications of the round function of RC6 is a cycle and serves as the round function in the elastic version, as shown in Figure 7. Initial and end-of-round whiten-

ing, and the initial and final key-dependent permutations are also added to create the elastic version. The rotations and XOR in the initial and final permutations were performed at the word level this time instead of at the byte level as done in elastic AES and elastic Camellia. The number of cycles in RC6 for 128-bit blocks is 5 (20 applications of RC6's round function). The number of rounds in the elastic version ranges from 5 when $y=0$ to 10 when $y=103$ (20 to 40 applications of RC6's round function). The swap step was implemented with the starting position rotating to the right one word each round.

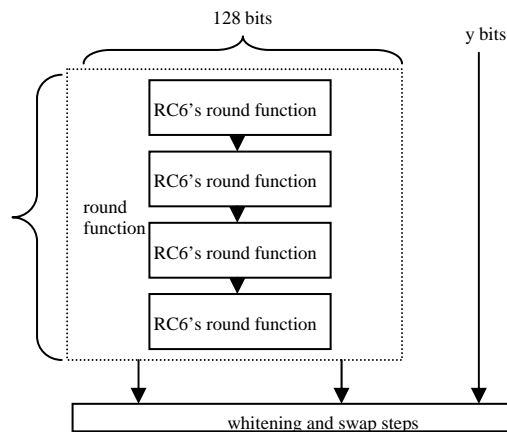


Figure 7: Round of Elastic RC6

The elastic version provides a performance benefit compared to padding for blocks of under 21 bytes in length. The results shown in Figure 8. Using a lower bound of twice the work of padding for Bellare and Rogaway's method, the elastic version of RC6 provides a performance benefit for blocks under 30 bytes in length when compared to Bellare and Rogaway's method. Patel's method encrypted 52 blocks $(b+y)$ -bit blocks, $0 < y \leq b$, in the time it took RC6 with padding to encrypt 100 blocks. Elastic RC6 is more efficient than Patel's method for block sizes up to 29 bytes.

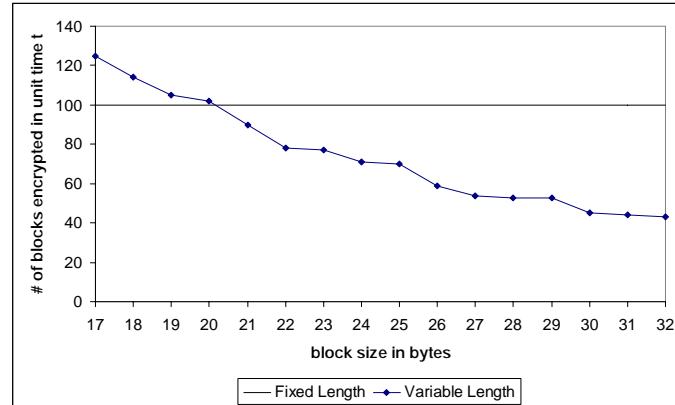


Figure 8: Normalized # of Blocks Encrypted by Elastic RC6 in Unit Time (Regular RC6 = 100)

2.7 Randomness Test Results

We applied statistical tests used by NIST on the AES candidates to both the original and elastic versions of the four ciphers. While these tests do not prove a cipher is secure, they do assist in determining if there are any obvious weaknesses with the cipher. There are sixteen tests performed on eight sets of data for each cipher. Refer to NIST's special publication 800-22 [12] for a description of the tests and to the NIST report entitled "Randomness Testing of the Advanced Encryption Standard Finalist Candidates" [11] for a description of the data sets. We tested every $(b+y)$ -bit block size where y is an integral of 8 and $b \leq b+y \leq 2b$. We also tested two block sizes that were not an integral number of bytes. These were 129-bit and 171-bit blocks for the elastic versions of AES, Camellia and RC6, and 69-bit and 75-bit blocks for the elastic version of MISTY1. We used 128-bit keys in all of our tests. Each data set required either an initial set of random plaintexts or random keys. We created these random bit strings by extracting bits from files of random bits available from random.org [16]. Based on the results, each of our three elastic block cipher examples show no signs of any statistical weakness compared to the original ciphers. In the AES competition, finalists passed each test at a rate of 96.33% or higher [11]. The elastic versions of the ciphers also met or exceeded this rate. For the elastic versions of the ciphers, the percentage of samples passing each test was consistent across all block sizes and data sets.

2.8 Key Schedules

The key schedule for an elastic version of a block cipher has to generate more expanded-key bits than the key schedule of the original block cipher. Additional key bits are needed due to the expansion or addition of whitening steps, the two key-dependent mixing steps and the increase in the number of rounds. In practice, every block cipher includes its own key schedule, which is typically designed with a focus on performance and little concern about the lack of pseudorandomness in the expanded-key bits. This tendency in key schedule design results in key schedules contributing to attacks (due to the ease in which additional key bits can be determined once a few are found and by increasing the opportunity for related key attacks [3]) and forces applications supporting multiple block ciphers to support a separate key schedule for each cipher. When creating elastic block ciphers, we wanted to avoid these disadvantages of existing key schedules. Furthermore, unlike the encryption algorithms of block ciphers which follow a somewhat generic structure by being a series of rounds, key schedules vary extensively in their structures. This makes it unlikely a general method can be devised for modifying the key schedules to generate additional bits as needed based on the block size. Therefore, we required a generic key schedule that is independent of the block cipher and that generates as many pseudorandom expanded-key bits (or close to pseudorandom) as needed while adhering to a performance bound. Existing stream ciphers are potential candidates for satisfying these requirements. We used RC4 as the key schedule in the elastic block ciphers to illustrate the concept of a generic key schedule satisfying these requirements. The first 512 bytes of RC4's output are discarded due to a slight statistical weakness in the initial bytes output from RC4 [9]. We re-initialized RC4's "S" array for each expanded key. A disadvantage of a generic key schedule is that if a weakness is discovered in the key schedule, it will impact any block cipher using the key schedule. However, having one key schedule decreases the likeliness of overlooked design flaws and implementation errors compared to when multiple key schedules are required.

In contrast to RC4 and any other stream cipher used in practice, the key schedules of AES and Camellia generate expanded keys that can easily be distinguished from random bits. In AES, an expanded-key byte is a combination of two other expanded-key bytes. When designing AES, Daemen and Rijmen noted the benefit of pseudorandom key bits, but stated that they took a "less ambitious" approach focused on avoiding symmetry between rounds and attacks due to related keys because "All other attacks are

supposed to be prevented by the rounds of the block cipher." [6], page 77. In Camellia, there is a large overlap amongst the round keys. In MISTY1, the same expanded key bits are used in multiple locations within the block cipher. In RC6, it is more difficult to determine key bits from other expanded-key bits compared to AES and Camellia. Each original key byte is altered with an addition and a rotation. The resulting byte is then added to a previous expanded-key byte and a constant to create the next expanded-key byte.

Cipher	Block Size in Bytes	# of Rounds	# of Expanded-Key Bytes
AES	16	10	180
AES	17	11	208
AES	32	20	676
Camellia	16	9	340
Camellia	17	10	383
Camellia	32	18	980
MISTY1	8	4	196
MISTY1	9	5	246
MISTY1	16	8	444
RC6	16	20	516
RC6	17	21	562
RC6	32	40	1652

Table 1: Number of Expanded Key Bytes in Elastic Versions

We compared the performance of RC4 when generating enough expanded key bits to encrypt a b -bit block to the performance of the four ciphers' key schedules. When encrypting b bits, the number of expanded-key bits in an elastic block cipher is 32 more than the number in the original cipher (due to the key-dependent permutations) plus the number of bits needed for any initial and/or end-of-round whitening that was not in the original cipher. Recall that whitening steps were added when forming the elastic versions of Camellia and RC6; whereas, AES already contained whitening and only required that its whitening steps be expanded to cover all $b+y$ bits.

When measuring the performance of the original key schedules, we removed any statements from the original ciphers' key schedules that were present only for the support of key sizes other than 128 bits. Specifically, we removed the statements from AES's key schedules that were for the support of 192 and 256-bit keys. We also compared each elastic block cipher's key expansion rate to that of AES's original key schedule because

in practice AES's key expansion rate is presently accepted. Let t_i , for $i = 1, 2, 3, 4$, correspond to the key expansion rate for the fixed-length versions of AES, Camellia, MISTY1 and RC6, respectively. Table 1 shows the number of expanded-key bytes needed in the elastic block ciphers for block sizes of b , $b+8$ and $2b$ bits. The key-expansion rates for the elastic versions compared to that of the original versions are shown in Table 2.

Elastic Cipher	Block Size in Bytes	Elastic Version's Rate (RC4) vs Fixed-Length Version's Rate	Elastic Version's Rate (RC4) vs Fixed-Length AES's Rate
AES	16	$5.94t_1$	$5.94t_1$
Camellia	16	$43.54t_2$	$6.89t_1$
MISTY1	8	$119.24t_3$	$6.09t_1$
RC6	16	$6.29t_4$	$7.84t_1$

Table 2: Key Expansion Rate

We note that Camellia and MISTY1 have the fastest key schedule of the four ciphers and also requires the most expanded-key bits, thus resulting in RC4 appearing to be significantly slower. However, Camellia's and MISTY1's key schedules have the least amount of randomness of the four ciphers due to reusing expanded-key bits in multiple locations. Overall, the RC4-based key expansion used in the elastic ciphers when encrypting b -bit blocks is just under six to just under eight times the rate of AES's key schedule.

3 Modes of Encryption

3.1 Overview

An elastic block cipher can be used in existing modes of encryption in two ways. The first option is to use the block size of the original, fixed-length block cipher for all blocks except the last block, then use a variable-length block at the end to avoid padding. A second option is to use a block size different from the fixed-length block cipher for all blocks, with the size of the last block set to avoid padding. When using an existing mode, the only benefit the elastic version of a cipher provides is the elimination of padding; it does not eliminate any existing attack against the mode. For short segments of data between one and two blocks, an elastic block cipher allows all of the bits to be encrypted as a single block, avoiding the need to use a mode of encryption and creating a stronger binding across the cipher-

text bits compared to the ciphertext produced by a mode of encryption. Elastic block ciphers also allow for new modes of encryption. We provide a sketch of two new modes, Elastic Chaining and Elastic Electronic Code Book (Elastic ECB). Both modes are intended as initial ideas for future work.

3.2 Elastic Chaining Mode

Elastic Chaining is depicted in Figure 9. y bits from the i^{th} ciphertext block are prepended to the $(i+1)^{\text{st}}$ plaintext block and the result encrypted as a $(b+y)$ -bit block. This concatenation creates a stronger binding between the i^{th} and $(i+1)^{\text{st}}$ blocks compared to that created by the XOR used in CBC mode. The stronger binding is achieved by increasing the work per block from the number of rounds required for b bits to the number required for $b+y$ bits, while the number of blocks is unchanged. The output consists of the leftmost b bits from each ciphertext block for all but the last block and the entire ciphertext of the last block. The first block to be encrypted can consist of b plaintext bits with a y -bit IV prepended to it, $b+y$ plaintext bits, or contain only b plaintext bits. Overall, the ciphertext will be at most y bits longer than the plaintext. If the plaintext is not an integral number of b -bit blocks, the last block may be shorter than $b+y$ bits. When the plaintext is not an integral number of b -bit blocks, the mode can be implemented without padding the last block; whereas, using the non-elastic version of the block cipher would require padding and also produce a ciphertext longer than the plaintext. The performance of the mode depends on the size of y . For a block cipher with r rounds, nr rounds are computed to encrypt n b -bit blocks with ECB, CBC or CTR mode. The number of rounds using elastic chaining will range from $n(r+1)$ when $y=1$ to $2nr$ when $\lceil (ry)/b \rceil = r$. This mode is useful in applications where the decryption can start at the last block. For example, when decrypting a file or segments of a database.

The ciphertext can be decrypted by decrypting the last block, concatenating the y bits from the plaintext block with the previous ciphertext block, and then decrypting the next block. When using an IV with the first block, the IV is not needed for decryption; however, having it available for decryption provides a type of integrity check in that the first y bits of the resulting plaintext can be verified against the IV.

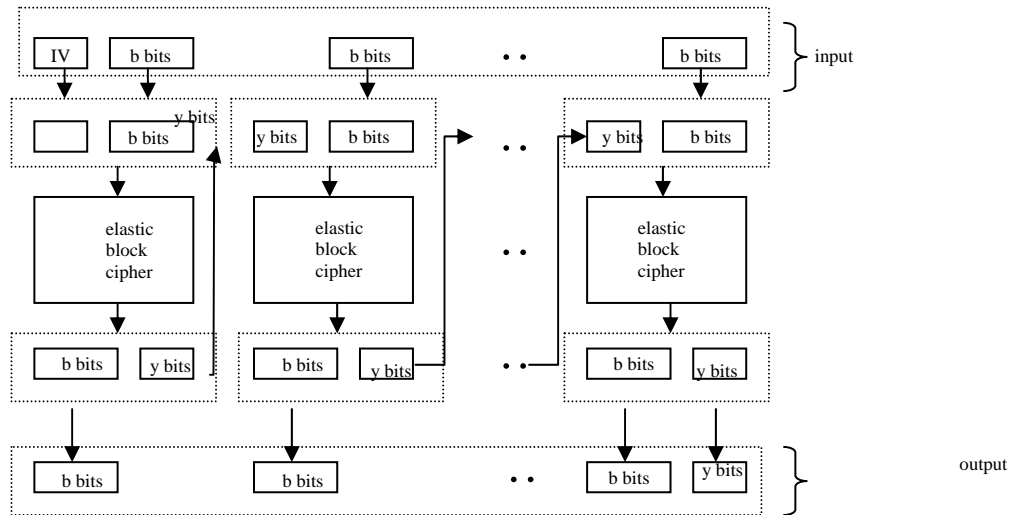


Figure 9: Elastic Chaining Mode

The mode allows for variations. These include altering which positions the y bits from the previous ciphertext block are inserted into in the current plaintext block. Instead of prepending the y bits to the next plaintext block, they could be appended or inserted amongst the b bits as either y consecutive or nonconsecutive bits. The size of y can also vary between blocks, possibly based on the key value.

This mode offers several security benefits because, even if the plaintext is known, an attacker does not know the actual $(b+y)$ -bit block being encrypted. If y varies per block based on key material, the attacker does not even know the length of each block being encrypted. Incorporating the previous ciphertext block into the current plaintext block when encrypting will hide plaintext patterns. In the way the mode is depicted in Figure 9, a single bit toggled in the ciphertext is detectable because it will garble all plaintext prior to and including the altered block. In order to insert or splice together ciphertext blocks, the inserted ciphertext block must decrypt to a plaintext which produces the same leftmost y bits as the original ciphertext block; otherwise, all plaintext blocks prior to this one will be garbled, resulting in a much more noticeable impact than the single garbled block produced by a splicing attack on CBC. Block-wise adaptive attacks [7], to which CBC is subject, are prevented because there is no need for the device performing the encryption to output the last y bits of each

ciphertext block, except for the last block. This prevents the attacker from knowing the actual block being encrypted because the attacker only gets to choose b bits of the $b+y$ bit block and block-wise adaptive attacks depend on the attacker knowing the exact plaintext.

To prepend blocks to the ciphertext, the attacker must be able to insert a ciphertext block that, when prepended to the leftmost y bits of the original first plaintext block will decrypt to some meaningful plaintext. Since these y bits are the IV, if the IV is not secret, the attacker will know what the y bits are and needs to find b bits that can be prepended to the y bits. However, notice that the attacker does not have a library of (plaintext, ciphertext) pairs from which to search for a possible b -bit value to prepend to the IV unless the entire plaintext is one block, in which case the mode is not necessary. The attacker will not have $(b+y)$ -bit (plaintext, ciphertext) pairs from the (input, output) pairs of data encrypted with this mode because the leftmost y bits of the ciphertext are not included in the output except for the last block and the $b+y$ input to the last block is not known. Appending blocks requires that the attacker append blocks of ciphertext which decrypt to a plaintext whose leftmost y bits are the same as the last y bits of the original ciphertext. In both cases, the smaller y is, the more likely it is that the attacker can form meaningful blocks to prepend or append, since there are only 2^y values to try. If y or the bit positions used for the y bits vary per block based on the key, an attacker will need to try all values of y and possible positions for the y bits.

It is not possible to rearrange ciphertext blocks without garbling the plaintext because y bits from each plaintext block are used to decrypt the previous plaintext block. In order to swap ciphertext block i with ciphertext block j , the attacker has to find a ciphertext block in position i which, when prepended to the leftmost y bits from the $(j+1)^{st}$ plaintext block, will decrypt to a plaintext block whose leftmost y bits are the same as the y bits appended to the $(j-1)^{st}$ ciphertext block during decryption. Likewise, the j^{th} block must be such that when it is prepended to the leftmost y bits from the $(i+1)^{st}$ plaintext block, will decrypt to a plaintext block whose leftmost y bits are the same as the y bits appended to the $(i-1)^{st}$ ciphertext block during decryption. Furthermore, because the recipient of the ciphertext does not receive the rightmost y bits of each block except for the last block. The attacker does not even know all of the ciphertext bits used to decrypt a given block of plaintext when trying to determine what ciphertext blocks can be rearranged without garbling the message.

3.3 Elastic ECB Mode

Our second new mode, shown in Figure 10, is a possible alternative to ECB mode that offers some protection against pattern detection in and alterations of the ciphertext compared to ECB. In tests, Elastic ECB significantly reduced the number of patterns when encrypting data that has repeated plaintext blocks aligning on 16-byte boundaries [4]. The data is encrypted as in ECB mode, but the block size varies per block based on the key, as shown in Figure 10. The i^{th} block is of length $b + y_i$ for $0 \leq y_i \leq b$ and y_i is based on key bits. The i^{th} block can be decrypted without decrypting any other block by determining its starting position and length from the key. If the key bits are sufficiently random, y_i will be uniformly random within $[0, b]$. Another option is to use key bits to set the first block's length then set each subsequent block size based on bits from the previous ciphertext block, although this will not allow the block lengths to be set in advance.

The i^{th} block can start at any position in the range $b(i-1) + 1$ and $2b(i-1) + 1$, with an average of $(3b(i-1)+2)/2$. For a plaintext pattern to show up in the ciphertext, the starting position of the block (which is now random) and y_i would have to match the starting position of the plaintext pattern and its length in the file. This method does not work for all cases because there are $b+1$ possible block sizes (b to $2b$) if all values of y are used and $(b/8)+1$ possible block sizes if the block size must be an integral number of bytes. If the file is large enough and has a significant number of repeated entries, ciphertext repetitions will occur. The degenerate case is a file consisting entirely of the same byte value repeated, in which case there will be $b/8$ distinct ciphertext blocks if y is restricted to being a multiple of 8.

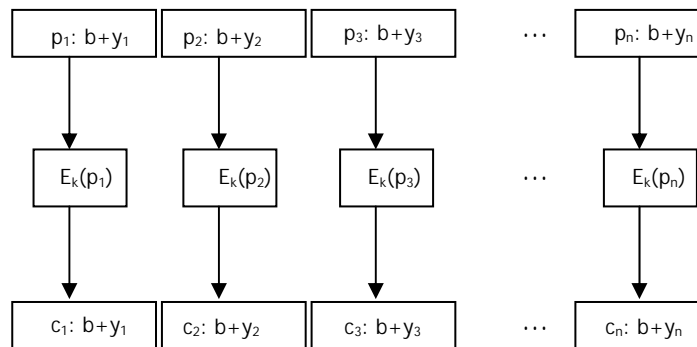


Figure 10: Elastic ECB Mode

Replacing individual blocks without garbling the plaintext is possible if the attacker can determine the start and end position of the individual blocks where the modification will occur. Any block being replaced will have to be replaced with a block of the same length; otherwise, the block and all subsequent plaintext blocks will be garbled. The probability of an attacker determining the start and end of the i^{th} block is $1/(b^2(i-1))$. (The probability of guessing the start position of the i^{th} block is $1/(b(i-1))$ and the probability of guessing the length, y_i , of the i^{th} block is $1/b$.) Splicing is even more difficult than replacing individual blocks. If two ciphertexts are being spliced together, the individual block lengths of the result must be the same as the lengths corresponding to the key. If a block is removed, the block boundaries for all subsequent blocks will not correspond to the boundaries used in encryption and the remaining plaintext will be garbled.

Elastic ECB mode is aimed at applications where at least two b -bit blocks are available when encrypting all but the last block so the block size can be varied, with y set to any value in the range of 0 to b . Elastic ECB mode may require a greater amount of computation than ECB due to the need to compute the y_i 's from the key and due to varying the block length. Overall encryption time compared to ECB may or may not increase because the longer block lengths will result in fewer blocks to encrypt. The total number of rounds required of the elastic ECB mode to encrypt nb bits, for some integer $n > 0$, will depend on the n, b and y_i values.

To illustrate how elastic ECB mode reduces patterns, the number of times two or more identical blocks occur within a file was determined when using 16-byte blocks and $(16+Y)$ -byte blocks, where Y is an integer between 0 and 16 that varies per block. We focused the tests on files where patterns are present. English text such as news articles and research papers are unlikely to have repeated phrases that align on 128-bit block boundaries [4]. In contrast, patterns are likely to appear in the ciphertext produced by ECB mode when encrypting structured files where the format of the content results in patterns, such as email logs.

We used files where repetitions of plaintext were frequent but not intentionally aligned on 16-byte boundaries. These files consisted of emails, email logs (SMTP header information) and a log of visitors (IP addresses, and related information) to a web site. The email consisted of emails between three people. The emails were generally short and included forwarded emails but no attached files or images. 160,000 bytes were used

from each file. When using elastic ECB mode, the block sizes were determined randomly using the key value as a seed to a random number generator. We ran 10 trials, each with a different key. The results are summarized in Table 3. A block counted as a match if it was identical to any previous block in the file. The maximum number of matches (greatest percent) out of the 10 trials is reported for elastic ECB mode. The number of blocks ranged from 6792 to 6845 across the combined 30 trials of elastic ECB mode on the three file types.

File Type	Percent of Blocks Counting as a Match with ECB (10,000 total blocks)	Percent of Blocks Counting as a Match with ECB (max over 10 trials).
emails	13.62%	0.85%
email log	34.38%	9.46%
web log	38.70%	7.49%

Table 3: Percent of Matching Blocks in ECB Mode vs Elastic ECB Mode

4 Conclusions

The constructions of the elastic versions of AES, Camellia, MISTY1 and RC6 illustrate how to apply the method for creating variable-length block ciphers. By applying the statistical tests used in NIST's AES competition, we conclude that there is no obvious flaw in the design because the level of randomness of the ciphertext produced by each of the elastic versions is consistent with the level required in the AES competition. The workload of the elastic version of a cipher is proportional to the block size, with the number of rounds increasing as the block size increases. The performance benefit from using the elastic version of a block cipher depends on the original cipher and the exact implementation. The percent of overhead involved in adding the swap steps, whitening and two key-dependent permutations varies based on the number of operations and exact implementation of the original cipher. For AES, whose block size is 16 bytes, there is a significant performance benefit when using the elastic version to encrypt blocks up to 25 bytes in length using an implementation of AES that requires little memory; whereas, there is no performance benefit when using a memory intensive implementation that consists entirely of table lookups and XORs. For Camellia, whose block size is 16 bytes, there is a performance benefit when using the elastic version for block sizes up to 19 bytes in length when the initial and final key-dependent permutations are not in-

cluded. For MISTY1, whose block size is 8 bytes, there is a performance benefit when using the elastic version for block sizes up to 12 bytes. For RC6 with a block size of 16 bytes, there is a performance benefit when using the elastic version for blocks up to 20 bytes in length. The elastic versions offer a performance benefit over previous methods that treat the block cipher as a black box and apply it multiple times.

The ability to encrypt variable-length blocks allows new modes of encryption to be designed. We proposed two ways of using variable-length blocks to create new modes. Elastic Chaining involves processing blocks in a manner such that bits from the i^{th} ciphertext block become part of the $(i+1)^{\text{st}}$ plaintext block. When encrypting a sequence of blocks, y bits from the previous ciphertext block are prepended to the current plaintext block to form a $(b+y)$ -bit block. This mode prevents the block-wise adaptive attacks to which CBC is subject and, compared to CBC, results in more garbled plaintext blocks when attempting to splice or otherwise alter ciphertext blocks. Elastic ECB mode is ECB mode with key bits determining each block's size such that the block size varies across the blocks. This significantly reduces the probability that patterns are detected, even in highly repetitious data. Furthermore, insertion, removal or rearrangement of blocks requires determining the start position and length of the blocks. These proposals for modes of encryption are intended as initial concepts to demonstrate additional potential uses of elastic block ciphers and require further analysis.

Acknowledgments

This work was partially supported by NSF Grants ITR CNS-04-26623 and CPA CCF-05-41093. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF or the U.S Government.

References

1. K. Aoki and T. Ichikawa and M. Kanda and M. Matsui and S. Moriai and J. Nakajima and T. Tokita. “*Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis*”. In *Proceedings of Selected Areas in Cryptography, LNCS 2012, Springer-Verlag*, pages 39-56, 2000.
2. M. Bellare and P. Rogaway. “*On the Construction of Variable Length-Input Ciphers*”. In *Proceedings of Fast Software Encryption, LNCS 1636, Springer-Verlag*, 1999.

3. M. Ciet, G. Piret, and J. Quisquater. “*Related-Key and Slide Attacks: Analysis, Connections and Improvements, Extended Abstract*”. UCL Crypto Group Technical Report, 2002.
4. D. Cook. “*Elastic Block Ciphers*”. Ph.D. Thesis, Columbia University, New York, NY, 2006.
5. D. Cook, M. Yung, and A. Keromytis. “*Elastic Block Ciphers: The Basic Design*”. In *Proceedings of ASIACCS, ACM*, pages 350-355, March 2007.
6. J. Daemen and V. Rijmen, “*The Design of Rijndael: AES the Advanced Encryption Standard*”. Springer-Verlag, Berlin, 2002.
7. A. Joux, G. Martinet, and F. Valette. “*Blockwise-Adaptive Attackers: Revisiting the (In)Security of Some Provably Secure Encryption Models*”. In *Proceedings of Advances in Cryptology - CRYPTO, LNCS 2442, Springer-Verlag*, August 2002.
8. M. Matsui, “*New Block Encryption Algorithm MISTY*”. In *Proceedings of Fast Software Encryption, LNCS 1267, Springer-Verlag*, pages 54-68, 1997.
9. I. Mironov. “*(Not So) Random Shuffles of RC4*”. In *Proceedings of Advances in Cryptology - CRYPTO, LNCS 2442, Springer-Verlag*, August 2002.
10. “*NESSIE Security Report, Version 2*”. <https://www.cosic.esat.kuleuven.ac.be/nessie>, February 2003.
11. NIST. “*Randomness Testing of the Advanced Encryption Standard Finalist Candidates*”, <http://citeseer.ist.psu.edu/soto00randomness.html>, March 2000.
12. NIST. “*A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*”. NIST Special Publication 800-22. <http://www.csrc.nist.gov/publications/nistir>, 2001.
13. NIST. “*FIPS 197 Advanced Encryption Standard (AES)*”, <http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf>, 2001.
14. NIST. “*FIPS 180-2 Secure Hash Standard*”, <http://www.csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf>, 2002.
15. S. Patel, Z. Ramzan, and G. Sundaram. “*Efficient Constructions of Variable-Input-Length Block Ciphers*”. In *Proceedings of Selected Areas in Cryptography, LNCS 3357, Springer-Verlag*, 2004.
16. random.org. <http://www.random.org/files>.
17. Rivest, Robshaw, Sidney, and Yin. “*RC6 Block Cipher*”. <http://www.rsa.security.com/rsalabs/rc6>, 1998.
18. R. Rivest. “*RC4*”. In *Applied Cryptography* by B. Schneier, John Wiley and Sons, New York, 1996.