Elastic Block Ciphers: Method, Security and Instantiations

Debra L. Cook[1], Moti Yung[2], Angelos D. Keromytis[3]

[1] Department of Computer Science, Columbia University, New York, NY, USA
dcook@cs.columbia.edu
[2] Google, Inc. and Dept. of Computer Science, Columbia University, New York, NY, USA
moti@cs.columbia.edu
[3] Department of Computer Science, Columbia University, New York, NY, USA
angelos@cs.columbia.edu

We introduce the concept of an *elastic block cipher*, which refers to stretching the supported block size of a block cipher to any length up to twice the original block size while incurring a computational workload that is proportional to the block size. Our method uses the round function of an existing block cipher as a black box and inserts it into a substitution- permutation network. Our method is designed to enable us to form a reduction between the elastic and the original versions of the cipher. Using this reduction, we prove that the elastic version of a cipher is secure against key-recovery attacks if the original cipher is secure against such attacks. We note that while reduction-based proofs of security are a cornerstone of cryptographic analysis, they are typical when complete components are used as sub-components in a larger design. We are not aware of use of such techniques in the case of concrete block cipher designs. We demonstrate the general applicability of the elastic block cipher method by constructing examples from existing block ciphers: AES, Camellia, MISTY1 and RC6. 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. We also use our examples to demonstrate the concept of a generic key schedule for block ciphers.
key words: elastic block ciphers, variable-length block ciphers, security analysis, reduction proof, key recovery attacks.

1 Introduction

Standard block ciphers are designed around one or a small number of block sizes, with most supporting 128-bit blocks. In applications, the length of the data to be encrypted is often not a multiple of the supported block size. This results in the use of plaintext-padding schemes that impose computational and space overheads by appending bits to the data. When the data being encrypted is relatively small, the padding can become a significant portion of the encrypted data. For example, encrypting a database at the field or row level to allow for efficient querying can easily result in a substantial amount of padding. When the plaintext is 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 ciphertext bits compared to the ciphertext produced by a mode of encryption, such as cipher block chaining (CBC).

This work is an extended version of our previous work on elastic block ciphers, including a more detailed explanation for the selection of the basic structure utilized when creating elastic block ciphers, extended proofs of claims and extended descriptions of instantiations of the method from [11–13].

We introduce the concept of an *elastic block cipher,* which allows us to ”stretch” the supported block size of a block cipher up to a length double the original block size, while increasing the computational workload proportionally to the block size. This, together with modes of encryption, permits block sizes to be set based on an application's requirements, allowing, for example, a non-traditional block size to be used for all blocks, or a traditional block size to be used for all but the last block in a given mode of encryption. We propose a general method for creating an elastic block cipher from an existing block cipher. Our intent is not to design a new *ad-hoc* cipher, but to systematically build upon existing block ciphers. Our method consists of a network structure that uses the round function from an existing block cipher, allowing us to treat the round function of the original cipher as a black box and reuse its properties. This results in the security of the elastic version of a cipher being directly related to that of the original cipher.

Previous proposals for converting existing block ciphers into variable-length ones focused on treating a block cipher as a black box and combining it with other operations [4, 29] in what amounts to a mode of encryption. While such an approach allows the security of the variable-length block cipher to be defined in terms of the original block cipher, the resulting constructions require multiple applications of the original block cipher, making them computationally inefficient compared to padding. These methods may be valuable in providing modes of encryption that preserve the length of the data but they do not address how to design block ciphers to support variable-length blocks. There have also been ad-hoc attempts to design a variable-length block cipher from scratch [31, 36]. Ciphertext stealing is another way of preserving the length of the data when using a mode of encryption. It involves padding the last plaintext block using ciphertext from the previous block. However, it provides no computational savings, requires altering how the mode is applied to the last two blocks and requires decrypting the last block before the next-to-last block.

We consider the security of elastic block ciphers against practical attacks. These attacks typically attempt to recover the keys or the round keys of the block cipher; differential [7, 18], linear [20] and exhaustive search methods are instances of such attacks (but other attacks exist [6, 38]). The fact that the round function of the original block cipher is used as a black box in the elastic version directs us to relate the security of the elastic version of a block cipher directly to the security of the original cipher. This is motivated by reduction-oriented proofs of security. Such proof techniques are not typical in symmetric-key cryptography, especially in concrete designs (for a survey of proof techniques in this area, see [37]:Chapter 4) and are more common in generic designs based on strong assumptions on the components (*e.g.,* a component is a random or pseudorandom function [19]).

We prove that the elastic version of a block cipher is secure against attacks that attempt to recover key bits if the original, fixed-length version of the cipher is secure against such attacks. *Our method is unique in that we show how to convert such an*

attack on the elastic version directly into an attack on the original version. As a result of our proof, if the original cipher is proven to be immune to a specific key-recovery attack then the elastic version is also immune to the attack.

We illustrate the method for creating elastic block ciphers with four constructions. We construct elastic block ciphers from AES [27], Camellia [2], MISTY1 [21] and RC6 [33] to serve as examples of the general applicability of the method. We analyze the randomness of each 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.

The remainder of the paper is organized as follows. Section 2 summarizes related work. Section 3 describes our method for constructing elastic block ciphers. Section 4 defines the relationship between the security of the elastic version of a cipher against key recovery attacks to the security original cipher against such attacks. Section 5 describes four instances of elastic block ciphers. Section 6 concludes the paper.

2 Related Work

Block ciphers are, ideally, pseudorandom permutations (PRPs), which are a subset of pseudorandom functions (PRFs). Previous work on variable-length PRFs and PRPs includes support for variable-length inputs with fixed-length outputs as applicable to MACs and hash functions [1, 3, 5, 8] and on multiples of the original block length [15, 16, 19, 28] (although the same goal is accomplished by modes of encryption, for which there are numerous examples used in practice, *e.g.*, CBC, OFB, CFB, CTR). There has also been work on using PRPs to create PRFs [17].

Three previous approaches for creating variable-length block ciphers are designing a cipher from scratch, using an existing block cipher as a black box and adding operations around it, and altering existing modes of encryption. The Hasty Pudding Cipher (HPC) [36], a submission to the AES competition that was deemed insecure in the first round [25], is an example of designing a variable-length block cipher from scratch. While creating a new block cipher from scratch allows the design to incorporate new features, such as support for a range of block sizes, it also requires analyzing the cipher against all known attacks.

A proposal by Bellare and Rogaway uses any existing block cipher as a black box to create a variable-length block cipher [4]. Their method is shown in Figure 1. Patel, *et al.*, proposed a modification to their method [29]. Bellare and Rogaway do not modify the original block cipher, but instead add operations around it. They treat the original cipher as a black box and analyze the construction independently of the specific block cipher. The security of their variable-length block cipher is defined in terms of the original cipher. Given a $(mb+y)$-bit segment of plaintext and a b-bit block cipher, for $0 < y < b$ and $m > 1$, $b - y$ bits of padding are added before the last (rightmost) b-bit block to produce $m + 1$ b-bit blocks. The data is then encrypted in CBC mode. The last block output from the CBC mode is encrypted again and the resulting b-bit output, α, is used as input to the block cipher in CTR mode. CTR mode is run until $(m - 1)b + y$ bits are output. These bits are XORed with the leftmost $(m - 1)b + y$ bits of input, to produce $u \oplus v$. The ciphertext is the concatentation of α and and $u \oplus v$. Encrypting

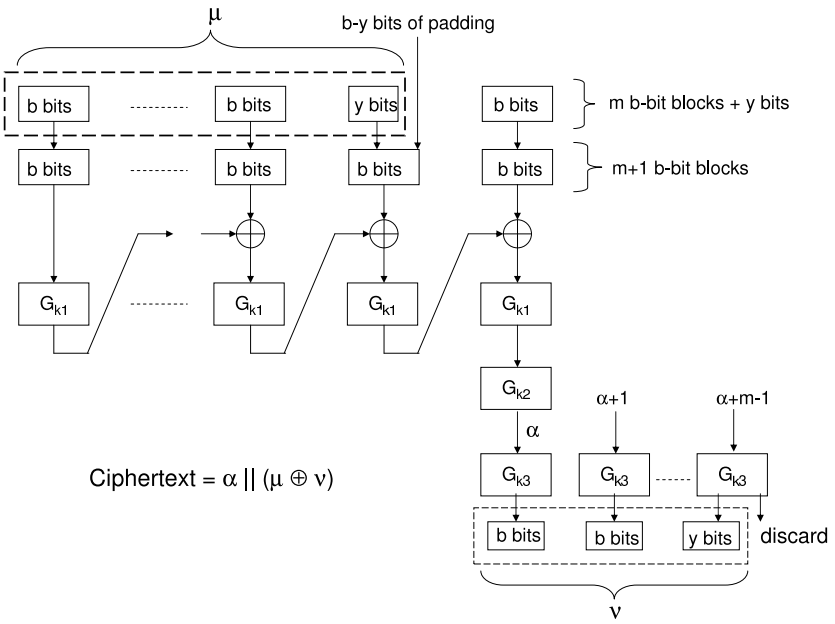

Fig. 1. Bellare and Rogaway's Variable-Length Block Cipher

one plus a fractional block, $b + y$ bits, involves four applications of the block cipher (two applications in CBC mode, one more application to obtain α and one application in CTR mode) plus additional operations, thus requiring more than four times the work of the original block cipher to encrypt one plus a fractional block regardless of the number of extra bits actually encrypted (*e.g.*, even if the data is one bit longer than the original block length). Therefore, while their approach preserves the length of the data, it requires at least twice the work of padding to encrypt one plus a fractional block.

The proposal by Patel, *et al.*, is a modification of Bellare and Rogaway's method [29]. The CBC portion is replaced by a hash function, potentially reducing the amount of computation in this component of the algorithm. Now block sizes between one and two times the original block length require one application of a hash function and two applications of the cipher instead of four applications of the cipher. This modified version also treats the original block cipher as a black box with operations added around it and is computationally less efficient than padding.

Finally, existing modes of encryption can be used to encrypt data without expanding the length of the plaintext by using what is called ciphertext stealing. This method does not provide any computational savings over padding; instead, it adds minor computational overhead. Ciphertext stealing pads the last plaintext block using some of the ciphertext from the previous block, and does not output the ciphertext bits used for the padding in order to maintain the length of the plaintext. Ciphertext stealing generally works as follows: For a block size of b bits, when encrypting $nb + y$ bits, for an integer $n \geq 1$ and integer y where $0 < y < b$, the mode of encryption proceeds as normal

through the last full block. $b - y$ bits of the last full ciphertext block are prepended to the remaining y bits of plaintext to form the $(n+1)^{st}$ block. The ciphertext consists of the output from the mode of encryption on the first $n-1$ blocks, the y bits from the n^{th} block of output that were not prepended to the remaining y bits of plaintext and the entire b bits of output from the $(n+1)^{st}$ block. When decrypting, the last block is decrypted before the next to last block, and bits from the last plaintext block appended to the next to last (the partial) ciphertext block. This requires switching the order of the last two blocks when decrypting and computing the length of the last block to determine how many bits from the plaintext must be appended to the next to last ciphertext block. The need to switch the order of the last two blocks when decrypting is a disadvantage of ciphertext stealing because changing the block order impacts the performance of high-speed hardware encryptors. Ciphertext stealing works with any block cipher without requiring modification to the block cipher, the only impact is to how a mode of encryption is applied to the last one plus fractional block of data. However, ciphertext stealing is a mode of encryption and not a block cipher design. Unlike a $(b+y)$-bit block cipher, there is no diffusion across all $(b+y)$ bits and the output can easily be distinguished from random bits. For example, encrypting two $(b+y)$-bit plaintext blocks of data in which the leftmost b bits are identical will result in the first y bits of the ciphertext being identical.

3 Elastic Block Cipher Construction

3.1 Overview

We describe our algorithm for creating elastic block ciphers and the underlying structure, the elastic network, that serves as the basis for the algorithm. The algorithm converts the encryption and decryption functions of existing block ciphers to accept blocks of size b to $2b$ bits, where b is the block size of the original block cipher. Our method uses a network structure, the elastic network, into which the round function of the original block cipher is inserted. This allows the properties of the original block cipher's round function to be reused. The elastic network creates a permutation on $b+y$ bits from a round function that processes b bits, where $0 \leq y \leq b$. We neither modify the round function of the block cipher nor decrease the number of rounds applied to each bit; instead, the method allows bits beyond the supported block size to be combined with bits in the supported block size.

Our approach in designing elastic block ciphers falls between the design from scratch and black box approaches. We treat the round function of the original block cipher as a black box instead of using the entire cipher as a black box. This is the major methodological difference between our work and the proposals by Bellare and Rogaway, and Patel, *et al*. However, we are still able to use the security properties of the original cipher to avoid having to analyze the elastic version of a cipher against all attacks. Our method results in the computational workload of elastic block ciphers being proportional to the block size, in contrast to the black box approaches.

First, we describe the elastic network and explain why we could not use an existing structure, specifically, an unbalanced Feistel network [34]. Second, we describe the

steps for converting any fixed-length block cipher to a variable-length block cipher. Four instantiations of elastic block ciphers are described in Section 5.

3.2 Elastic Network

Before introducing the elastic network, we define the following terms used in the descriptions of the elastic network and the elastic block cipher construction. To assist in understanding the need for these definitions, we point out that a round of a fixed-length b-bit block cipher takes many forms in practice and a single application of the round function often does not operate on the entire b-bit block. The most common example of a round function processing a subset of the bits is found in a block cipher built on a balanced Feistel network, where the round function operates on $\frac{b}{2}$ bits. DES [24], MISTY1 and Camellia are examples of such block ciphers. An example other than a balanced Feistel network is RC6, where the block is processed as quarters, with two quarters being updated, although differently, per each application of the round function.

Definition 1. *Bit Influence: Let $x1$ be the i^{th} bit of input to a b-bit block cipher. Let $x2$ be the j^{th} bit of output from round q of the block cipher. $x1$ influences $x2$ if changing $x1$ while holding all other $b-1$ input bits to the block cipher constant causes $x2$ to change with probability > 0 when the probability is taken over all possible values of other input bits and the key bits are held constant.*

Definition 2. *Rate of Diffusion: Let x be one bit of input to a b-bit block cipher. The rate of diffusion is defined in terms of the number of bit positions influenced by x in some number of rounds.*

For example, if there are 4 rounds in a particular 128-bit block cipher and bit 1 of the input impacts 32 bits of output from the first round, 64 from the second round, 96 from the third round and all 128 bits of output from the fourth round, this indicates the rate of diffusion for bit 1.

Definition 3. *Complete Diffusion: If every input bit to a b-bit block cipher influences the value in all b bits after q rounds, then the block cipher is said to have complete diffusion in q rounds.*

Complete diffusion does not imply security and is not the same as diffusion in an ideal block cipher where changing a single bit of input will cause each individual bit of output to change with 50% probability. In complete diffusion, the probability each individual bit of output changes must only be $> 0\%$ and may be 100%.

Definition 4. *Active Bit: A bit position input to a block cipher is called active in round j if the bit in that position is input to the round function in round j. For example, if the third bit of the block is input to the second application of the round function, it is active in the second round.*

Definition 5. *Cycle: A cycle in a fixed-length, b-bit block cipher is the sequence of steps in which all b bits have been processed by the round function. A cycle (with fixed key bits, if any) is a bijection.*

Another way of phrasing this definition is from Schneier and Kelsey: ”A cycle is the number of rounds necessary for each bit in the block to have been part of both the source and target blocks at least once”, meaning each of the b bits has been in the input and output of the round function [34, 35]. For example, in AES, the round function is a cycle. In a balanced Feistel network, a sequence of two applications of the round function, which processes $\frac{b}{2}$ bits in each application, is a cycle [34]. In RC6, the sequence of four applications of the round function is a cycle.

Definition 6. *Cycle Key: The expanded key bits used within a cycle will be referred to as the cycle key.*

Definition 7. *Round Function for an Elastic Block Cipher: A round function in the elastic version of a fixed-length, b-bit block cipher is a cycle of the b-bit block cipher. The round function of an elastic block cipher (with a fixed key bits, if any) is a bijection.*

Definition 8. *Round Key: The expanded key bits used within a round (of the original or elastic version of a cipher) will be referred to as the round key.*

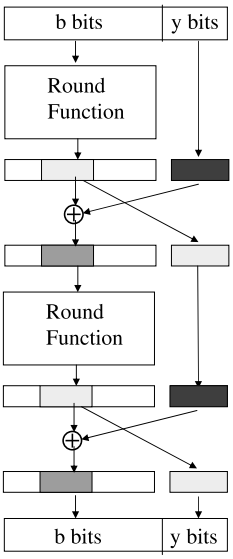

Fig. 2. Two-Round Elastic Network

The purpose for creating the elastic network is to have a structure that enables existing fixed-length block ciphers to be converted to variable-length block ciphers by adding steps between cycles of the block cipher. While we did not want to use an existing block cipher as a black box in order to gain computational efficiency compared to padding data to an integral number of blocks, we did want to use the round function of the block cipher in order to leverage its properties. Therefore, one of our goals was to

create a structure in which operations can be inserted between cycles of a block cipher that are independent of the cycle (and thus round function) of the block cipher. The properties we require of the structure are:

1. It provides a permutation on $b + y$ bits for any $0 \leq y \leq b$ where b is the block size of the fixed-length block cipher.
2. It is a single, generic, construction that can be used with any block cipher.
3. The round function of any existing b-bit block cipher becomes a component of the structure without any modifcation required to the round function.
4. The number of rounds is not set by the structure, but rather the round function can be applied as many times as needed by a specific cipher.
5. The operations involved in the structure allow for efficient implementations in terms of time and memory requirements.
6. The rate of diffusion for $b + y$ bits is defined in terms of the rate of diffusion for b bits in the fixed-length block cipher.

The elastic network satisfies these properties. A two-round version of the network is shown in Figure 2. It works by inserting the unmodified cycle of the original, b-bit block cipher into the network. To create a permutation on $b+y$ bits, b bits are input to the round function, as would normally occur in the original block cipher, and y bits (let Y denote these bits) are omitted from the round function. After the round function is applied, but before its output is given as input to the next application of the round function, y of the b bits output from the round function (let X denote these y bits) are XORed with Y, allowing Y to become part of the b bits input to the next application of the round function. X becomes the y bits omitted from the next application of the round function. Therefore, the first, second and third properties are satisfied. Any number of rounds of the elastic network can be applied, satisfying the fourth property. The operations added around the round function are simple, involving only the XOR of bits and swapping of bit segments, thus satifying the fifth property. Finally, the rate of diffusion is defined in terms of the rate of diffusion of the original cipher. Complete diffusion refers to the point at which every bit of the input to the block cipher has influenced every other bit. The elastic network requires at most one more cycle than the original block cipher to obtain complete diffusion. Therefore, the last property from our list is satisfied.

Claim 1: If complete diffusion occurs after q cycles in the original, fixed-length version of a block cipher, it occurs after at most $q + 1$ rounds in the elastic version of the block cipher.

Proof. A round in the elastic version of the block cipher uses a cycle from the original version of the cipher followed by the swapping of bits. By the end of the first round, the y bits left out of the round function have not impacted any other bits. The rate of diffusion for the b bits input to the first round function is the same as in the cycle of the original cipher. The inputs to the second through the last application of the round function are influenced by all $b + y$ bits because of the XOR in the swap step after each round. Thus beginning at the second round, all $b+y$ bits influence the input to the round function and complete diffusion will occur within q rounds, which are q cycles of the original cipher. The b bits output from the $(q+1)^{st}$ round function have been influenced by all $b + y$ bits; therefore, after the swap of bits that follows the $(q + 1)^{st}$ application

of the round function, all $b+y$ bits have influenced the leftmost b bits and the rightmost y bits resulting from the swap step. Therefore, complete diffusion occurs by the end of the $(q+1)^{st}$ round in the elastic version.

3.3 Comparison of the Elastic Network to an Unbalanced Feistel Network

The elastic network is similar to an unbalanced Feistel network. One question that arises is why an unbalanced Feistel network cannot be used instead of the elastic network? An unbalanced Feistel provides the benefit of being its own inverse, with the round keys used in reverse so the round function does not have to be invertible; whereas this is not true of the elastic network. We compare the elastic network to an unbalanced Feistel network and explain why an unbalanced Feistel network does not possess all the properties required to create a variable-length block cipher from any existing block cipher.

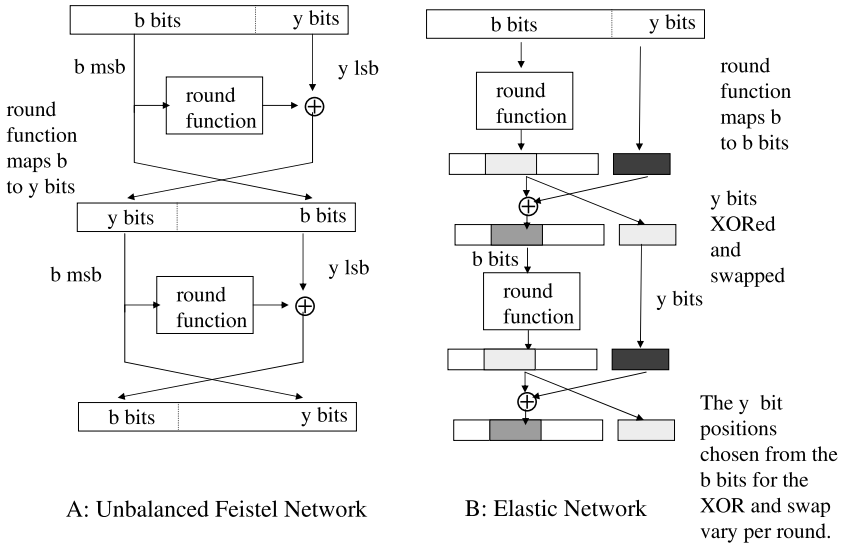

Fig. 3. Unbalanced Feistel Network Compared to Elastic Network

While the two networks may appear similar, it is not feasible to use an unbalanced Feistel network to create elastic block ciphers. Figure 3 shows the structure of an unbalanced Feistel network compared to the elastic network. In a (balanced) Feistel network, the block is split into two components of equal length; whereas, in an unbalanced Feistel network the components do not have the same length and the lengths of the round function's input differs from the length of its output. The elastic network also involves splitting the block into two components, applying the round function to one component then XORing and swapping bits between the components to form the input to the

next round. However, the elastic network differs from an unbalanced Feistel network in several ways.

1. The round function of the elastic network must be invertible; whereas, the round function of the unbalanced Feistel network does not need to be invertible. This is because the structures differ in what bits form the input to the round function.
2. In an unbalanced Feistel network the input to round i is XORed with the output of round $i + 1$ to form the input to round $i + 2$. In the elastic network, bits from the outputs of rounds i and $i + 1$ are XORed when forming the input to round $i + 2$.
3. The round function maps b bits to b bits in the elastic network and maps b bits to y bits in the unbalanced Feistel network. This alone does not prevent an unbalanced Feistel network from being used with the round function of an existing block cipher that maps b bits to b bits because y bits can be chosen from the output of the round function when $y \leq b$.
4. $y \leq b$ in the elastic network. Whereas, an unbalanced Feistel network places no restriction on the length of y in relation to b.
5. The most important difference is that the unbalanced Feistel network provides poor diffusion to the extent that, for r rounds and a $(b + y)$-bit block, $b - y(r - 1)$ bits of input appear in the output. Therefore, when encrypting data, part of the plaintext appears in the ciphertext when $y(r - 1) < b$. In contrast, the elastic network guarantees complete diffusion in at most one more round than the original cipher. Even when r is large enough to prevent input bits from appearing in the output for all y, where $0 < y \leq b$, an unbalanced Feistel network provides no guarantee on the rate of diffusion. Instead, the rate of diffusion depends on the specific round function. This is due to the second and third items.

It is the last item that prevents an unbalanced Feistel network from being used to convert existing block ciphers to variable-length block ciphers in the same manner as the elastic network. Obviously, such a cipher is insecure when $y(r - 1) < b$. Even if the number of rounds is set such that $y(r - 1) \geq b$, an unbalanced Fiestel network is not suitable for creating a variable-length block cipher by inserting the round function or cycle of an existing block cipher into the network. In order to (attempt) to use an unbalanced Feistel network to create a variable-length block cipher, the block will be divided into b-bit and y-bit portions where $y \leq b$ and y bits will be selected from the round function's output to use in the XOR. However, this can result in poor diffusion. The cycles of block ciphers used in practice do not provide complete diffusion in a single application (which is one reason for multiple cycles). We consider what happens in an unbalanced Feistel network when $y < b$ and if all input bits to the round function do not impact all output bits. If one of the b bits, let q be the position of this bit, input to the round function does not impact the bit positions that are involved in the XOR with the y bits and the bit in position q only influences bits in the rightmost y bits of the output, then the bit in position q going into round i will have no influence in the $(i + 1)^{st}$ round. In fact, the round function in an unbalanced Feistel network must be defined very carefully; otherwise, it is possible for certain bits to have no impact on the other bits over several rounds. We require a network structure that allows "plugging in" a cycle from any existing block cipher and viewing the cycle (and thus round function of the original cipher) as a black box while at the same time providing the same level of

security as the original cipher (in that the elastic block cipher is immune to any practical attack that recovers key or round-key bits to which the original cipher is immune).

3.4 Elastic Block Cipher Algorithm

The method for converting a fixed-length block cipher into an elastic block cipher involves inserting the block cipher's cycle into the elastic network. Also, we add (or expand from the original cipher) whitening steps, and we add a key-dependent permutation before the first round and after the last round. The general structure of the method is shown in Figure 4. The following notation and terms will be used in the description and analysis of the elastic block cipher:
Notation:

- G denotes any existing block cipher with a fixed-length block size that is structured as a sequence of rounds. By default, any block cipher that is not structured as a sequence of rounds is viewed as having a single round.
- r denotes the number of cycles in G.
- b denotes the block length of the input to G in bits.
- y is an integer in the range $[0, b]$.
- G' denotes the modified G with a $(b + y)$-bit input for any valid value of y. G' will be referred to as the elastic version of G.
- r' denotes the number of rounds in G'.
- The round function of G' will refer to one entire cycle of G, as defined in Section 3.2.

The process of converting a fixed-length block cipher into an elastic block cipher involves inserting the cycle of the block cipher into the elastic network, adding initial and final key-dependent permutations, adding or expanding initial and end of round whitening, and determining the number of rounds required. Given a block cipher G with a b-bit block size, the following modifications are made to G to convert it to its elastic version, G', that can process $b + y$ bits, for $0 \leq y \leq b$.

1. Set the number of rounds, r', such that each of the $b + y$ bits is input to and active in the same number of cycles in G' as each of the b bits is in G. $r' = r + \lceil \frac{ry}{b} \rceil$.
2. Apply initial and end of round whitening (XORing with expanded-key bits) to all $b + y$ bits. If G includes these whitening steps, the steps are modified to include all $b + y$ bits. If G does not have these whitening step, the steps are added when creating G'. In either case, additional bits of expanded-key material are required beyond the amount needed for G.
3. Prior to the first round and after the last round, apply a key-dependent mixing step that permutes or mixes the bits in a manner that any individual bit is not guaranteed to be in the rightmost y bits with a probability of 1. The leftmost b bits that are output from the initial mixing step are the input to the first round function. The initial mixing step is between the initial whitening and first round function. The final mixing step is after the last round function and prior to the final whitening.

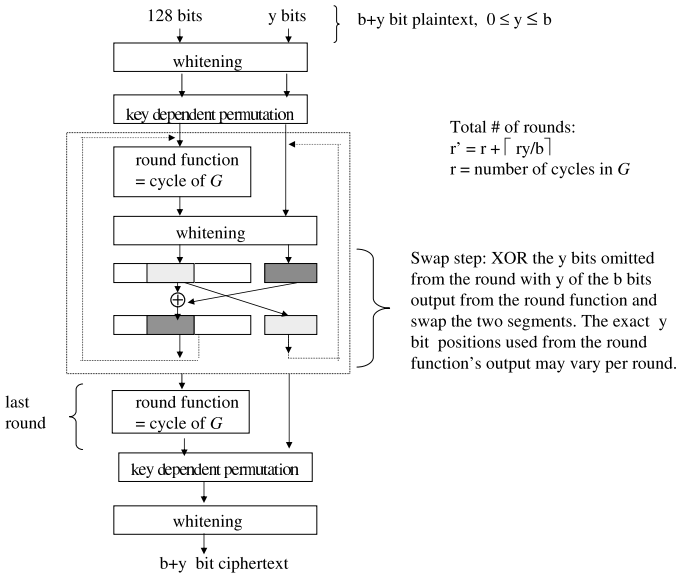

Fig. 4. Elastic Block Cipher Structure

4. Alternate which y bits are left out of the round function by XORing the y bits left out of the previous round function with y bits from the round function's output, then swap the result with the y bits left out of the previous round. This step is performed after the end of round whitening. Specifically:
   (a) Let Y denote the y bits that were left out of the round function.
   (b) Let X denote some subset of y bits from the round function's output of b bits. A different set of X bits (in terms of position) is selected in each round. How to select X is discussed in Section 3.5.
   (c) Set $Y \leftarrow X \oplus Y$.
   (d) Swap X and Y to form the input to the next round.
   This step will be referred to as "swapping" or the "swap step", and may be added to the last round if it is required that all rounds be identical. However, having the swap step after the last round does not provide additional security.

The result, G', is a permutation on $b+y$ bits. Its inverse, the decryption function, consists of the network applied in reverse and the round function replaced by its inverse.

3.5 Explanation of Algorithm

The method is designed for G' to be equivalent to G, with the possible addition of whitening and the key-dependent mixing steps, when the data is an integral number of b-bit blocks, while accommodating a range of b to $2b$-bit blocks. The construction allows the round function of G (in the form of a cycle) to be reused and thus builds

upon the round function's properties, including its differential and linear bounds. The following is an explanation of why specific steps are included in the construction.

Step 1: Each bit position of the input is required to be active in the same number of rounds in G' as the number of cycles in which it is active in G. This requirement allows the computational workload to increase proportionately to the block size while avoiding a reduced round attack on G from being applied to G'. As y increases, the number of rounds increases gradually from $r+1$ when $0 < \frac{ry}{b} \leq 1$ to $2r$ when $r-1 < \frac{ry}{b} \leq r$.

Step 2: Whitening is a useful heuristic against attacks that relate the output of a round to the input of the next round. The whitening steps assist in letting rounds work in isolation from each other in that the input to a round is unknown even when given the output of the previous round. In differential cryptanalysis, whitening does not impact the probability of a differential characteristic holding across the rounds of a block cipher because the whitening cancels with itself when computing the XOR of two inputs or outputs of a round. Linear cryptanalysis is an example of an attack whitening helps to prevent. In linear cryptanalysis, linear relationships amongst the plaintext, ciphertext and key bits are now based on the input of the i^{th} round being the output of the $(i-1)^{st}$ round $\oplus$ whitening as opposed to being equal to the output of the $(i-1)^{st}$ round.

Step 3: A key-dependent permutation or mixing of bits prior to the first round when encrypting or decrypting eliminates a one round differential that occurs with a probability of 1. This allows the first round to contribute to preventing a differential attack. The mixing step will need to take less time than a single round; otherwise, an additional round can be added instead to decrease the probability of a specific differential occurring. A trivial mixing that prevents the attacker from knowing with probability 1 which y bits are excluded from the first round is a key-dependent rotation. Refer to Section 5 for the exact steps in the permutation used in four constructions of elastic block ciphers. The key-dependent mixing steps are assumed to be designed in a sensible manner. For example, the inverse of the round function would not be used.

Step 4: $X \oplus Y$ is performed instead of merely swapping X and Y in order to increase the rate of diffusion. If G does not have complete diffusion in one cycle, then at the end of the first round of G' there is some subset S of bits output from the round that have not been impacted by some of the bits in X. While the bits in Y may impact S in the second round of G', swapping X and Y would result in the bits in X having no impact in the second round; whereas, swapping X with $X \oplus Y$ will allow the bits in X to impact the second round. Per Claim 1, complete diffusion in the elastic version of a block cipher takes at most one more cycle than in the original version. As shown in Section 4.3 when proving that there is a direct relationship between the security of G' and the security of G, the relationship is independent of the bit positions involved in the swap step. In practice when implementing elastic block ciphers, we chose to vary the bit positions selected for X to ensure that all bit positions are involved in both the b-bit and y-bit components, as opposed to always selecting the same y positions for use in X. The bit positions chosen to be swapped out after each round are a known part of the algorithm and are not determined by the key, plaintext or ciphertext.

Key Schedule: The options for a key schedule include modifying the key schedule of G to produce additional bytes, increasing the original key length and running the key schedule multiple times, or using an existing efficient stream cipher that is considered

secure in practice (this also permits the key schedule to independent of the choice of G). In all of our implementations of elastic block ciphers, the RC4 stream cipher with the first 512 bytes of output discarded is used as the key schedule. Having one standard key schedule that can output as many expanded-key bits as needed is beneficial because it means only one implementation of a key schedule is necessary regardless of the block cipher and it avoids the need to analyze one key schedule per block cipher for flaws. A stream cipher was chosen to significantly increase the randomness of the expanded-key bits over those produced by existing key schedules. This does incur a performance penalty over existing key schedules, but eliminates certain attacks which arose because of the structure of existing key schedules. Refer to Section 5 for further discussion of key schedules.

Decryption: The inverse of the round function, if it is not its own inverse, must be used for decryption. We remind the reader that the swap step is added after a complete cycle when the original cipher is a Feistel network, thus the inverse of the "round" function in the elastic version is merely running the cycle in reverse, as is normally done in any block cipher which is a Feistel network.

4 Security Analysis

4.1 Overview

For any concrete block cipher used in practice, as opposed to a pseudorandom permutation (PRP) in theory, the cipher cannot be proven secure in a theoretical sense (is not proven to be a PRP or strong PRP) but rather is proven secure against known types of attacks. Thus, we can only do the same for the elastic version of such a cipher. In order to provide a general understanding of the security of elastic block ciphers, we provide a method for reducing the security of the elastic version against key-recovery attacks to that of the original version. Our security analysis of G' exploits the fact that there is an instance of G embedded in G' and is independent of the specific block cipher used for G.

We prove that G' is secure against any attack that attempts to recover the key or the expanded-key bits if G is secure against the attack. This is accomplished by showing how to convert such an attack on G' to an attack on G. The result is independent of the method by which the attacker obtains plaintexts and/or ciphertexts to use in the attack, (*i.e.*, in general any known plaintext/ciphertext attack that recovers key bits is covered by our result.) This result is important because it implies G' does not have to be analyzed against any practical attack to which G is immune. Our approach is novel because we show how to convert an attack on the variable-length version of a block cipher directly into an attack on the fixed-length version of the block cipher. Security against key recovery attacks does not by itself imply security (*e.g.*, the identity function which ignores the key is insecure while key recovery is impossible). However, all concrete attacks against real ciphers (linear, differential, higher order differential, impossible differential, related key attacks, *etc.*) attempt key (or expanded-key) recovery and thus practical block ciphers should be secure against such attacks.

4.2 G within G'

Before stating our theorem, we provide some preliminary analysis that assists us in conveying the linkage between the original and elastic versions of a block cipher. We first draw attention to the fact that the operations performed in G' on the leftmost b-bit positions in r consecutive rounds is an application of G. This is depicted intuitively in Figure 5. We note that we are concerned only with r consecutive rounds of G' and do not include either the initial or final key dependent mixing step present in the definition of elastic block ciphers. This relationship can be used to convert an attack which finds the round keys for G' to an attack which finds the round keys for G. Let G_{rk} denote G using round keys rk. Specifically, if ${G_k}'(p \parallel x) = c \parallel z$, a set of round keys, rk, for G such that $G_{rk}(p) = c$ can be formed from the round keys and the round outputs in G' by collapsing the end-of-round whitening and swapping steps in G' into a whitening step. The leftmost b bits of the round key for the initial whitening are unchanged, and the rightmost y bits are dropped. The resulting whitening key bits will vary in up to y positions across the (plaintext, ciphertext) pairs due to the previous round's output impacting the end-of-round whitening step. However, it is possible to use these keys to solve for the round keys of G.

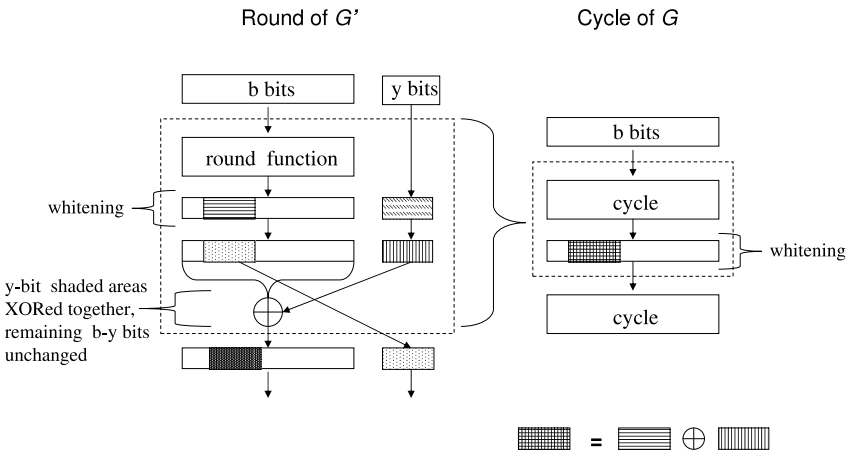

Fig. 5. G within G'

The following claim shows that for any set of (plaintext, ciphertext) pairs encrypted under sets of round keys in G' where the rightmost y bits used for whitening in each round may vary amongst the sets and all other key bits are identical amongst the sets, there exists a corresponding set of (plaintext, ciphertext) pairs for G where the round keys used in G' for the round function and the leftmost b bits of each whitening step are the same as those used in G, the plaintexts used in G are the leftmost b bits of the plaintexts used in G', and the ciphertexts for G are the leftmost b bits of output of the r^{th} round of G' prior to the swap step.

Claim 2: Let G be a b-bit block cipher and G' be its elastic version. Let $\{(pi, ci)\}$ denote a set of n (plaintext, ciphertext) pairs such that $|pi| = |ci| = b$. Let $b + y$ be the variable block size for G' where $0 \leq y \leq b$. Let w be a y-bit constant. Let vi be a y bit string that may vary per i, for $i = 1$ to n. Under the following assumptions regarding the key schedules:

- The rightmost y bits of each whitening step in G' can take on any value and are independent of any other expanded-key bits within the round and in other rounds.
- There are no message-dependent expanded keys. Any expanded-key bits utilized in G depend only on the key and do not vary across plaintext or ciphertext inputs.
- Any expanded-key bits used in the round function of the r consecutive rounds of G' can take on the same values as the expanded-key bits used in the round functions of G.
- If G contains initial and end of cycle whitening, any expanded-key bits used for the leftmost b bits of each whitening step in r consecutive rounds of G' can take on the same values as the whitening bits in G.

if $G_k(pi) = ci$ then there exists n sets of round keys for the first r rounds of G' that are consistent with inputs $pi \parallel w$ producing $ci \parallel vi$ as the output of the r^{th} round prior to the swap at the end of the r^{th} round, for $i = 1$ to n, such that the leftmost b bits used for whitening in each round are identical across the n sets and any expanded-key bits used internal to the round function are identical across the n sets.

Proof. Let $rk = \{rk_0, rk_1, ...rk_r\}$ be the set of round keys corresponding to key k for G. rk_0 denotes the key bits used for initial whitening. For each (pi, ci), form a set of the first r round keys for G' as follows: Pick a constant string, w, of y bits, such as a string of $0's$. Let $pi \parallel w$ be the input to G'. Let $rki' = \{rki'_0, rki'_1, ...rki'_r\}$ denote the round keys for G' through the r^{th} round for the pair (pi, ci). Set any bits in rki'_j used internal to the round function to be the same as the corresponding bits in rk_j. Set the leftmost b bits used for whitening in rki'_j to the b bits used for whitening in rk_j. Set the rightmost y bits used for whitening in rki'_j to be the same as the y bits left out of the round function in round j of G'. This is illustrated in Figure 6. Notice that the leftmost b bits used for whitening in each round are identical across the n sets of round keys formed, and any bits used internal to the round function are identical across the n sets; specifically, they correspond to rk in each case, and the rightmost y bits used in each whitening step differ based on (pi, ci) across the n sets. The case in which G does not contain whitening steps corresponds to using 0's for the leftmost b bits of each whitening step in G'.

The operations of G' on the leftmost b bits of rounds 1 through round r, prior to the last swap, are identical to the operations in $G_k(pi)$ because the swap step in G' results in XORing y bits of a round function's output with y $0's$. Thus, the leftmost b bits in the output of the r^{th} round prior to the swap step is ci. Therefore, for $i = 1$ to n there exists a set of round keys, rki' for $G'_{rki'}$ such that $G'(pi)$ produces ci as the leftmost b bits in the r^{th} round prior to the swap step, thus proving the claim.

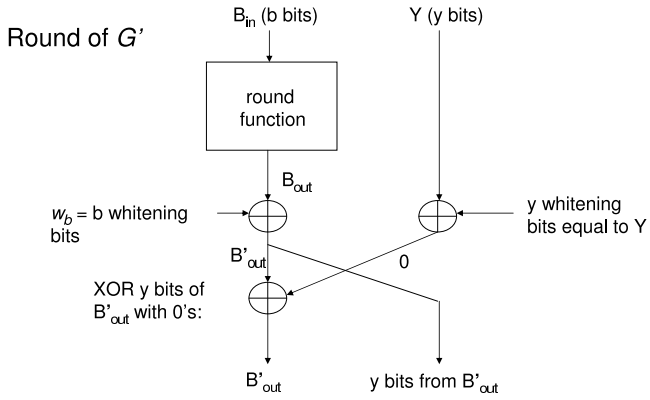

Fig. 6. Converted Key Unchanged in b Whitening Bits

4.3 Reduction Between the Original and Elastic Versions of a Cipher

We use the fact that an instance of G is embedded in G' to create a reduction from G' to G. As a result of this reduction, an attack against G' that allows an attacker to determine some of the round keys implies an attack against G that is polynomially related in resources to the attack on G'. Assuming that G itself is resistant to such attacks, we conclude that G' is also resistant to such attacks. We note that if an attack finds the key as opposed to the expanded-key bits (the round keys) then the attacker can apply the key schedule to the key to obtain the round keys. Therefore, in our analysis, we view any key recovery attack as providing the round keys to the attacker. The reduction requires a set of (plaintext, ciphertext) pairs. This is not considered a limiting factor because in most types of attacks, whether they are known plaintext, chosen plaintext, adaptive chosen plaintext, chosen ciphertext *etc.*, the attacker acquires a set of such pairs.

In our analysis, we consider G' without the initial and final key-dependent mixing steps. This allows us to focus on the core components of the elastic block cipher algorithm. If present, the mixing steps only serve to increase the security of G' since they prevent an attacker from knowing with probability one which bits are omitted from the first application of the round function when encrypting or decrypting. Furthermore, since the mixing steps are added steps (as opposed to modifications to components of G) using key material that is independent of the round and whitening key bits, they do not impact our analysis.

Theorem 1. *Given a fixed-length block cipher, G, that works on b-bit blocks and its elastic version, G', that works on $(b+y)$-bit blocks, where $0 \leq y \leq b$, if there exists an attack, $A'_{G'}$, on G' that allows the round keys to be determined for r consecutive rounds*

of G' using polynomial (in b and r) time and memory, then there exists an attack on G with r cycles that finds the expanded key for G and that uses polynomial (in b and r) many resources as $A'_{G'}$, assuming there are no message-dependent expanded key, meaning any expanded-key bits utilized in G depend only on the key and do not vary across plaintext or ciphertext inputs.

Before beginning the proof, we have a few comments on the theorem and assumptions. We first note that for an attack on G' to be computationally feasible, it must involve $< 2^b$ (plaintext, ciphertext) pairs because otherwise an exhaustive search on G would be possible, implying G is insecure against practical attacks. The assumption that the expanded key bits do not depend on the input to the cipher (the plaintext or ciphertext) is true of block ciphers used in practice and of elastic block ciphers. With no further assumptions about the key schedules, an attack that finds an expanded key for G' implies an attack that finds an expanded key for G that produces a set of $(plaintext, ciphertext)$ pairs consistent with G, but which may or may not adhere to the key schedule of G. If the expanded key is inconsistent with the key schedule of G, this itself indicates a weakness in G because it means there is some expanded key that is not produced by the key schedule of G but which produces a set of $(plaintext, ciphertext)$ pairs consistent with which G would produce when using some key generated by G's key schedule (*i.e.* the attack finds an equivalent key for the set of $(plaintext, ciphertext)$ pairs used in the attack). If the following three assumptions are placed on the expanded key bits of G', then the attack on G will find a key consistent with the key schedule of G:

- The rightmost y bits of each whitening step in G' can take on any value and are independent of any other expanded-key bits.
- Any expanded-key bits used in the round function of the first r consecutive rounds of G' can take on the same values as the expanded-key bits used in G.
- If G contains initial and end of cycle whitening, any expanded-key bits used for the leftmost b bits of each whitening step in the first r consecutive rounds of G' can take on the same values as the corresponding whitening bits in G.

These assumptions are easily satisfied in practice by using the key schedule of G to generate a subset of the round key bits and a separate algorithm to generate the expanded-key bits required in G' for the additional $r' - r$ rounds and any whitening present in G' that is not present in G. Another option is if the key schedule of G' generates pseudo-random expanded-key bits such that it is possible the expanded-key bits for the round function and leftmost b bits of whitening in r consecutive rounds can take on the same values generated by the key schedule of G. In practice, given an expanded-key, it is feasible to check if the expanded-key adheres to a specific block cipher's key schedule. A subset of the expanded-key bits being tested can be inserted into the key schedule to generate additional key bits which can be checked against the bits in the value being tested.

The theorem holds by default for the case when $y = 0$, since G' is just G (with the possible addition of whitening which can be set to 0's when applying the attack if G does not contain whitening). We view G as having whitening steps in the proof to Theorem 1. This is not an issue for the following reason. If the attack on G' involves solving

for the round key bits directly and allows the bits used in the whitening steps to be set to 0 for bit positions not swapped and to 0 or 1, as necessary, for bit positions swapped, then the whitening on the leftmost b bits is equivalent to XORing with $0,$ which is the same as having no whitening in G. Setting a subset of bits in each whitening step in G' to 0's is equivalent to using a weaker version of G'. Any attack that works on G' will work on the weaker version. This is merely the case where the attacker knows certain bits of each whitening step are 0's.

We note that Theorem 1 only states that an attack on G' can be converted to an attack on G and not the reverse. This is because, in general, the claim that an attack on G can be converted into an attack on G' does not hold. Consider the case when G contains the initial and end of round whitening steps. When $y = 0$, G' is G with the initial and final key-dependent permutations added and the key schedule replaced (such as by a stream cipher). If the attack on G is due to the original key schedule, the attack does not necessarily hold if the key schedule is changed to generate pseudorandom bits when creating G'. For any attack not due to the key schedule, in order to claim that an attack on G implies an attack on G', it is necessary that the attack on G be such that the addition of the initial and final key-dependent permuations, the addition or expansion of the whitening steps and the addition of the swap steps do not result in the attack becoming inapplicable or computationally infeasible. In general, the conversion of an attack from G' to G works because there is a decrease in the complexity of the block cipher being attacked when going from G' to G; whereas, the reverse is not true because there is an increase in the complexity of the block cipher when converting G to G'.

To prove Theorem 1, we must show for any value of y, where $0 \leq y \leq b$, that if an attack exists on G' it can be converted into an attack on G using polynomial time and memory. We define the steps for converting a round-key recovery attack on G' to an attack on G. We describe two ways of performing the conversion. The first method works for any value of y where $0 \leq y \leq b$. The second method is is applicable for values of y satisfying $r(y-2) < b$, where r is the number of rounds in the original cipher. We include the second method because it requires fewer computations than the first method and thus is useful for small values of y. The methods treat whitening key bits as if they are pseudorandom in that the whitening key bits can take on any value. In G, if there is a relationship amongst the whitening key bits and/or between whitening key bits and key material used within the cycle due to the key schedule of G, such keys will be a subset of all the possible sets of round keys found using the attack on G'. Then the set of round keys that satisfies the key schedule of G can be determined by checking which of the potential keys corresponds to the key schedule. If the number of potential sets of round keys found by the attack on G' is large enough such that it is computationally infeasible to determine which ones adhere to the key schedule of G, then the attack on G' is not computationally feasible. This is because the number of potential sets of round keys the attack finds for a set of (plaintext, ciphertext) pairs will also be large enough such that it is computationally infeasible for an attacker to determine which set to use to decrypt additional ciphertexts.

When we refer to converting the round keys of G' into cycle keys for G, we mean the following: In round j of G', let b_{jl} denote the l^{th} bit of the b bits output from the round function prior to the end of round whitening. Let kw_{jl} denote the end of round

whitening key bit applied to b_{jl}. If b_{jl} is involved in the swap step at the end of round j, let y_{jh} denote the bit from the rightmost y bits with which b_{jl} is swapped and let kw_{jh} denote the whitening key bit applied to y_{jh}. Set the l^{th} whitening bit in round j of G to $kw_{jl} \oplus kw_{jh} \oplus y_{jh}$ when $j \geq 2$. When $j = 1$, the l^{th} whitening bit is set to $kw_{1l} \oplus kw_{1h} \oplus y_{1h} \oplus kw_{0h}$ because the initial whitening is included in the conversion. Set all other key bits used in G (both whitening and any internal to the cycle) to be identical to the key bits used in G'. We refer to the initial whitening as round 0 of G' and cycle 0 of G. The initial whitening for G' is converted to initial whitening for G by using the leftmost b expanded-key bits of the initial whitening as the initial whitening in G.

Proof of Theorem I: First Method We describe here a method for converting the attack on G' to an attack on G. Without loss of generality, we use the first r rounds of G' as the r consecutive rounds for which the round keys are found. The conversion is presented in terms of solving for the round keys from the initial whitening to round r, but may also be performed by working from round r back to the initial whitening or by using any consecutive r rounds with whitening applied before the first round as long as the plaintext for G is the leftmost b bits of input to the r rounds and the corresponding ciphertext from G is the leftmost b bits of the output of the r rounds.

This attack runs in quadratic time in the number of cycles of G. The attack, $A'_{G'}$, on G' is used to solve for round keys 0 and 1 for G, then repeatedly solves for one cycle key of G at a time, using the output of one cycle of G as partial input to a reduced round version of G', running the attack on G' and converting the 1^{st} round key of G' to the cycle key for the next cycle of G. By the second condition in Theorem 1, an if an attack on G' with r' rounds exists, then a reduced round attack on G' exists for any number of rounds $< r'$.

Let P be a set of plaintexts and C be a set of ciphertexts. We use the notation $\{(P, C)\}$ to indicate a set of (plaintext,ciphertext) pairs of the form (pi, ci) with $pi \in P$ and $ci \in C$. Given a set $\{(P^*, C^*)\} = \{(pi^*, ci^*)\}$ of n (plaintext, ciphertext) pairs for G, create a set $\{(P, C)\} = \{(pi^* \parallel 0, ci^* \parallel vi_r)\}$ of n (plaintext, ciphertext) pairs for an r-round version of G'. Note: we only require that the y bits appended to each pi^* when forming $\{(P, C)\}$ be a constant; we choose to use 0. The vi_r values appended to the $ci's$ are arbitrary and do not need to be identical. The r subscript in vi_r denotes the number of rounds. Our method runs reduced round attacks on G' and the vi_r's can vary each time. Solve G' for round keys 0 and 1. Sets of round keys exist that correspond to $\{(P, C)\}$ and which are identical in at least the initial whitening and first round. The initial whitening on the leftmost b bits and any expanded key bits used by the first application of the round function in G' can be set to the same values as they would in G, and the intial whitening on the rightmost y bits of the plaintext can be set to be equal to the y plaintext bits, which are constant across all plaintexts when performing the attack. We note that the round keys across all n pairs may be identical in additional rounds, but we are only concerned with the initial whitening and first round at this point in the process. rk'_0 and rk'_1 will refer to the expanded key bits used for initial whitening and in the first round of G', respectively. Use the leftmost b bits of rk'_0 as rk_0, for G. Since the rightmost y bits are identical across all inputs to G', when rk'_1 is converted to

a cycle key for G, the result will be the same across all n elements of $\{(P, C)\}$. Use the converted round key as cycle key 1, rk_1, for G. For each pi^*, apply the initial whitening and first cycle of G using the two converted round keys. Let $p1i$ denote the output of the first cycle of G for $i = 1$ to n. Using a reduced round version of G' with $r - 1$ rounds and the initial whitening removed, set $\{(P, C)\} = \{(p1i \parallel 0, ci^* \parallel vi_{r-1})\}$ and solve for the first round key of G'. As before, convert the resulting round key for the first round to a cycle key for G. Use the converted round key as the second cycle key for G. Repeat the process for the remaining cycles of G, each time using the outputs of the last cycle of G for which the cycle key has been determined as the inputs to G' and reducing the number of rounds in G' by 1, to sequentially find the expanded-key bits for G one cycle at a time.

This attack involves applying each cycle of G to n inputs for a total of rn cycles of G. $\frac{n(r+1)r}{2}$ rounds of G' are computed in the worst case if $A'_{G'}$ requires knowing the output of each round of the reduced round version of G' to find the first round key. r applications of $A'_{G'}$ are needed on the reduced round versions of G'. Let t_A denote the time to run $A'_{G'}$. Let ks_t be the time to check that an expanded-key found by $A'_{G'}$ adheres to the key schedule of G. The time to attack G is $O(nr^2 + rt_A + ks_t)$.

In summary, the attack on G can be written as:

Input $\{(P^*, C^*)\} = \{(pi^*, ci^*)$ for $i = 1$ to $n\}$.
Create $\{(P, C)\} = \{(pi^* \parallel 0, ci^* \parallel vi_r)$ for $i = 1$ to $n\}$ for a r-round version of G', where the $vi's$ are arbitrary.
Using $A'_{G'}$, solve a r-round version of G' for rk'_0 and rk'_1.
Convert rk'_0 to rk_0 and rk'_1 to rk_1.
Set $p1i =$ first cycle's output of G using rk_0 and rk_1, for $i = 1$ to n.
For $j = 1$ to $r - 1$ {
    $\{(P, C)\} = \{(pji \parallel 0, ci^* \parallel vi_{r-j})$ for $i = 1$ to $n\}$.
    Solve a $r - j$ reduced round version of G' for the first round key, rk'_1.
    Convert rk'_1 to form rk_{j+1}.
    $p(j + 1)i =$ output of cycle $j + 1$ of G on pji using rk_{j+1} for $i = 1$ to n.
}

Proof of Theorem I: Second Method Our second method for proving Theorem 1 requires fewer computations than the first method, but provides rounds keys for a smaller set of (plaintext, ciphertext) pairs. The attack works as follows: Assume there exists a known (plaintext, ciphertext) pair attack on G' which produces the round keys either by finding the original key and then expanding it, or by finding the round keys directly. Using round keys for rounds 0 to r of G', convert the round keys into cycle keys for G one cycle at a time. For each round, extract the largest set of (plaintext, ciphertext) pairs used in the attack on G' that have the same converted cycle key. If there are n_j (plaintext, ciphertext) pairs involved at round j, there will be at least $\frac{n_j}{2^y}$ pairs remaining for which the round keys are consistent after round j. The end result is the expanded-key bits for G that are consistent with a set of $\frac{n}{2^{y(r-2)}}$ b-bit (plaintext, ciphertext) pairs for G. We then describe how to take a set of (plaintext, ciphertext) pairs for G, convert them into a set of (plaintext, ciphertext) pairs for G' in order to run the attack on G' to find the expanded-key bits for G.

Let $\{(P, C)\} = \{(pi \parallel xi, ci \parallel zi)\}$ (for $i = 1$ to n) denote a set of n known $(b+y)$-bit (plaintext, ciphertext) pairs for G', where $|pi| = |ci| = b$ and $|xi| = |zi| = y$.

Let $A_{G'}$ be an attack on G' that finds the key(s) corresponding to $\{(P, C)\}$ in time less than a exhaustive search for the key. Let m denote the number of keys found. Without loss of generality, it is assumed the keys are available in expanded form. Let k_j denote the j^{th} key found by $A_{G'}$. In practice, only one key should be found for any set of (plaintext, ciphertext) pairs.

Let $S = \{ek_j\}$ for $j = 1$ to m be the set of expanded-keys used for whitening for which ek_j is from the expansion of key k_j and $G'_{k_j}(pi \parallel xi) = ci \parallel zi$ for $i = 1$ to n.

Let R_{int} denote any key material utilized within the round function of G' and cycle of G. The values found for such key bits for the j^{th} round of G' will be the same as the key bits for the j^{th} cycle of G, for $1 \leq j \leq r$.

Let $\{(P, U)\} = \{(pi||xi, ui||vi)\}$ such that $ui||vi$ is the output of the r^{th} round of G', where $|ui| = b$ and $|vi| = y$.

Let $S' = \{ek'_j \mid ek'_j =$ bits of $ek_j \in S$ corresponding to rounds 0 to r used for whitening $\}$ be the set of expanded-key bits used for whitening in rounds 0 to r of G'.

For each $ek_j \in S'$ and each $(pi \parallel xi, ui \parallel vi) \in \{(P, U)\}$, convert the round keys to cycle keys for G. Let ek'_{ij} be the converted key corresponding to the i^{th} element of $\{(P, U)\}$ and the j^{th} element of S'. The part of ek'_{ij} corresponding to round 0 will be identical across all elements. When the round keys are converted, at most y bits change in the leftmost b bits. Thus, the resulting round keys for round q, $1 \leq q \leq r$ can be divided for each of the y impacted bits into those that have a 0 in the affected bit and those that have a 1 in the affected bit. For $q = 1$ to r, define S'_{rnd_q} as the maximum-sized set of $ek'_{ij}s$ from $S_{rnd_{q-1}}$ that have identical round key(s) for round q, where $S'_{rnd_0} = S'$. Let $\{(P, U)_{rnd_q}\}$ be the corresponding elements of $\{(P, U)\}$. When forming $\{(P, U)_{rnd_q}\}$, at least $2^{-y}|\{(P, U)_{rnd_{q-1}}\}|$ of the elements from $\{(P, U)_{rnd_{q-1}}\}$ are included.

To illustrate how the sets S'_{rnd_q} and $\{(P, U)_{rnd_q}\}$ are created, consider the example shown in Figure 7 where $b = 4$, $y = 2$, and the leftmost 2 bits are swapped with the y bits in the swap step. The round number is q and $\{(P, U)_{rnd_{q-1}}\}$ contains three (plaintext, ciphertext) pairs. Suppose the outputs of the round function in the q^{th} of G' are $100101, 110011$ and 111111 and the whitening bits in the q^{th} round are 011010. The whitening bits of the converted round keys corresponding to the three cases are $0110, 1110$ and 1110. Since 1110 occurs in the majority of the cases, set the q^{th} cycle key of G to 1110. S'_{rnd_q} contains the round keys for rounds 0 to $q-1$ from $S'_{rnd_{q-1}}$ and 0010, and $\{(P, U)_{rnd_q}\}$ contains the second and third (plaintext, ciphertext) pairs from $\{(P, U)_{rnd_{q-1}}\}$.

Let $\{(P, C)_G\} = \{(pi, ci) | (pi \parallel yi, ui \parallel vi) \in \{(P, U)_{rnd_r}\}\}$. $|\{(P, C)_G\}| \geq n/2^{yr}$. $\{(P, C)_G\}$ is a set of (plaintext, ciphertext) pairs for which $G_{rk}(pi) = ci \ \forall$ $(pi, ci) \in \{(P, C)_G\}$ where the whitening round keys of $rk \in S'_{rnd_q}$ and any additional key material utilized by the cycles is the same as that for the rounds of G', namely R_{int}.

To perform the attack on G when given a set of (plaintext, ciphertext) pairs for G, convert the pairs into a set of (plaintext, ciphertext) pairs for G' and find the round keys for G', then convert the round keys of G' to cycle keys for G as follows: Given a set $\{(P^*, C^*)\} = \{(pi^*, ci^*)\}$ for $i = 1$ to n known (plaintext, ciphertext) pairs for G,

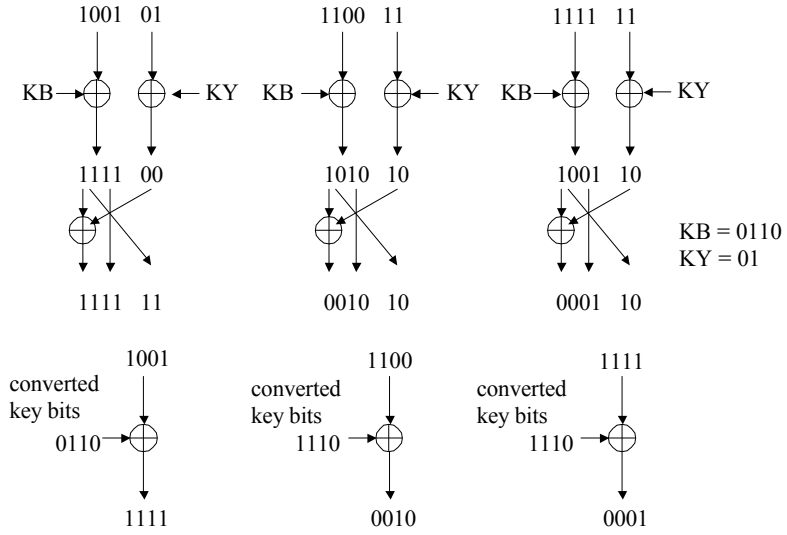

Fig. 7. Forming S'_{rnd_q}

create the set $\{(P, C)\}$ of (plaintext, ciphertext) pairs to use in the attack on an r-round version of G' by setting $pi \parallel xi = pi^* \parallel 0$ and $ci \parallel zi = (ci^* \parallel zi)$ for $i = 1$ to n. For the set of (P, C) pairs are created, $\{(P, U)\} = \{(pi^* \parallel 0, ci^* \parallel zi)\}$. Apply the attack on G' to solve for the round keys of G' then produce the sets $\{(P, U)_{rnd_r}\}$ and S_{rnd_r}. The sets of round keys in S_{rnd_r} will be consistent with the (plaintext, ciphertext) pairs in $\{(P, U)_{rnd_r}\}$.

Let t_r be the time to run r rounds of G', t_A be the time to run $A_{G'}$ and m be the number of keys (sets of round keys) found by $A'_{G'}$. In the case of obtaining at least one set $\{(P, U)_{rnd_r}\}$ of size $\geq \frac{n}{2^{yr}}$, the time required beyond t_A consists of: nmt_r time to obtain the outputs of the first r rounds for each $\{(P, U)\}$, $O(nmr)$ time to perform the conversion of the round keys from G' to cycle keys for G and $O(nmr)$ time to form the S'_{rnd_r} sets. Let ks_t be the time to check that an expanded-key adheres to the key schedule of G. Thus, the additional time required to attack G (beyond the time required to attack G'_{b+y}) is $O(nm(r + t_r) + mks_t)$. The only unknown value is m, the number of keys produced by the attack on G'_{b+1}. If m is large enough, to the extent that it approaches the average number of keys to test in a brute force attack on G', then this contradicts the assumption that an efficient attack exists on G' because the attacker is left with a large set of potential keys for decrypting additional ciphertexts.

So far we have defined a method which produces a set of at least $\frac{n}{2^{yr}}$ (plaintext, ciphertext) pairs which are consistent with the round keys. This lower bound on the number of plaintext, ciphertext pairs can be slightly increased to $\frac{n}{2^{y(r-2)}}$ by using $(b + y)$-bit plaintexts that are the same in the rightmost y bits (which we did by setting these

bits to 0), and by defining the ui values representing the ciphertext output of G in the r^{th} round of G' to be the output of the r^{th} round prior to the swapping step. This will result in $|S'_{rnd_1}| = n$ and $|S'_{rnd_r}| = |S'_{rnd_{r-1}}|$, thus in first and r^{th} rounds the set of (plaintext, ciphertext) pairs is not reduced. Then the number of (plaintext, ciphertext) pairs produced for G that are consistent with the expanded-key for G is $\geq \frac{n}{2^{y(r-2)}}$. The number of possible plaintexts for G is 2^b; therefore, it is necessary for $y(r-2) < b$ to use this method.

5 Elastic Block Cipher Examples

5.1 Overview

We describe 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 [23], 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 version 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.

5.2 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 involve a rotation of bits followed by a swapping of bits. The $b + y$ bit data is broken into segments at either the byte or 32-bit word level. The elastic versions of AES and Camellia used byte-sized segments, and the elastic versions of MISTY1 and RC6 used word-sized segments. The choice of the segment size was due to the nature of the original cipher. The operations of AES and Camellia lend themselves to byte level processing; whereas, the operations of MISTY1 and RC6 lend themselve to processing at the word level. Two expanded-key bytes are utilized by each of the permutations. The two bytes are each viewed as an integer modulo the number of full segments in the block. Refer to these two integers as $n1$ and $n2$. If the $b + y$ bits of data is not an integer number of segments, the rightmost bits are treated as a fractional segment. First, the block is rotated to the right $n1$ segments. The fractional byte or word, if any, is omitted from the rotation. The fractional component is then swapped with consecutive bits from the $n2^{th}$ segment of rotated component. If there is no fractional segment, $n2$ is unused. RC4 [32] was used for the key schedule. The first 512 bytes of RC4's output

are discarded [22], 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 Section 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 $(b + y)/8$ 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 version 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 the proposals by Bellare and Rogaway, and Patel, Ramzan and Sundarama described in 2. As explained in Section 2, ciphertext stealing requires slightly more work than plain padding. Therefore, the performance of the elastic block ciphers in comparison to the original ciphers using ciphertext stealing will be slightly better than the performance when compared to padding alone. We do not include separate measurements as a result. Furthermore, as mentioned in Section 2, unlike the other methods, ciphertext stealing is a mode of encryption as opposed to a block cipher design. It does not provide diffusion across all bits and its output can easily be distinguished from random by encrypting $(b + y)$-bit blocks with the first b bits held constant. Therefore, a comparsion between ciphertext stealing and variable length block ciphers is misleading because the later are designed to be block ciphers, with the associated security requirements such as the inability to distinguish the output from random bits, as opposed to a mode of encryption.

5.3 Elastic AES

We created the elastic version of AES by adding the swap step between rounds of AES (the round function of AES is a cycle and therefore becomes the round function in the elastic version), 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 [27]. 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, with the first swap step starting with the leftmost bit of the b-bit segment. 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 [27]. In Version II, we combined these steps into a table lookup. This results in the round function being a series of byte-level table look-ups 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.

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 its. 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 8 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 previously. Case 3: Version II tested in the Linux environment described previously. 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

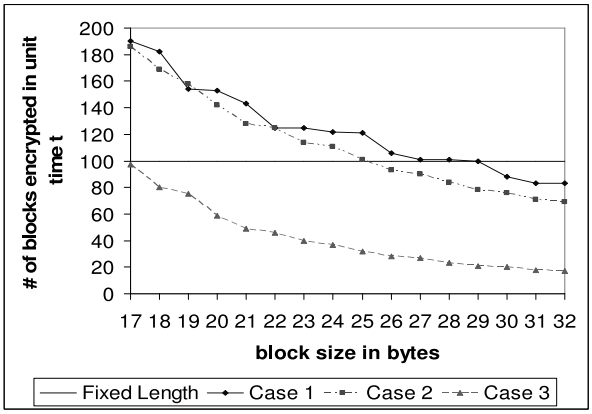

Fig. 8. Normalized # of Blocks Encrypted by Elastic AES in Unit Time (Regular AES = 100)

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.

5.4 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 9. The data is processed as bytes. The swap step was implemented by altering the starting position between the left and right halves of the b-bit portion then rotating it one byte to the right within the half, with the first swap step starting with the leftmost bit of the left half. Camellia has 9 cycles. The number of rounds in the elastic version ranges from 9 when $y = 0$ to 18 when $114 \leq y \leq 128$.

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

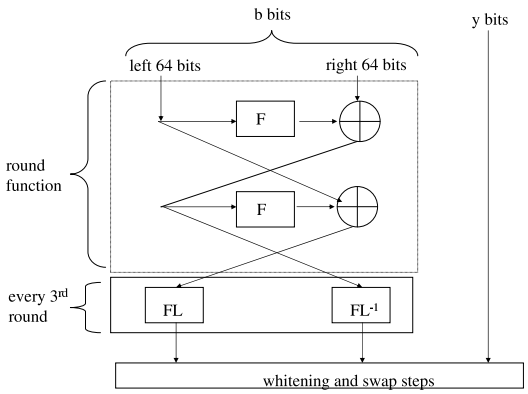

Fig. 9. Round Function for Elastic Camellia

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 10: 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, $0 < y < b$, 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.

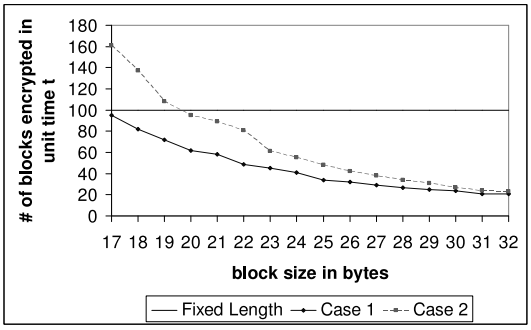

Fig. 10. Normalized # of Blocks Encrypted by Elastic Camellia in Unit Time (Regular Camellia = 100)

5.5 Elastic MISTY

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 [23] 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 11. Creating the elas-

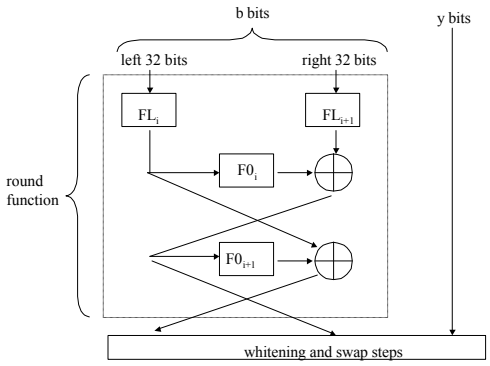

Fig. 11. Round Function for Elastic MISTY1

tic version involved adding the whitening steps, the the initial and final key-dependent permutations and the swapping of bits after each cycle. The data is processed as 32-bit words. We alternate the starting position for the swap between the left and right halves of the round function's output, with the first swap step starting with the leftmost bit of the left half.

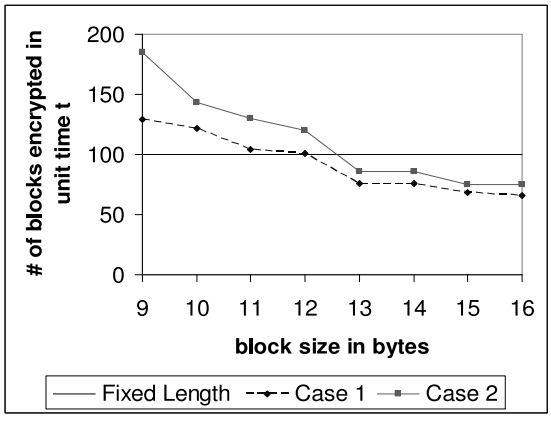

Fig. 12. Normalized # of Blocks Encrypted by Elastic MISTY1 in Unit Time (Regular MISTY1 = 100)

We implemented elastic versions, with and without the key-dependent permutations, and the regular version of MISTY1. The performance results are shown in Figure 12. 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, $0 < y < b$, 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.

5.6 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 13. Initial and end-of-round whitening, and the initial and final key-dependent permutations are also added to create the elastic version. 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, with the first swap step starting with the leftmost bit of the b-bit portion.

The elastic version provides a performance benefit compared to padding for blocks of under 21 bytes in length. The results shown in Figure 14. 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,

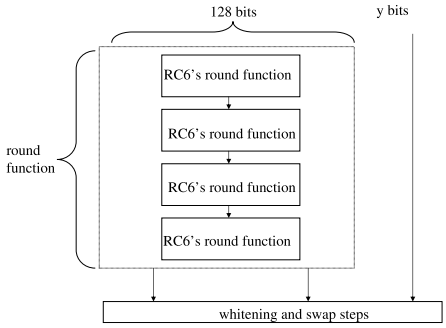

Fig. 13. Elastic RC6 Round Function

$0 < y < b$, 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.

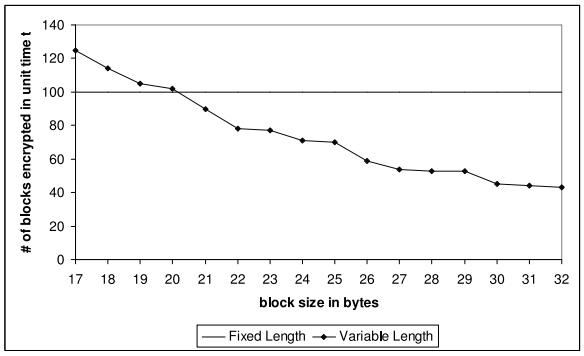

Fig. 14. Normalized # of Blocks Encrypted by Elastic RC6 in Unit Time (Regular RC6 = 100)

5.7 Randonmess 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 [26] for a description of the tests and [25] for a description of the data sets. We tested every $(b + y)$-bit block size where y is an integral of 8 and $0 \leq y \leq< b$. 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 [30]. Based on the results, each of our four 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 [25]. 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. The detailed test results are available in [10].

5.8 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 [9]) 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 [22]. 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." [14], 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.

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.

Cipher	Block Size in Bytes	# of Rounds (or Cycles)	# 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 Needed

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 in order to avoid executing unnecessary tests in conditional statements. Specifically, we removed code from AES's key schedule that was 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 ti, 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. Recall that Bellare and Rogaway's method requires 4 applications and Patel's method requires 2 applications of the original block cipher.s key schedule. The rates for the key schedule of the elastic block ciphers in relation to the rates for Bellare and Rogaway's method

and Patel's method can be estimated by dividing the values in column 3 of Table 2 by 4 and 2, respectively.

Elastic Cipher	Block Size	Rate vs Original Cipher's Rate	Rate vs 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 Rates

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.

6 Conclusions and Future Work

We have proven that the elastic version of a block cipher is secure against any practical attack that attempts to recover key or expanded-key bits if the original cipher is secure against the attack. This eliminates the need to analyze an elastic version of a block cipher against these types of attacks if the original cipher is secure against such attacks. Our result follows from the network structure used in creating elastic block ciphers and the fact that the round function (cycle) of the original fixed-length block cipher is used as a black box when forming its elastic version. We note that while reduction-based proofs of security are a cornerstone of cryptographic analysis, they are typical when complete components are used as sub-components in a bigger design. We are not aware of use of such techniques in the case of concrete block cipher designs.

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.

Unlike the encryption and decryption functions of block ciphers which can be viewed as a series of rounds, there is no general structure to key schedules in practice. In order

to avoid modifying key schedules on an individual basis when creating elastic versions of block ciphers, we propose the use of a generic key schedule that will output as many expanded key bits as needed regardless of the specific block cipher and block size, and to increase the randomness of the expanded key bits compared to the expanded keys generated by existing block ciphers' key schedules. When creating the four instantiations of elastic block ciphers, we used the stream cipher RC4 to illustrate the concept of a generic key schedule. Future work includes the creation of a generic key schedule for use in elastic block ciphers with improved performance.

References

1. J.H. An and M. Bellare, Constructing VIL-MACs from FIL-MACs: Message Authentication Under Weakened Assumptions, *Proceedings of Advances in Cryptology - Crypto 1999*, LNCS 1666, Springer-Verlag, 1999.
2. K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima and T. Tokita, Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis, *Proceedings of Selected Areas in Cryptography*, LNCS 2012, Springer-Verlag, pages 39-56, 2000.
3. M. Bellare, R. Canetti and H. Krawczyk, Pseudorandom Functions Re-Visited: The Cascade Construction and its Concrete Security, *Proceedings of Foundations of Computer Science, IEEE*, 1996.
4. M. Bellare and P. Rogaway, On the Construction of Variable Length-Input Ciphers, *Proceedings of Fast Software Encryption 1999*, LNCS 1636, Springer-Verlag, pages 231-244, 1999.
5. D. Bernstein, How to Stretch Random Functions: The Security of Protected Counter Sums, *Journal of Cryptology*, Vol. 12(3), Springer-Verlag, pages=185-192, 1999.
6. E. Biham, New Types of Cryptanalytic Attacks Using Related Keys, *Proceedings of Advances in Cryptology - Eurocrypt 1993*, LNCS 0765, Springer-Verlag, 1994.
7. E. Biham and A. Shamir, Differential Cryptanalysis of the Data Encryption Standard, Springer-Verlag, New York, 1993.
8. J. Black and P. Rogaway, CBC MACs for Arbitrary-Length: The Three-Key Constructions, *Proceedings of Advances in Cryptology - Crypto 2000*, LNCS 1880, Springer-Verlag, 2000.
9. M. Ciet, G. Piret and J. Quisquater, Related-Key and Slide Attacks: Analysis, Connections and Improvements, Extended Abstract, UCL Crypto Group Technical Report, 2002.
10. D. Cook, Elastic Block Ciphers, Ph.D. Thesis, 2006.
11. D. Cook, M. Yung and A. Keromytis, Elastic Block Ciphers in Practice: Constructions and Modes of Encryption, *Proceedings of EC2ND*, Springer-Verlag, 2007.
12. D. Cook, M. Yung and A. Keromytis, Elastic Block Ciphers: The Basic Design, *Proceedings of ASIACCS*, ACM, pages 350-355, 2007.
13. D. Cook, M. Yung, and A. Keromytis, The Security of Elastic Block Ciphers Against Key-Recovery Attacks. *Proceedings of ISC*, LNCS 4779, Springer-Verlag, pages 89-103, 2007.
14. J. Daemen and V. Rijmen, The Design of Rijndael: AES the Advanced Encryption Standard, Springer-Verlag, Berlin, 2002.
15. S. Halevi and P. Rogaway, A Parallelizable Enciphering Mode, Cryptology ePrint Archive, Report 2003/147, 2003.
16. S. Halevi and P. Rogaway, A Tweakable Enciphering Mode, *Proceedings of Advances in Cryptology* - Crypto 2003, LNCS 2729, Springer-V erlag, 2003.
17. C. Hall, D. Wagner, J. Kelsey and B. Schneier, Building PRFs from PRPs, *Proceedings of Advances in Cryptology - Crypto 1998*, LNCS 1462, Springer-Verlag, pages 370-389, 1998.

18. L. Knudsen, Truncated and Higher Order Differentials, *Proceedings of Fast Software Encryption 1994*, LNCS 1008, Springer-Verlag, pages 196-211, 1995.
19. M. Luby and C. Rackoff, How to Construct Pseudorandom Permutations from Pseudorandom Functions, *Siam Journal of Computing*, vol. 17, no. 2, pages 373-386, April 1988.
20. M. Matsui, Linear Cryptanalysis Method for DES Cipher, *Proceedings of Advances in Cryptology - Eurocrypt 1993*, LNCS 0765, Springer-Verlag, 1993.
21. M. Matsui, New Block Encryption Algorithm MISTY, *Proceedings of Fast Software Encryption 1997*, LNCS 1267, Springer-Verlag, pages 54-68, 1997.
22. I. Mironov, (Not So) Random Shuffles of RC4, *Proceedings of Advances in Cryptology - Crypto 2002*, LNCS 2442, Springer-Verlag, 2002.
23. NESSIE, NESSIE Security Report, Version 2, https://www.cosic.esat.kuleuven.ac.be/nessie 2003.
24. NIST, FIPS 46-3 Data Encryption Standard (DES), 1999.
25. NIST, Randomness Testing of the Advanced Encryption Standard Finalist Candidates, 2000.
26. NIST, A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, NIST Special Publication 800-22, http://wwww.csrc.nist.gov/publications/nistir, 2001.
27. NIST, FIPS 197 Advanced Encryption Standard (AES), 2001.
28. M. Noar and O. Reingold, On the Construction of Pseudo-random Permutations: Luby-Rackoff Revisited, *Journal of Cryptology*, vol. 12, pages 29-66, 1999.
29. S. Patel and Z. Ramzan and G. Sundaram, Efficient Constructions of Variable-Input-Length Block Ciphers, *Proceedings of Selected Areas in Cryptography 2004*, LNCS 3357, Springer-Ve rlag, 2004.
30. random.org, http://wwww.random.org/files,
31. J. Reeds,III, Cryptosystem for Cellular Telephony, US Patent 5,159,634, 1992.
32. R. Rivest, RC4, in Applied Cryptography by B. Schneier, John Wiley and Sons, New York, 1996.
33. Rivest, Robshaw, Sidney and Yin, RC6 Block Cipher, http://www.rsa.security.com/rsa labs/rc6, 1998.
34. B. Schneier and J. Kelsey, Unbalanced Feistel Networks and Block Cipher Design, *Proceedings of Fast Software Encryption 1996*, LNCS 1039, Springer-Verlag, 1996.
35. B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall and N. Ferguson, Twofish, http://www.schneier.com/twofish.html 1998.
36. R. Schroeppel, Hasty Pudding Cipher, http://www.cs.arizona.edu/rcs/hpc, 1998.
37. S. Vaudenay, A Classical Introduction to Cryptography, Springer, Berlin, 2006.
38. D. Wagner, The Boomerang Attack, *Proceedings of Fast Software Encryption 1999*, LNCS 1636, Springer-Verlag", pages 156-170, 1999.

Author Biographies

Debra Cook's research interests are focused in applied cryptography and security. She has a M.S. and Ph.D. in computer science from Columbia University in New York, and a B.S. and M.S.E. in mathematical sciences from the Johns Hopkins University in Baltimore, Maryland. After graduating from Johns Hopkins, she was a senior technical staff member at Bell Labs and AT&T Labs before pursuing her Ph.D. She was then a security researcher at Bell Labs. She has also taught at Columbia University, the New Jersey Institute of Technology and the University of Crete.

Angelos D. Keromytis is an Associate Professor of Computer Science and the Director of the Network Security Laboratory at Columbia University in New York. His current research interests include protection mechanisms against denial of service attacks, network worms, and collaborative self-healing software systems, with a larger view toward system survivability and performance. He is author of more than 100 technical papers and has served on over 60 technical program committees. He is an associate editor for the ACM Transactions on Information and System Security and the IET Proceedings in Security. He received his Ph.D. and M.Sc. from the University of Pennsylvania, and his B.Sc. from the University of Crete, in Greece.

Moti Yung works at Google, Inc. and is a visiting research scientist in the Computer Science Department at Columbia University in New York. Previously, Moti has also been an industry consultant, the Director of Advanced Authentication Research at RSA Laboratories, the Chief Scientist of CertCo and a researcher at IBM's T.J. Watson Research Center. He received his Ph.D. in computer science from Columbia University in New York. He has published extensively in many areas of cryptography, security and computer science.