Pipelined Statistical Cipher Feedback: A New Mode for High Speed Self-Synchronizing Stream Encryption

Howard M. Heys and Liang Zhang

Abstract — In this paper, we introduce a new block cipher mode of operation targeted to providing high speed hardware-based self-synchronizing stream encryption. The proposed mode is a modification of statistical cipher feedback (SCFB) mode and is designed to be implemented using pipeline architectures for the block cipher. We refer to the mode as pipelined SCFB mode or PSCFB. In this paper, we consider the implementation characteristics and show that PSCFB is able to achieve speeds that are very close to pipelined block cipher implementations configured for counter mode. Such speeds are achieved with modest latency through the system and a small amount of memory required for the system queues with a provable guarantee of no queue overflow. Further, we examine the characteristics of PSCFB mode in response to bit errors and synchronization losses in the communication channel. Specifically, we show that the error propagation factor is modest and comparable to conventional SCFB and that synchronization recovery delay is reasonable given the expectation that synchronization loss is infrequent. Given the high efficiency and good communication characteristics of the mode, it is concluded that PSCFB is an excellent choice for high speed network applications requiring stream-oriented encryption with self-synchronizing capabilities.

Index Terms—Cryptography, Advanced Encryption Standard (AES), block ciphers, mode of operation, stream ciphers, synchronization, error propagation.

1 Introduction

In recent years, many modes of operation of block ciphers have been proposed. The motivations behind different modes vary and the appropriateness of a mode must be considered in the context of the targeted application requirements. In this paper, we consider the configuration of a block cipher, such as the Advanced Encryption Standard (AES) [1], intended to provide stream-oriented encryption with the feature of self-synchronization. Self-synchronization is required in applications where it is possible to lose synchronization between the transmitter and receiver (eg. through bit slips caused by timing errors). The mode presented in this paper is an extension of the concept of statistical self-synchronization introduced in [2] and analyzed in [3]. A typical application for the mode presented in this paper is stream-oriented physical layer communication where high speeds are required and where it is possible for timing errors or other network faults to lead to synchronization losses.

2 Background

Symmetric key cryptography is the field encompassing encryption techniques where, in a secure communication scenario, the same key is used for encryption at the transmitter and decryption at the receiver. Symmetric key cryptographic systems may be generally categorized as based on either block or stream ciphers. The focus of this paper is a mode of operation of a block cipher, resulting in a stream cipher configuration.

2.1 Block Ciphers and Stream Ciphers

The most common method of encryption in communication networks involves the use of a block cipher, such as AES, which encrypts a block of plaintext bits to produce a block of ciphertext bits, based on operations parameterized by a key. Since its adoption as a standard for the U.S. government, AES has been widely examined and is considered to be secure and efficient for a variety of implementation environments. As a result, AES has become the most applied block cipher today and is found in a variety of applications ranging from high speed servers to smartcards. AES uses a key of 128, 192, or 256 bits to encrypt plaintext blocks of 128 bits.

Stream ciphers encrypt one symbol, typically one bit, at a time. This is generally done by XORing a stream of plaintext data with a pseudo-random, unpredictable stream of bits, referred to as a keystream, to produce the ciphertext. At the receiver, decryption is performed by XORing the received ciphertext with the exact same keystream to reproduce the plaintext stream. This requires that the keystream at the receiver is exactly synchronized, relative to the ciphertext data, to the keystream at the transmitter, so that the plaintext is correctly reproduced by the XOR operation at the receiver.

Resynchronization between the encryption and decryption processes in a communications system using a stream cipher may be derived through the physical layer communication or be achieved by periodic resynchronizations through a secondary or higher level channel. However, using a secondary channel to resynchronize the two ends of a communication channel requires significant overhead and is difficult in high speed communication systems.

Synchronization solutions which are targeted to the physical layer use cipher modes which are referred to as self-synchronizing. Despite the fact that the concept of a self-synchronizing stream cipher has been a proposed methodology for many years (see, for example, [4]), there are surprisingly few specific proposals of self-synchronizing stream cipher algorithms. A recent pro-

- *H.M. Heys is with Electrical and Computer Engineering, Faculty of Engineering, Memorial University of Newfoundland, St. John's, NL, Canada, A1B 3X5. Email: hheys@mun.ca.*
- *L. Zhang is with Avalon Microelectronics, Mount Pearl, NL, Canada. Email: liang.zhang@avalonmicro.ca.*

posal [5], submitted to the eSTREAM project [6], suffered from security flaws and was not selected for the final eSTREAM portfolio of algorithms. The eSTREAM project – a recent major research initiative in Europe undertaken to develop stream ciphers suitable for use in a variety of applications – noted in its final report [7] that the design of self-synchronizing stream ciphers is difficult and is a very significant open area of research.

2.2 Classical Block Cipher Modes of Operation

There are several well-known modes of operation that may be applied to block ciphers to produce stream-oriented encryption, similar to that of stream ciphers. For example, output feedback (OFB) mode [8] produces a keystream by feeding back the output of the block cipher to the input to generate the next block of keystream bits. The first keystream block is produced by executing a block cipher operation on an initialization vector (IV). Although OFB can be relatively efficient by utilizing the complete block of bits as keystream bits and doing a bit-wise XOR on the entire block at once, OFB mode provides no inherent mechanism for resynchronization should the receiver lose synchronization with the transmitter. As well, since the next output block is only produced following the complete processing of the current block, it is not possible to implement a pipelined structure for the block cipher. Hence, the speed achieved by the mode can only reach the maximum speed available for an iterative implementation of the block cipher. For example, using 0.18-µm CMOS technology, AES iterative implementations, based on a 128-bit block size, have achieved speeds of 1.6 Gbps [9].

Similar to OFB, counter mode can be used to produce a block of keystream to be efficiently applied to encrypt a block of bits simultaneously [8]. Counter mode uses a counter, such as an LFSR, that increments or updates for every block to produce an input for the block cipher. As with OFB, an IV is used as the first counter value to produce a block of keystream. Since the next output does not depend on the current output, but a predictable counter value, counter mode can be used when the block cipher is implemented with a pipeline architecture and, hence, counter mode can be used for very high speed applications. For example, pipelined implementations of AES have been able to achieve speeds of over 40 Gbps using 0.18-µm CMOS technology [10]. As with OFB mode, counter mode has no inherent mechanism for resynchronizing.

To provide automatic resynchronization, it is possible to configure a block cipher as a self-synchronizing stream cipher. This can be done straightforwardly using cipher feedback (CFB) mode [8]. In CFB mode, the input to the block cipher is derived by feeding back ciphertext bits. In doing so, if synchronization is lost between the transmitter and receiver, both will eventually produce keystream based on ciphertext produced at the transmitter and, as a result, resynchronization will occur. Although resynchronization is automatic, in order for resynchronization to occur for any number of lost bits in the stream, the feedback must be done using just one ciphertext bit for every block cipher operation. As a result, the efficiency of CFB mode is very low, making CFB not suitable for many applications. For example, for AES with a block size of 128, single-bit feedback CFB can only achieve speeds of 1/128 of an iterative implementation of the cipher. Thus, since a pipeline architecture of AES is typically an order of magnitude faster than an iterative implementation, CFB mode is 3 orders of magnitude slower than high speed, pipelined AES implementations!

2.3 Conventional SCFB Mode

To overcome the speed limitations of CFB, SCFB mode has been proposed [2] and analyzed [3]. Further analysis of a slightly modified version of SCFB is presented in [11].

The original SCFB mode, which we shall refer to as *conventional SCFB*, is essentially a hybrid of CFB and OFB: the cipher operates in OFB mode, while scanning the ciphertext for a special sync pattern of n bits in length (typically, $n \sim 8$). Since the ciphertext is a pseudo-random sequence of bits, the sync pattern will be observed at a statistically random point in the ciphertext stream. For a block cipher using blocks of size B bits, when the sync pattern is observed, the following B bits of ciphertext are used as the initialization vector and fed back to the input of the block cipher. Effectively, this means that the block cipher temporarily operates in CFB mode before returning to OFB mode, with the B bits following the sync pattern used as an initialization vector for OFB mode. Since both the transmitter and receiver will observe the same ciphertext, both will detect the sync pattern and will resynchronize by using the same IV block following the sync pattern.

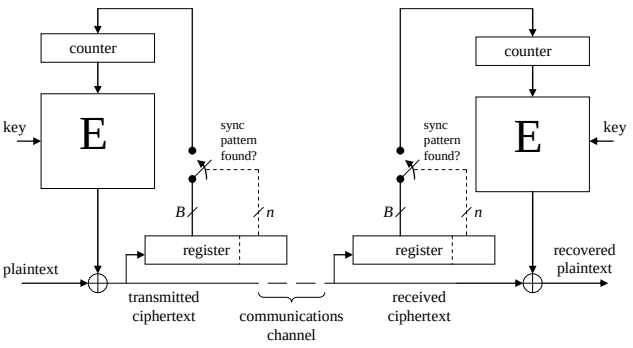

Fig. 1. Conventional SCFB (using counter mode).

Figure 1 illustrates the basic method of conventional SCFB with E representing the block cipher operation. However, the system illustrated in Figure 1 replaces OFB mode with counter mode (as indicated by the input to the block cipher being a counter and there being no feedback from the block cipher output to the input). Upon resynchronization, the system obtains an IV block from the ciphertext to reload the counter value; otherwise the input to the block cipher is taken directly to be the counter value, which is incremented for every block cipher operation. Such a construction has the same basic characteristics of OFB-based SCFB discussed in [3]. As we shall see, using counter mode in the SCFB configuration is consistent with the new mode described in this paper. Other modes based on statistical self-synchronizing

concepts and having properties similar to conventional SCFB have also been proposed in [12] and [13].

Figure 2 illustrates the sequence of ciphertext bits created at the transmitter and received at the receiver (assuming no errors in the channel). In the figure, time is increasing moving to the right. We refer to the *scanning period* as the period following the initialization of the counter with the IV from the ciphertext and prior to the sync pattern appearance in the ciphertext. The duration in bits of the scanning period is represented by random variable k, as it is dependent on the occurrence of the sync pattern in the pseudo-random ciphertext. During the scanning period, the transmitter (or receiver) scans the ciphertext stream for the n bit sync pattern. When the n bit sync pattern is observed in the ciphertext, the following B bits are used as the IV for the resynchronization of the counter mode. During the collection of the IV from the ciphertext stream, further scanning for the sync pattern is disabled. Once the IV is collected, counter mode restarts with the new IV generating the keystream for the block of bits immediately following the IV block and scanning for the sync pattern is enabled. Subsequently, when the sync pattern is next observed, the process repeats. The full duration from one sync pattern to the next is referred to as a synchronization cycle. As discussed in [3], a typical sync pattern would be n = 8 bits in length and be of the form "10000000". If AES is used as the block cipher, the block size would be B = 128.

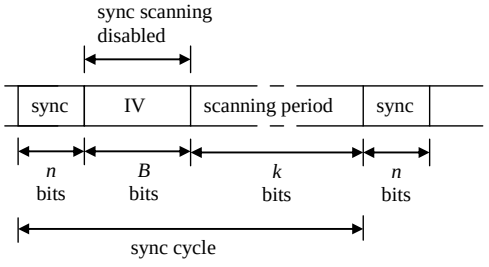

Fig. 2. Synchronization cycle for conventional SCFB using counter mode.

SCFB mode is analyzed thoroughly in [3]. It is shown that the mode has good characteristics such as modest error propagation factor (EPF) of about $(B+n)/2$ for $n \geq 5$ and a synchronization recovery delay (SRD) of about 2^n for large n. Most significantly, it is shown that SCFB can be implemented efficiently, approaching throughputs close to the throughput of the block cipher implemented in a non-pipelined, iterative style. It is shown that to achieve the highest throughput, requires buffers of increasing size (and the resulting increased system latency) to avoid data loss within the SCFB system. However, it is noted that with a buffer size of $2B$ bits and resulting latency of $2B$ bit times, SCFB implemented at an efficiency of 50% or less (that is, less than half the throughput of the block cipher) is guaranteed to have no overflow within the system queues. Hence, it is recommended that an implementation of conventional SCFB be constrained to 50% of the throughput of the block cipher implemented using an iterative architecture in order to ensure that the queue sizes in the implementation remain modest and that no bits are lost due to queue overflow.

For many network communication applications, very high speeds are required. As a result, if cryptography is to be applied at the physical layer in communication networks, implementing block ciphers using pipelining methods is often preferred, in order to achieve the desired speeds. Employing conventional SCFB using OFB mode with a pipelined implementation of a block cipher such as AES will not allow the mode to reach the potential throughput of a pipelined implementation for two reasons: (1) since OFB uses feedback, pipelining is not effective and (2) when the sync pattern is recognized in the ciphertext, the pipeline data would have to be discarded, resulting in a delay while the IV block works its way through the pipeline stages to become keystream and resynchronize the OFB mode. The first issue can be resolved by replacing OFB with counter mode (as shown in Figure 1), which can be implemented using a pipeline architecture. The second issue can not be fixed by simply using counter mode and will require an adjustment of the definition of SCFB to ensure that a long processing delay does not occur at every resynchronization. This is critical because resynchronization occurs frequently in SCFB – for example, every few hundred bits for n = 8 [3].

For these reasons, in this paper, we present a modified version of SCFB mode, designed to be compatible with pipelined block cipher implementations. We refer to this new mode as pipelined SCFB or PSCFB.

3 Proposed New Mode: PSCFB

Pipelined SCFB mode has been created to allow for efficient utilization of a block cipher implemented using a pipeline architecture. PSCFB makes use of counter mode configuration of the block cipher and extends the duration of time during which scanning for a new sync pattern in the ciphertext is disabled.

3.1 Pipelined Block Ciphers

Block ciphers, in an appropriate mode such as counter mode, lend themselves to a pipeline architecture. A cipher round typically consists of nonlinear substitutions (in AES, these are called S-boxes), linear transformations (in AES, this involves the XOR of subsets of bits and is performed through the use of the ShiftRow and MixColumn operations), and key mixing (typically, as in AES, the XOR of key bits with the data bits). As part of the process, round keys must be derived from the original cipher key and applied at the appropriate round. It is straightforward to place registers between rounds and subsequently create a pipeline structure with each stage of the pipeline corresponding to a round[1]. The resulting L-stage pipeline structure for a general block cipher is illustrated in Figure 3, where L is the number of rounds in the cipher. For example, for AES, L = 10 for a basic pipelined implementation[2] with a 128-bit block size and a 128-

[1] Equating pipeline stages to cipher rounds is sometimes referred to as outer round pipelining. Inner round pipelining is also possible [10], which places pipeline registers also within the round of a cipher, thereby increasing the number of stages and increasing the cipher throughput.

[2] Note that some implementations of AES, such as [10], may also use a

bit key size.

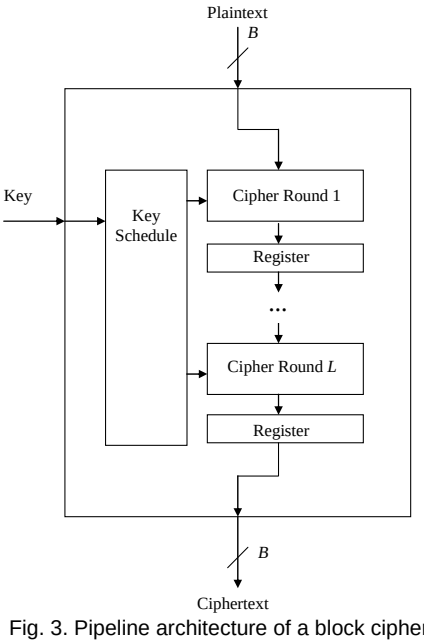

Fig. 3. Pipeline architecture of a block cipher.

It is well known that a structure such as that of Figure 3 could be used to implement a block cipher configured for counter mode, since the input to counter mode is not dependent on the previous output block. Hence, consecutive pipeline stages can be used to operate on consecutive blocks. The throughput of the system is given by

$$T_{max} = B / t_{clk} \text{ bps} \quad (1)$$

where t_{clk} represents the period of the clock used to drive the pipeline and t_{clk} can be as small as the critical path delay through the pipeline stages.

3.2 Applying Pipelining to Conventional SCFB

Consider the behaviour of a conventional SCFB system using counter mode as shown in Figure 1 based on the pipeline architecture of Figure 3. Upon the recognition of a sync pattern in the ciphertext, the next B bits of ciphertext will be used as the IV which is used to set a new counter starting value for the next scanning period. During the IV collection, sync pattern scanning is disabled. In conventional SCFB using a pipeline architecture, in order for the IV to produce a new block of ciphertext to follow the IV block, the system would have to load the IV and wait for all L stages to complete for the next B bits of keystream and resulting ciphertext to be produced. The data residing in the pipeline registers of Figure 3, would be discarded and the system would need to absorb the delay of $L \cdot t_{clk}$ required to produce the first block following the IV block. The remaining blocks produced during the scanning period could be produced at a rate of one every t_{clk} since the pipeline is primed. The long delay during resynchronization will require large queues to absorb the large variations in the production of a keystream block and will result in it being necessary to run the mode at a rate significantly less than the full pipeline rate, resulting in a throughput much less than T_{max} in (1).

Assume that B = 128 bits and consider that, for n = 8 and sync pattern "10000000", the average length of the scanning period is $2^n - n$ = 248 bits [3]. Since the scanning period is followed by 8 bits of the sync pattern, assuming, for simplicity, that the scanning period length is exactly the average, a full sync cycle would consist of 248 + 8 = 256 bits followed by 128 bits of IV and it would take $L \cdot t_{clk}$ + $2 \cdot t_{clk}$ to produce the 3 blocks (384 bits) of the full sync cycle (that is, 2 blocks for the scanning period plus the sync pattern, followed by one block for the IV). As a result, the throughput achieved will be

$$T_{SCFB} = \frac{3B}{(L+2)t_{clk}} \text{ bps.} \quad (2)$$

For an AES pipelined implementation, with L = 10, T_{SCFB} = $32/t_{clk}$. This means that, given the specific assumptions of (2), conventional SCFB using counter mode with a pipelined implementation has a throughput which is T_{SCFB}/T_{max} = 25% of the throughput of a basic pipelined counter mode (assuming that the critical path of each pipeline implementation is similar and therefore the value of t_{clk} can be considered to be the same in (1) and (2)). The assumptions of (2) represent an idealized, deterministic view and, in practice, the duration of the scanning period will vary: some scanning periods will be longer than 248, which will improve the throughput beyond (2); some periods will be shorter, which will diminish the throughput below (2). Hence, the expression in (2) is simply a rough guideline for the expected throughput and is only useful to gain perspective on the drawbacks of conventional SCFB using a pipelined implementation of counter mode.

3.3 Pipelined SCFB Mode

In order to mitigate the effect of the discarded pipeline data and the resulting limitations on the SCFB throughput, we propose a new block cipher mode: pipelined SCFB or PSCFB mode. PSCFB adds to the synchronization cycle, a number of blocks following the IV block during which the counter mode continues to operate without resynchronization and with the sync pattern scanning still disabled. The entire period during which sync pattern scanning is disabled is referred to as the *blackout period*. The resulting synchronization cycle is illustrated in Figure 4. As shown in the diagram, there are L blocks (totaling $L \cdot B$ bits) during which sync scanning is disabled with the first B bits of the blackout period corresponding to the IV collection phase.

The value of L is selected to correspond to the number of pipeline stages in the counter mode implementation of the block cipher. Thus the blackout period allows the pipeline of a block cipher to produce L–1 blocks, one block every t_{clk}, until the IV block works its way through all stages of the pipeline. Hence, it is not necessary to discard the blocks in the process of being produced in counter mode (i.e., held within the pipeline stages) because a resynchronization occurs. This enables an implementation of the mode to operate very efficiently. Con-

pipeline stage for the first key mixing operation, resulting in L = 11. However, in this paper, we assume that the first key mixing operation can be incorporated into the first pipeline stage, resulting in L = 10, with the penalty of a slightly higher critical path delay in the first pipeline stage.

sider, for example, n = 8 and a scanning period of exactly 248 bits, followed by 8 bits of the sync pattern. Since the blackout period length in blocks corresponds to the number of pipeline stages, one block of keystream and, hence, B bits of ciphertext is produced every t_{clk} as is desired to achieve the full capability of a pipelined block cipher. That is, if all scanning periods were exactly the average length, PSCFB mode theoretically achieves the full pipelined throughput T_{max} as defined in (1).

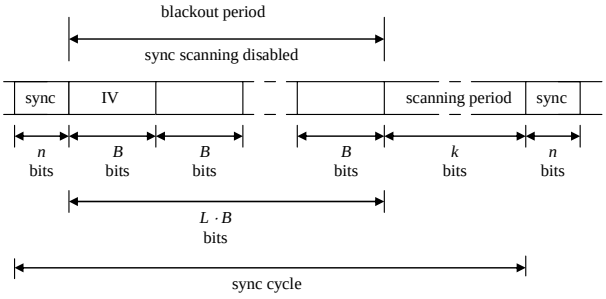

Fig. 4. Synchronization cycle for PSCFB.

In practice, it is not possible to achieve T_{max}. Since not all blocks produced by the block cipher produce B bits of keystream (that is, partial blocks can result when resynchronization occurs), the practical maximum throughput is less than T_{max}. Generally, partial blocks from the output of the block cipher are used when the last bit of the blackout period does not align with the last bit of a block generated by the block cipher. In such cases, the last block cipher operation during the blackout period will be used to produce fewer than B keystream bits. Subsequently, the first B bits in the scanning period use keystream based on the new counter value, determined from the IV block in the ciphertext stream.

Consider, for example, the inefficient scenario that occurs when the first n bits following a blackout period are the sync pattern (that is, the scanning period length is k = 0) and, hence, a sync cycle consists of L full blocks (one IV and L–1 remaining blackout blocks) and one partial block consisting of n bits. As a result, in L+1 time periods of t_{clk}, $L{\cdot}B + n$ bits of keystream are produced. If this was true for every sync cycle, the resulting throughput of this inefficient scenario would be

$$T_{PSCFB^*} = \frac{L \cdot B + n}{(L+1) \cdot t_{clk}} \text{ bps} \tag{3}$$

resulting in a *encryption efficiency*, η, of

$$\eta = T_{PSCFB^*} / T_{max} = \frac{L + n/B}{L+1} . \tag{4}$$

The encryption efficency η represents the number of ciphertext bits that can be produced in PSCFB mode relative to the number of output bits produced by the block cipher (which is equivalent to the number of bits produced in a pipelined implementation of basic counter mode). For n = 8, L = 10, and B = 128, η = 91.477%. Note that, in deriving (4), we have assumed that the critical path timing of PSCFB and a pipelined counter mode implementation are similar and, hence, t_{clk} is the same value in both (1) and (3).

It can be shown that, for the normal operation of PSCFB (i.e., not just the special case leading to (4)), η can be lower bounded by $L/(L+1)$. This implies that for R bits produced at the block cipher output, at least $R{\cdot}L/(L+1)$ ciphertext bits of PSCFB mode are produced. In practice, this is lower bound on encryption efficiency because many sync cycles will lead to more efficient production of keystream than the inefficient scenario described above – as high as 100% encryption efficiency when the blackout period ends on block cipher boundary and no partial block is used as keystream. As we shall discuss in Section 4, when considering the implementation of PSCFB, $L/(L+1)$ will also represent a maximum on the relative throughput of an actual implementation of PSCFB, in the sense that $[L/(L+1)]{\cdot}T_{max}$ is the maximum rate at which data may be encrypted using PSCFB to ensure that no queue overflow occurs in the system queues.

Note that the special case of a purely iterative implementation of the block cipher used in conventional SCFB mode corresponds to the scenario of L = 1 and the resulting encryption efficiency from (4) for n = 8 and B = 128 is η = 53.125%. This is consistent with the requirement discussed in [3] to run the SCFB system at 50% or less of the block cipher throughput so that the queues associated with the implementation will not overflow.

The operation of PSCFB is described in detail using pseudo-code in Appendix A.

4 PSCFB Implementation

When assessing the value of PSCFB mode, it is critical to consider implementation issues such as queuing requirements, system latency, and practically achievable throughput. A block diagram of PSCFB for encryption is given in Figure 5. The diagram illustrates the pipelined nature of the block cipher and shows that queues are required on both the input (plaintext) and output (ciphertext) sides of the system. Each element in a queue stores one bit of data. We refer to these queues as the plaintext queue and the ciphertext queue. These queues are needed to ensure that, during periods of resynchronization, data can be buffered temporarily prior to XORing with keystream to manage scenarios which require partial block cipher outputs (that is, cases where fewer than the full B bits of a block cipher output are required for the keystream due to the resynchronization boundary). Since a large proportion of a sync cycle consists of the blackout period (see Figure 4), when the sync scanning is disabled, with an appropriate processing rate and large enough queue, buffer overflow can be avoided. Note that a PSCFB decryption implementation is similar except that bits enter the system at the ciphertext queue and leave from the plaintext queue and sync scanning is performed on the arriving ciphertext bits.

In order to maximize the throughput of the PSCFB system, input and output are handled in blocks of D bits as shown in Figure 5. It is assumed the data arrives at the plaintext queue and leaves the ciphertext queue at a rate of D/t_{clk} bps, where t_{clk} represents the system clock period. Similarly, data is transferred out of the plaintext queue to the ciphertext queue at a rate of typically B/t_{clk} bps during the scanning period. However, when the system is resynchronizing, typically the transition from the blackout pe-

riod to the reinitialized counter mode for the scanning period does not fall on a block boundary and, in this case, d bits, where $d < B$, are transferred from the plaintext to the ciphertext queue during one period of the clock. The system clock is used to drive the pipeline architecture as well: one period of the clock, t_{clk}, represents the time required per pipeline stage. The value of t_{clk} must be larger than the critical path delay in the queuing circuitry and the pipeline stages of the block cipher. Assuming that the critical path delay of a pipeline stage is substantial (which is expected when a pipeline stage is equivalent to a full round of a cipher such as AES), then it is reasonable to assume that t_{clk} is determined from the critical path delay of the pipeline stages.

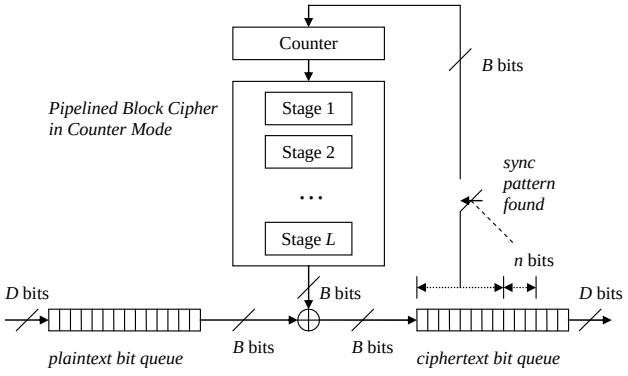

Fig. 5. PSCFB encryption system implementation.

When a partial block is transferred, usually the amount of data in the plaintext queue will increase. The value of D must be small enough to ensure that the number of bits in the queue does not exceed some finite value. However, from the perspective of the system throughput, it is desirable for D to be as large as possible since the system throughput is given by

$$T_{PSCFB} = D / t_{clk} \text{ bps.} \quad (5)$$

The ciphertext queue acts in a complementary manner to the plaintext queue. The system is initialized with the plaintext queue empty and the ciphertext queue full of arbitrary data. At the start of a clock period, D bits are enqueued in the plaintext queue, while D bits are dequeued from the ciphertext queue. Subsequently, an appropriate number of bits are transferred from the plaintext queue to the ciphertext queue. In any one clock period, there are 3 cases representing the transfer of bits from the plaintext queue to the ciphertext queue: (i) B bits (as is the case for normal counter mode operation), (ii) zero bits (because there are not sufficient bits in the plaintext queue to transfer), and (iii) d bits, $1 \le d < B$ (as is the case at the end of the blackout period and only a partial block is used for keystream). For case (ii), it is necessary to stall the pipeline of the block cipher for the clock period, since no keystream bits are used.

Assuming that the size of each queue is M bits, then if there are h bits in the plaintext queue, the number of bits in the ciphertext queue is given by $M–h$. If there is a delay in moving data from the plaintext queue to the ciphertext queue, while the plaintext queue fills up, the ciphertext queue empties; if M is too small, it is possible for an overflow to occur in the plaintext queue and this would simultaneously result in an underflow in the ciphertext queue. It is desirable to have M large enough that the plaintext queue does not overflow. Since there is a total of M bits in the queuing system at one time and every clock period, D bits arrive and D bits leave, assuming that M is large enough so that there is no queue overflow, the total delay experienced by a bit from when it enters the plaintext queue to when it leaves the ciphertext queue is upper bounded by $\lceil M/D \rceil \cdot t_{clk}$.

The block cipher operation is capable of producing a throughput of B/t_{clk} bps, while the system throughput is given by D/t_{clk}. We define *implementation efficiency*, α, to represent the ratio of throughput of an implementation of a PSCFB system to the throughput that could be achieved by a block cipher configured in counter mode and implemented using a pipeline architecture. Hence, the implementation efficiency is given by

$$\alpha = D / B . \quad (6)$$

In order to deal with the block realignment as the result of a resynchronized counter mode, α must be suitably small enough to ensure that the queues, of selected size M bits, do not overflow. Of course, having α to be too small, would be equivalent to a diminished PSCFB system throughput.

enqueue D bits from input
if (# bits in plaintext queue $\ge B$) then
    dequeue d bits, $1 \le d \le B$
        and XOR with keystream bits
else
    stall pipeline

Fig. 6. Plaintext queue process during each clock period.

Consider the operation of the plaintext queuing system during each clock period, as specified in Figure 6. The number of bits dequeued from the plaintext queue is represented by d and $d = B$ except in the case that the boundary between the blackout period and the new scanning period results in the use of a partial block produced by the block cipher, thereby resulting in $d < B$. Although not illustrated, the process for the ciphertext queue would involve dequeuing D bits, followed by the enqueuing of the bits transferred from the plaintext queue.

Given the specification of Figure 6, the following theorem can be considered to ensure that the queue size M is large enough and the implementation throughput does not exceed an acceptable value to ensure that overflow does not occur in the plaintext queue.

Theorem. Assume the operation of the plaintext queuing process as defined in Figure 6. If the implementation efficiency, α, satisfies $\alpha \le L/(L+1)$, then there will be no overflow in a PSCFB system implemented using a plaintext queue of size M bits, when $M \ge B + 2D - 2$.

Proof: Consider the number of bits in the queue at the start of the process illustrated in Figure 6 (i.e., just prior to the "enqueue" step). Given the behaviour of PSCFB

mode, there are 3 general cases in terms of the change in the number of bits in the queue. These are illustrated in the following table, where $\delta = B - D$ and ε is the number of bits in the queue at the start of the process. We will briefly discuss each case.

	Case 1	Case 2	Case 3
# bits enqueued	D	D	D
# bits dequeued	B	0	$d < B$
Change in ε	$-\delta$	$+D$	$+D-d$
Required ε for case to occur	$\geq \delta$	$< \delta$	$\geq \delta$

Tab. 1. Cases Used in Proof

Case 1: This case represents the circumstance where all B output bits of the block cipher are to be used as keystream and there are at least B bits in the queue when the dequeue operation is to take place, thereby ensuring that B bits will be dequeued. As D bits must have been enqueued at the start of the process, there is a decrease in the number of bits in the queue by $\delta = B - D$.

Case 2: In this case, there are not B bits in the queue at the time of dequeuing and, hence, the pipeline is stalled and no bits are dequeued. As a result, there is an increase of D bits in the queue during a clock cycle.

Case 3: In this case, due to the behaviour of PSCFB during a resynchronization, at the end of the blackout period, only a partial block of d bits, $1 \leq d < B$, from the block cipher is used as keystream and it is only necessary to dequeue d bits. As a result, there is an increase in ε of $D - d$ for $D > d$, a decrease of ε of $d - D$ for $D < d$, and no change in ε if $D = d$. Due to the behaviour of PSCFB, there must be at least $L \cdot B$ bits dequeued from case 1 before the next partial block is used, since there are $L \cdot B$ bits during the blackout period before the next resynchronization boundary. Hence, instances of case 3 must be separated by at least L occurrences of case 1.

Consider now the combination of all 3 cases. Assuming that the queue starts at empty (that is, $\varepsilon = 0$), the first clock cycle will correspond to case 2 and then proceed with a variable sequence of cases 1, 2, and 3, dependent on the operation of PSCFB mode based on the ciphertext data. Since for most of the operation of PSCFB, the system operates in counter mode, case 1 will tend to drive the number of bits in the queue towards 0, with the pipeline stalls of case 2 momentarily increasing the number of bits in the queue.

The scenario resulting in the most bits in the queue would be a sequence of clock cycles which starts with $\varepsilon = \delta - 1$. This has to be followed by case 2, which increases the number of bits to $\varepsilon = \delta - 1 + D$. Subsequently, if the next clock cycle corresponds to a partial block dequeue (case 3) with $d = 1$, then this removes only 1 bit from the queue, while adding another D bits, resulting in

$$\varepsilon = \delta - 1 + D + D - 1 = B + D - 2 . \quad (7)$$

Since in (7), $\varepsilon \geq \delta$ for all $D \geq 1$, case 2 will not occur until more bits are removed from the queue and $\varepsilon < \delta$. Hence, the number of bits in the queue will not be increased beyond the value of ε in (7) as the result of case 2. Also, we must encounter case 1 at least L times, before executing case 3 again. This means that $L \cdot \delta$ bits must be dequeued before case 3 adds more bits to the queue. Since $\alpha \leq L/(L+1)$,

$$B \geq \frac{L+1}{L} D \quad (8)$$

and

$$L\delta = L(B-D) \geq L\left(\frac{L+1}{L} D - D\right) = D . \quad (9)$$

Hence, at least D bits must be removed before $D - d$ bits are added back into the queue. So case 3 can not increase the queue size beyond the value of ε in (7).

Consequently, equation (7) represents the maximum number of bits in the queue at the start of the process defined in Figure 6. However, since D bits are always enqueued prior to the dequeuing considerations, the maximum number of bits in the queue will be the value indicated by (7) plus the D bits that are added when the input bits are enqueued. Hence, there are at most $B + 2D - 2$ bits in the queue and the theorem is proven. □

The significance of the theorem is that it is possible to construct PSCFB systems with high implementation efficiency (i.e., $\alpha \to 1$), when L is large, such that a modest queue size is guaranteed to have no overflow. Also, as a consequence of modest queue sizes, system latency is modest. Although the theorem is expressed with respect to the plaintext queue, the implication is that the ciphertext queue with the same size would not underflow. Using a pipelined implementation of AES with $B = 128$ and $L = 10$, allows us to set $D = 116$, thereby achieving an implementation efficiency of $\alpha = 90.625\%$ and, with a queue size of $M = 358$ bits, ensuring that the plaintext queue will not overflow. The resulting delay from when a bit enters the plaintext queue to when it leaves the ciphertext queue is no more than $\lceil 358/116 \rceil \cdot t_{clk} = 4 \cdot t_{clk}$.

5 Resynchronization Delay and Error Propagation

In this section, we investigate how the sizes of the blackout period and sync pattern affect the resynchronization properties and error characteristics of PSCFB mode. We shall do this by considering two metrics – the synchronization recovery delay (SRD) and the error propagation factor (EPF) – and by undertaking simulations of a PSCFB system, modeled as an encryption system (transmitter), communication channel, and decryption system (receiver). Each simulation result produced in this section is based on the encryption, and subsequent decryption, of 10^{10} bits, with the individual bit slips or bit errors, whichever is appropriate, generated in the channel at a fixed rate of one event every 10^5 bits. In all cases, AES is used for the block cipher and the format of the sync pattern used is "10…00". For all scenarios, the 95% confidence interval is calculated for all simulation points and plotted in each figure. For most points, the confidence interval is too small to be visible on the graphs.

5.1 Synchronization Recovery Delay

The synchronization recovery delay is the expected num-

ber of bits following a sync loss due to a slip before synchronization is regained [3]. SRD does not include the bits that are lost directly due to the slip.

5.1.1. Bounds on SRD

In [3], lower and upper bounds are developed for SRD of conventional SCFB, parameterized by n and B. In Appendix B, we have extended this analysis to PSCFB with the additional parameter of L.

The resulting lower bound on SRD is given by:

$$SRD \geq \frac{3}{2}(n + L \cdot B) + \frac{1}{2\mu}(n + L \cdot B)E\{k\} + \frac{1}{2\mu}E\{k^2\} \quad (10)$$

where k is the size of the scanning period, $E\{k\}$ is the average size of the scanning period, $E\{k^2\}$ is the second moment of k and the average sync cycle size is given by $\mu = n + L \cdot B + E\{k\}$. As discussed in [3], the distribution of k can be approximated as the geometric distribution for many sync patterns including sync patterns of the format "10…00". As a result, we may approximate $E\{k\}$ and $E\{k^2\}$ to be $E\{k\} = 2^n - 1$ and $E\{k^2\} = 2^{2n+1} - 3 \cdot 2^n + 1$.

The upper bound, as shown in Appendix B, can be derived to be:

$$SRD < \frac{3}{2}(n + L \cdot B) + \frac{1}{2\mu}(n + L \cdot B)E\{k\} + \frac{1}{2\mu}E\{k^2\} + \frac{n}{2^n}(n + L \cdot B)\lambda. \quad (11)$$

where λ is an upper bound on the expected number of sync cycles until resynchronization is achieved after a false synchronization (that is, a bit sequence, created by a slip or insertion, is falsely interpreted to be the sync pattern) and $\lambda = (1 - 1/2^n)^{-(n+LB)}$.

5.1.2. SRD vs. Blackout Period Duration

In order to investigate SRD versus the blackout duration, simulations were undertaken, as discussed in the beginning of this section, with B = 128, n = 8, and varying L. During the simulations, bit slips were generated in the communications channel and the average number of bits prior to resynchronization was computed, giving an empirical estimate of SRD. The resulting relationship is presented in Figure 7. The bounds of (10) and (11) are also plotted on the graph. Note that the upper bound grows very rapidly with L and values for $L \geq 9$ are not shown, as they substantially exceed the range of the graph.

It is clear from the figure that the simulation results for SRD follow very closely the lower bound of (10) and, as predicted by (10), SRD increases with the size of the blackout period. This is the expected consequence of a larger L resulting in a larger sync cycle size. Larger SRD due to larger L is a tradeoff that must be made to accommodate fast pipeline designs requiring large L. The upper bound on SRD is reasonable for small L with moderate value of n = 8. However, as L increases, the upper bound becomes very poor as it is derived by accounting for the possibility of false synchronizations, which in the extreme cases considered by the bound become more difficult to recover from as L increases.

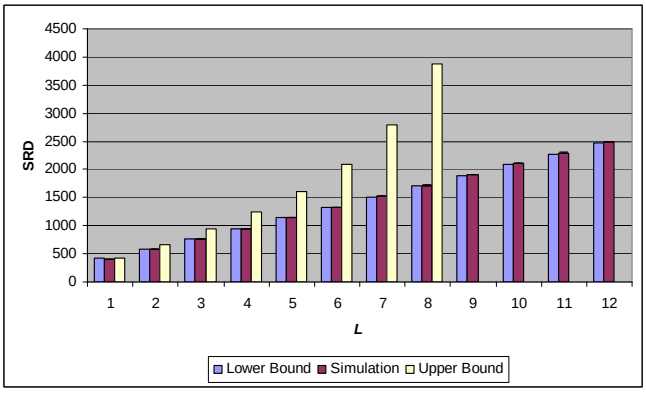

Fig. 7. SRD vs. size of blackout period for B = 128, n = 8.

5.1.3. SRD vs. Sync Pattern Sizes

We have also run simulations to investigate SRD versus different values of the size of sync pattern, n. Figures 8 and 9 show the results with B = 128, for conventional SCFB (i.e., L = 1) and for PSCFB with L = 10. It can be seen that as n increases, SRD is influenced by both the size of the blackout period, $L \cdot B$, and the size of the scanning period, which is approximately 2^n for a geometrically distributed value of k. For smaller values of n, the effect of false synchronizations causes legitimate synchronizations to be often lost (for example, a legitimate sync pattern might be ignored during the blackout period associated with a false synchronization), thereby causing longer delays to resynchronize for the mode with the longer blackout period. Hence, SRD is much larger for PSCFB with L = 10 than conventional SCFB with L = 1 for small n.

Since the approximate lower bound of (10) does not account for the occurrence of false synchronizations, for small n, when false synchronizations cause significant delays in resynchronization, the lower bound is very loose for PSCFB with L = 10, with the empirical SRD being significantly larger.

The upper bound of (11) generally provides a tight bound for L = 1 for modest to large sized values of n, such as $n \geq 6$. (Only values for $n \geq 5$ are plotted.) However, for L = 10, the upper bound is very loose until n increases so that $n \geq 9$. (Only values for $n \geq 8$ are plotted.) This results from the effects of false synchronizations considered in the upper bound: since systems with small n have small sync cycle sizes, many sync cycles occur before a resynchronization that recovers the system from a false synchronization. In practice, this effect is not as significant as predicted by the upper bound and the simulation results are better predicted by the lower bound.

Note that in some cases, such as for L = 1 and n = 10 in Figure 8, the lower bound is marginally higher than the simulation result. This can be explained by considering that the lower bound is derived based on the assumption that the size of the scanning period, k, follows the geometric distribution. As discussed in [3], although this is a good approximation, in fact, k is not exactly geometrically distributed and the average size of the scanning period is actually slightly smaller, which becomes evident in the comparison of the lower bound and simulation results

when they are very close in value.

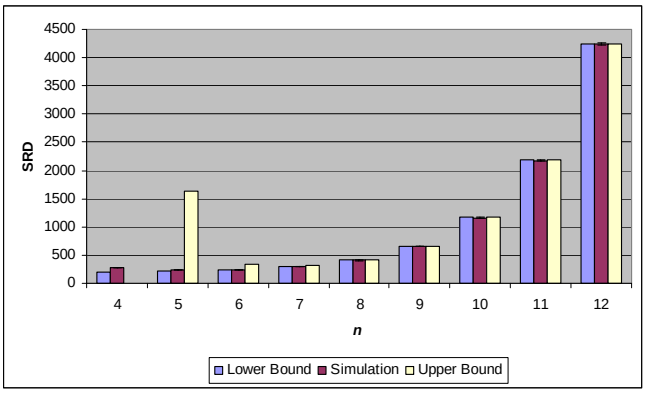

Fig. 8. SRD vs. sync pattern size for B = 128, L = 1.

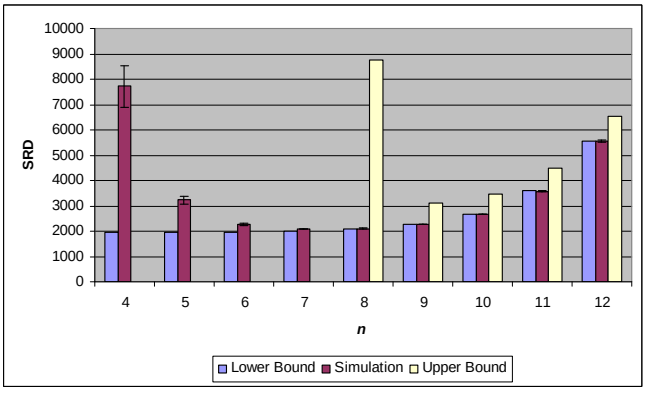

Fig. 9. SRD vs. sync pattern size for B = 128, L = 10.

5.2 Error Propagation Factor

The error propagation factor is defined as the bit error rate at the output of the decryption divided by the probability of a random bit error in the communication channel [3].

5.2.1. Bounds on EPF

In Appendix B, we have extended lower and upper bounds developed for conventional SCFB mode to PSCFB mode, parameterized by n, B, and L.

The resulting lower bound on EPF is simply given by:

$$EPF \geq 1 + \frac{1}{2}[n + B]. \tag{12}$$

Clearly, this lower bound is not dependent on the value of L, which is proportional to the duration of the blackout period, and is implying the possibility that L will have little impact on EPF.

Also, from the analysis in Appendix B, the upper bound is given as:

$$EPF < 1 + \frac{n+B}{2} + \frac{n+B}{\mu} + \frac{n}{\mu(2^n - 1)}\left[\frac{3}{4}(n + L\cdot B)E\{k\} + \frac{1}{4}E\{k^2\} + \frac{3}{4}(n + L\cdot B)^2 + \frac{(n + L\cdot B)}{2}\mu\cdot\lambda\right] \tag{13}$$

where $E\{k\}$, $E\{k^2\}$, μ, and λ can be estimated using the geometric distribution as indicated for (10) and (11).

5.2.2. EPF vs. Blackout Duration

In order to investigate EPF versus the blackout duration, simulations were undertaken with B = 128, n = 8 and varying L. The duration of the blackout period in bits is given by $L{\cdot}B$. During the simulations, bit errors were generated in the communications channel: the resulting errors generated at the receiver output were counted and an average was determined over all bit error events in order to determine the EPF. The resulting relationship is presented in Figure 10.

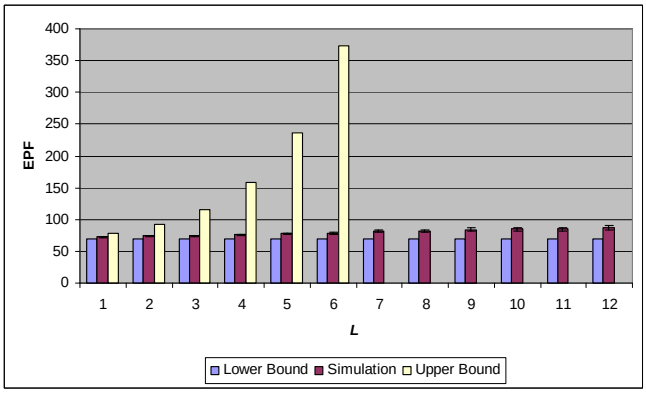

Fig. 10. EPF vs. size of blackout period for B = 128, n = 8.

The simulation results from Figure 10 illustrate that the EPF trends upward slowly when the size of the blackout period increases. There is a small difference between the simulated results and the lower bound on EPF and this difference can be explained as the result of the effects of false synchronizations, which can occur when bit errors erroneously result in a sync pattern appearing at the receiver. During a false synchronization event, much of a sync cycle will be unsynchronized between the transmitter and receiver. Since the size of a sync cycle is dependent on L, larger L implies greater EPF when a false synchronization occurs at the receiver. Hence, as L increases in the graph, the effects of false synchronizations become more evident and EPF increases. False synchronizations are not incorporated into the lower bound on EPF given in (12) and, hence, the lower bound is not dependent on L. The effect of false synchronizations is, however, incorporated into the upper bound of (13). Because the effect caused by false synchronizations is very difficult to bound tightly, as L increases the upper bound becomes very poor. For this reason, it is not plotted for $L \geq 7$.

5.2.3. EPF vs. Sync Pattern Sizes

We have also run simulations to investigate EPF versus different values of sync pattern size, n. For values of B = 128, the results of EPF versus n for conventional SCFB (L = 1) are shown in Figure 11 and for PSCFB with L = 10, the results are shown in Figure 12.

The simulation results illustrate that, for L = 10, the EPF increases significantly when the size of the sync pattern decreases. For L = 1, this increase is much less pro-

nounced. For small n, since the scanning period is much smaller, a false sync pattern may take several sync cycles to clear up as the effects of a loss of sync may spill over from the scanning period into the next sync pattern and blackout period. Hence, EPF is expected to be higher for smaller n. Conventional SCFB mode, where $L = 1$, has a shorter blackout period than PSCFB with $L = 10$. As a result, for small n, resynchronization is quicker for smaller L and EPF for conventional SCFB is not as high as for PSCFB mode with large L. As n increases, for values of $n \geq 8$, EPF for conventional SCFB and PSCFB are comparable and close to the lower bound. For small values of n, the upper bound, which considers the effect of false synchronizations, is very loose (and not plotted for small n), while as n increases, the upper bound becomes tighter for both $L = 1$ and $L = 10$.

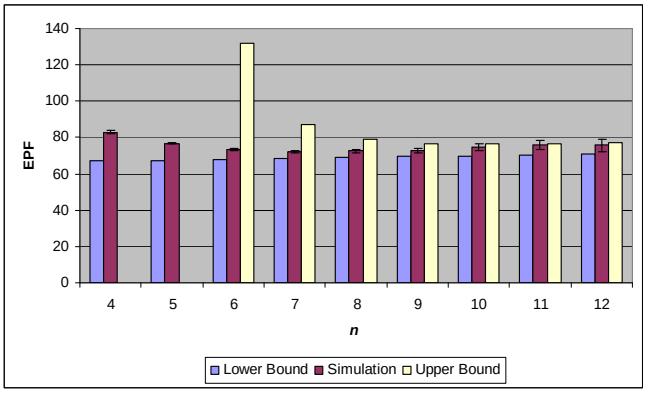

Fig. 11. EPF vs. sync pattern size for $B = 128$, $L = 1$.

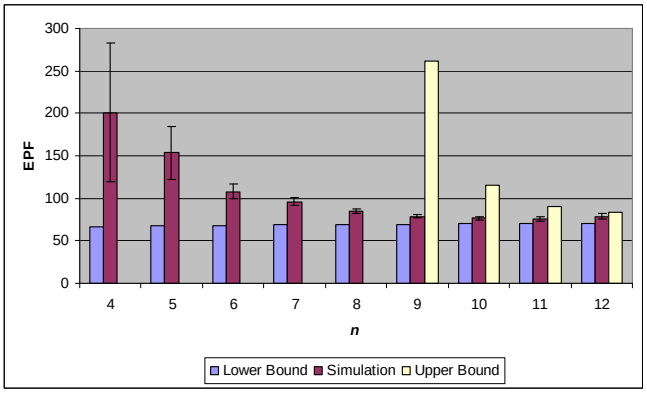

Fig. 12. EPF vs. sync pattern size for $B = 128$, $L = 10$.

6 Security of PSCFB

PSCFB mode is a hybrid mode constructed from the combination of two block cipher modes accepted as secure methods of encryption: counter mode and CFB mode. Formal security proofs exist for both counter mode [14] and CFB mode [12]. However, to date, no formal security proof has been given for SCFB mode.

Various attempts at designing efficient dedicated self-synchronizing stream ciphers (rather than applying CFB mode to block ciphers) have been shown to be susceptible to chosen ciphertext attacks which exploit weaknesses in the cipher structures [15][16]. For CFB, SCFB, and PSCFB block cipher modes, a chosen ciphertext attack is similar to a chosen plaintext attack on the block cipher in electronic codebook mode. Since immunity to chosen plaintext attack is a security requirement for block ciphers, attacks similar to those in [15][16] will not be applicable to well-designed block ciphers used in PSCFB mode.

In [3], it is argued that conventional SCFB based on OFB, has a very low probability of significant repeated keystream for a block size of $B = 128$ bits. The security of a PSCFB application will be compromised if a counter value occurs that is identical to a counter value used during a previous sync cycle, thereby resulting in a repitition of a significant number of keystream bits. In this circumstance, the keystream bits, in fact, will repeat until the next sync pattern is detected and a new counter IV is loaded. Since such periods of significant repeated keystream compromise the security of the system, we will investigate the probability that a portion of a sync cycle contains repeated keystream as the result of reused counter values.

Consider first the probability that the size of a sync cycle exceeds Γ bits. Assuming that the size of the scanning period, k, is approximated by the geometric distribution, the probability that the sync cycle size C is greater than Γ is given by

$$P(k \geq N) = \sum_{k=N}^{\infty} P(k) \tag{14}$$

where $P(k)$ represents the probability of a scanning period of size k and $N = \Gamma - (n + L \cdot B)$ since $C = n + L \cdot B + k$. For the geometric distribution, $P(k) = (1 - 1/2^n)^k / 2^n$, resulting in

$$P(k \geq N) = \left(1 - \frac{1}{2^n}\right)^N \approx e^{-N/2^n}. \tag{15}$$

Hence, for $P(k \geq N) < \sim e^{-40}$, $N > 40 \cdot 2^n$ and $C > n + L \cdot B + 40 \cdot 2^n$. For $n = 8$, $L = 10$, and $B = 128$, the probability that sync cycle size $C > 11528$ is less than 4.25×10^{-18}.

Counter values used in two sync cycles will overlap when the IV used to initialize counter mode of one cycle is equal to one of the counter values of the other cycle. Assuming that the counter increments its value for every execution of the block cipher, two cases can result in overlapping keystreams for different sync cycles: (1) the counter starts with an IV for the current sync cycle which increments up to the value of a previous sync cycle's IV and (2) the IV of the current sync cycle is equal to a counter value from a previous sync cycle. Hence, an overlap of keystream will not occur if a sync cycle starts with an IV that is not within about C_{max}/B values of the start of a previous sync cycle, where C_{max} is the size of the largest possible sync cycle (i.e., sync cycles larger than C_{max} have negligible probability of occuring). As discussed above, we can assume that $C_{max} = 11528$ for $n = 8$, $L = 10$, and $B = 128$ since the probability of a sync cycle size greater than this is very small.

We can now determine a lower bound, Ω, on the probability that t sync cycles do not have any overlap of keystreams (caused by repeated counter values) as

$$\Omega \approx \prod_{i=0}^{t-1}\left[1-\frac{i(2C_{\max}/B)}{2^B}\right] \tag{16}$$

which is derived by considering the selection from 1 up to t random sync cycle IVs which are not within a window of $2C_{max}/B$ values centred around previous selections of IVs. For large t, the lower bound can be further simplified, using $(1-x) \approx e^{-x}$ for $x << 1$ and letting $\exp\{z\}$ represent e^z, as follows:

$$\begin{aligned}\Omega &\approx \prod_{i=0}^{t-1}\exp\left\{-i(2C_{\max})/(B\cdot 2^B)\right\} \\ &\approx \exp\left\{-C_{\max}t^2/(B\cdot 2^B)\right\}.\end{aligned} \tag{17}$$

Let Φ represent an upper bound on the probability that, given t sync cycles, there is an overlap between 2 or more keystream sequences. Hence, $\Phi = 1 - \Omega$ and

$$\Phi \approx C_{\max}t^2/(B\cdot 2^B) \tag{18}$$

where, again, we have used $(1-x) \approx e^{-x}$ for $x << 1$, which is validly applied since $C_{\max}t^2 << B\cdot 2^B$.

Consider now having M bits of ciphertext data available. This data will have about $t = M/(n + L\cdot B + E\{k\}) = M/(n + L\cdot B + 2^n - 1)$ sync cycles based on the assumption that k follows the geometric distribution. Hence, for $M = 1$ terabit, $n = 8$, $L = 10$, and $B = 128$, $t = 6.48\times10^8$ and an upper bound on the probability of overlapping keystreams can be computed to be $\Phi = 1.11\times10^{-19}$ based on the assumption that no sync cycle size exceeds $C_{\max} = 11528$, which is expected to be the case with a probability of about $1 - (1 - 4.25\times10^{-18})^t \approx 2.75\times10^{-9}$. It should be noted that the analysis undertaken to derive Φ is very conservative and is not necessarily a tight upper bound.

In conclusion, for a large amount of data, we can expect there to be very small probability of any significant amount of repeated keystream due to the resynchronization process of PSCFB mode with reasonable size parameters (eg. using the AES block size of 128 bits). Although PSCFB is a mode built from counter mode and cipher feedback mode for which formal security proofs exist, a formal proof of PSCFB security using methods such as those in [14] is not the focus of this work and we leave this as an open problem.

7 Comparison to Other Cipher Modes

The relative merits of PSCFB when compared to other conventional block cipher modes are illustrated in Table 2. The variable T_{max} represents the potential throughput of the block cipher achieved by a pipelined implementation of $L = 10$ stages. It is assumed that the stages of each pipeline architecture have similar critical path delay, resulting in the same clock period, t_{clk}, for any mode based on a pipelined implementation. Hence, in the table, for a mode (such as counter or PSCFB) that can be implemented using a pipeline architecture, the maximum throughput achievable is given by T_{max} in (1) and for a mode (such as OFB, CFB, or SCFB) that can only be implemented using an iterative architecture, the maximum throughput is given as $T_{max}/L = 0.1\ T_{max}$.

In the table, the encryption efficiency η represents the fraction of ciphertext bits produced by the mode, relative to the output bits produced by the block cipher operation. For OFB, CFB, and SCFB modes, the efficiency also represents the maximum throughput at which the cipher mode can be implemented relative to an iterative implementation of the block cipher. For CFB, the efficiency is determined based on the assumption that full synchronization is possible (that is, 1 ciphertext bit is fed back for every block cipher operation), resulting in an efficiency of $1/B$, relative to an iterative implementation. For SCFB, the efficiency is determined such that the queue sizes can be $M = 2B$ bits with no queue overflow. For counter and PSCFB modes, the efficiency represents the maximum throughput relative to a pipelined implementation.

For OFB and counter modes, EPF = 1 but these modes cannot recover from a synchronization loss. For CFB, SCFB, and PSCFB, the EPF and SRD are influenced by the block size. For CFB, EPF = $B/2 + 1$ and SRD = B, while for SCFB and PSCFB, these values are determined by simulation using the parameter values of $B = 128$, $n = 8$ (with sync pattern "10000000"), and, for PSCFB, $L = 10$.

Mode	Encryption Efficiency (η)	Pipeline?	Throughput (relative to T_{max})	EPF	SRD
OFB	100%	No	10%	1	∞
Counter	100%	Yes	100%	1	∞
CFB	0.78%	No	0.078%	65	128
SCFB	50%	No	5%	73	409
PSCFB	91%	Yes	91%	85	2106

Tab. 2. Comparision of modes ($B = 128$, $L = 10$ for pipelined modes).

From the table, many conclusions can be drawn. For example, while counter mode is capable of the highest throughput (marginally more than PSCFB), equivalent to the speeds achieved by a pipelined implementation, it is not self-synchronizing. Further, while CFB is able to resynchonize very quickly (in 128 bits), only one ciphertext bit is produced for every block cipher operation, resulting in an efficiency of only .78% and an extremely low throughput. Conventional SCFB can have its encryption efficiency increased by allowing for larger queue sizes and increasing n. However, because it cannot be effectively pipelined, its throughput cannot exceed 10% of T_{max}.

It is evident from the table that PSCFB provides an excellent tradeoff of properties: it has modest error propagation, is capable of resynchronizing, and can achieve very high throughputs.

8 Conclusion

In this paper, we have proposed a novel, highly efficient, self-synchronizing block cipher mode, targeted to high speed physical layer data streaming. The mode, referred to as pipelined statistical cipher feedback or PSCFB, combines cipher feedback and counter mode in a manner that enables an effective pipelined implementation of the block cipher. Implementations of the mode are able to achieve very high speeds, approaching pipelined implementations of counter mode, while allowing self synchro-

nization, a highly desirable feature of stream ciphers targeted to channels susceptible to synchronization losses due to causes such as timing errors. Implementations of PSCFB mode can be achieved with small queuing structures and low delay from plaintext reception to ciphertext transmission. Further, it is shown that the effect of errors, as measured by the error propagation factor, is modest and, although synchronization recovery delay is increased with more pipeline stages, this can be considered an acceptable trade-off to achieve the significantly higher throughput implied by more pipeline stages.

APPENDIX A

The pseudocode representation of PSCFB encryption for a B-bit block cipher with L pipeline stages is given in Figures A1 to A4, where P_i and C_i represent the i-th plaintext and ciphertext bits, respectively. Variable $X_0 \ldots X_{B-1}$ represents the input to the block cipher (i.e., the counter value) and is initialized to a value known to the encryption and decryption systems; variable $Y_0 \ldots Y_{B-1}$ represents the output of the block cipher, that is, the generated keystream. The variable $W_0 \ldots W_{n-1}$ represents the window of n bits that is currently being compared to the sync pattern, which is set to "10…00".

```
collecting_IV ← false
blackout_on ← false
W0…Wn-1 ← 0…0
X0…XB-1 ← initial value
pipeline_in ← X0…XB-1
do L−1 times
    execute 1 step of pipeline
    X0…XB-1 ← increment(X0…XB-1)
    pipeline_in ← X0…XB-1
i ← 0
do
    execute 1 step of pipeline
    Y0…YB-1 ← pipeline_out
    j ← 0
    new_scanning_period ← false
    insert_IV ← false
    do
        execute process_keystream_bit
    while j < B and not new_scanning_period
    execute update_counter
while true
```

Fig. A1. Main line of PSCFB encryption pseudo-code.

```
process_keystream_bit:
    Ci ← Pi ⊕ Yj
    if blackout_on then
        execute process_blackout
    else
        W0…Wn-2Wn-1 ← W1… Wn-1Ci
        if W0…Wn-1 = 10…00 then
            collecting_IV ← true
            blackout_on ← true
            m ← 0
    i ← i + 1
    j ← j + 1
```

Fig. A2. Pseudo-code to process a keystream bit.

The variable *pipeline_in* represents the input to the first stage of the block cipher pipeline. Initially, after the value of $X_0 \ldots X_{B-1}$ is set to the pipeline input, the pipeline is primed with the execution of $L-1$ steps. The execution of a pipeline step moves the data from one stage to the next stage in the pipeline. One pass of the outer "do" loop is used to execute one step of the pipeline and retrieve the block cipher output from the variable *pipeline_out*.

Function "increment" represents an incrementing of the counter value. Variable $Z_0 \ldots Z_{B-1}$ is used to collect the IV bits and flags *collecting_IV*, *blackout_on*, and *new_scanning_period* are used to control the collecting of the IV, the disabling of sync pattern scanning, and the reinitialization of the counter mode, respectively. The flag *insert_IV* is used to indicate that a new IV is available for the pipeline input. Variables *i*, *j*, and *m* are used as indices to keep track of the absolute bit number of the plaintext/ciphertext, the number of the bit within the block produced by the block cipher, and the number of the bit within the blackout period, respectively.

```
process_blackout:
    m ← m + 1
    if collecting_IV then
        Zm-1 ← Ci
        if m = B then
            collecting_IV ← false
            insert_IV ← true
    if m = L·B then
        blackout_on ← false
        new_scanning_period ← true
        W0…Wn-1 ← 0…0
```

Fig. A3.Pseudo-code to process the blackout period.

```
update_counter:
    if insert_IV then
        X0…XB-1 ← Z0…ZB-1
    else
        X0…XB-1 ← increment(X0…XB-1)
    pipeline_in ← X0…XB-1
```

Fig. A4.Pseudo-code to update counter input to block cipher.

The pseudo-code does not reflect some of the implementation considerations discussed in Section 4. Notably, the queuing processes are not illustrated. Rather, it is assumed that plaintext data is always available during each pass of the outer "do" loop so that the generated keystream can be immediately used to create ciphertext. In practice, as discussed in Section 4, when the number of bits in the plaintext queue is below a threshold, the pipeline system must be stalled until enough plaintext bits are available to be XORed with the keystream bits.

APPENDIX B – BOUNDS ON SRD AND EPF

In this appendix, we develop lower and upper bounds on SRD and EPF based on the format of "10…00" for the sync pattern and the geometric distribution for the size of the scanning period.

B.1 Lower Bound on SRD

For PSCFB mode, when a bit slip occurs in a sync cycle of size $n + L{\cdot}B + k$, the lower bound on the synchronization recovery delay at the receiver is given by

$$SRD(k) \geq \left\lfloor \frac{n+L\cdot B+k}{2} + n + L\cdot B \right\rfloor. \tag{19}$$

This is derived by considering that, for a given value of k, on average, a slip occurs halfway through a synchronization cycle and following the end of the synchronization cycle, it will take at least another sync pattern and blackout period before synchronization is recovered. The expression of (19) is a lower bound, as it is possible that a bit slip will cause a false synchronization at the receiver that results in a longer period to resynchronize.

To determine the overall SRD, we must average over all possible sync cycle sizes, as in

$$SRD \geq \sum_{k=0}^{\infty} P^*(k) SRD(k) \tag{20}$$

where $P^*(k)$ represents the probability that a bit slip occurs in a sync cycle of size $n + L{\cdot}B + k$. As discussed in [3], probability $P^*(k)$ can be determined from

$$P^*(k) = \frac{n+L\cdot B+k}{n+L\cdot B+E\{k\}} P(k) \tag{21}$$

where $P(k)$ represents the probability that a selected sync cycle is of size $n + L{\cdot}B + k$.

Combining (19) to (21) results in an expression for a lower bound on SRD as given in (10). As can be seen in the figures presented in Section 5, the lower bound gives results that are very close to the simulation results.

B.2 Upper Bound on SRD

Consider now an upper bound on SRD similar to the analysis in [3]. We consider two regions in which a slip may occur within a cycle of size $n + L{\cdot}B + k$. If a slip occurs such that the first bit following the slip is not within $n + L{\cdot}B$ bits of the sync pattern for the next cycle, then synchronization is lost until the valid sync pattern is detected for the next cycle and the subsequent blackout period completes. The probabililty that a slip occurs in this region is $k/(n + L{\cdot}B + k)$. When a slip occurs in the last $n + L{\cdot}B$ bits of the sync cycle, one must consider the possibility that the resulting bit sequence at the receiver could result in a false synchronization. The probability of a slip occuring in this region is given by $(n + L{\cdot}B)/(n + L{\cdot}B + k)$ and the likelihood that the slip results in a false sync is upper bounded by $n/2^n$. Subsequently, false synchronizations may occur at the receiver if the receiver misinterpretes a sync pattern appearing within the blackout period to be a proper sync pattern. This may result in loss of sync for several sync cycles. However, we can say with certainty that synchronization must be regained when a sync cycle is encountered for which $k \geq n + L{\cdot}B$. (In reality, sync is likely to be regained much more quickly than this.)

As a result of these considerations, we can derive an upper bound on SRD to be

$$SRD < \sum_{k=0}^{\infty} \left[\left(2(n+L\cdot B) + \frac{k}{2} \right) \frac{k}{n+L\cdot B+k} + \left(\frac{3}{2}(n+L\cdot B) + \frac{n}{2^n}\mu\cdot\lambda \right) \frac{n+L\cdot B}{n+L\cdot B+k} \right] P^*(k) \tag{22}$$

which can be simplified to (11). Note that $\lambda = (1 - 1/2^n)^{-(n+LB)}$ is an upper bound on the expected number of cycles until resync is achieved, determined by conservatively ensuring $k \geq n + L{\cdot}B$ for resynchronization. Comparing the upper bound of SRD to simulation results from Section 5, it is obvious that the upper bound improves (i.e., gets tighter) for decreasing L and increasing n.

B.3 Lower Bound on EPF

In general, for PSCFB mode, errors at the receiver can be considered straighforwardly for two cases as follows. In the first case, consider the occurrence of an error in the sync pattern or IV block. For a subsequent scanning period of k bits in size, the resulting EPF satisfies

$$EPF_{sync/IV}(k) \geq 1 + \frac{1}{2}\left[k + n + L\cdot B\right]. \tag{23}$$

This is derived by considering that one bit in error in the channel in the sync pattern or IV block portion of the sync cycle will result directly in one bit in error after decryption, followed by a loss of synchronization (during which half the bits are in error) from the start of the scanning period, until the end of the following blackout period for a total of $k + n + L{\cdot}B$ bits. However, (23) is in fact a lower bound because it is possible that, when a receiver loses synchronization, a false resynchronization occurs (eg. a legitimate sync pattern is lost due to a bit error and a sequence during the legitimate blackout period is misinterpreted to be a sync pattern) and, as a result, it takes longer than implied above to resynchronize.

In the second case, consider the occurrence of a bit error in the channel during the blackout (excluding the IV) or scanning periods. The resulting EPF satisfies

$$EPF_{BO/CTR} \geq 1. \tag{24}$$

The equality of this expression corresponds to a bit error which occurs in the blackout period (but not in the IV) or the scanning period and causes one bit error at the receiver, such that it does not cause a false sync pattern to occur in the ciphertext. The equality does not account for the circumstance that a bit error causes a false sync pattern resulting in the receiver improperly assuming a resynchronization, which would result in EPF above the lower bound in (24).

Overall, weighting each case by its probability of occurrence, the lower bound on EPF is given by

$$EPF \geq \sum_{k=0}^{\infty} P^*(k) \left[\frac{n+B}{k+n+L\cdot B} EPF_{sync/IV}(k) + \frac{(L-1)\cdot B+k}{k+n+L\cdot B} \cdot EPF_{BO/CTR} \right] \tag{25}$$

where $P^*(k)$ represents the probability that a bit error occurs in a sync cycle of size k. Substituting (23) and (24) into (25) results in the lower bound on EPF being easily calculated as given in (12).

As can be seen from the figures in Section 5, for large n, the lower bound is strongly similar to EPF as determined through simulations.

B.4 Upper Bound on EPF

Following the analysis in [3], the upper bound on EPF is developed considering the effects of errors for 4 different scenarios, identified in Table B.1.

The probability that a bit error belongs to case 1 is

$$P_1(k) = \frac{n + L \cdot B}{n + L \cdot B + k} \tag{26}$$

and the resulting expected number of bit errors is

$$\delta_1(k) = 1 + \frac{n + L \cdot B + k}{2}. \tag{27}$$

Case	Scenario	Effect
1	Error in n+B bits of sync + IV.	Sync lost for entire cycle.
2	Error in blackout period (minus IV) or scanning period such that no sync pattern is falsely generated.	One bit error in recovered plaintext.
3	Error in scanning period such that false sync generated in first k – (n + LB) bits of scanning period.	$i/2$ bit errors generated, where i is number of bits between end of false blackout period and end of next legitimate blackout period.
4	Error in scanning period such that false sync generated in last n + LB bits of scanning period.	Next sync pattern will be missed because it is part of false blackout causing ½ bits in error until next detected sync + blackout.

Tab. B.1. Effects of different error scenarios

For case 2, the expected number of bit errors is $\delta_2(k)$ = 1. Since the probabilty of this case is quite high, for simplicity we assume that the probability that case 2 occurs is bounded by $P_2(k) < 1$.

In order to determine the probabilities for cases 3 and 4, we make use of the upper bound on the probability that a bit error results in a sequence of bits identical to the sync pattern, given by $n/(2^n - 1)$. Hence, case 3 occurs with a probability upper bounded by

$$P_3(k) < \frac{n}{2^n - 1} \cdot \frac{k}{n + L \cdot B + k}. \tag{28}$$

The expected number of bit errors caused in case 3 is upper bounded as in

$$\delta_3(k) < \frac{3}{4}(n + L \cdot B) + \frac{k}{4} \tag{29}$$

where the bound arises from the bits in the remainder of the synchronization cycle and the bits in the sync pattern and blackout period associated with the next sync cycle.

The upper bound on the probability of case 4 occurs is

$$P_4(k) < \frac{n}{2^n - 1} \cdot \frac{n + L \cdot B}{n + L \cdot B + k}. \tag{30}$$

For this scenario, we assume that at least the next sync pattern is missed and it could be several sync cycles to recover synchronization. Hence, the expected number of errors for case 4 can be large and is upper bounded by

$$\delta_4(k) < \frac{3(n + L \cdot B)}{4} + \frac{\mu \cdot \lambda}{2} \tag{31}$$

where μ and λ are defined as for the bounds on SRD. The first term is an upper bound on the expected number of bit errors between the end of the false blackout period and the end of the blackout period of the next sync cycle.

An estimate of the upper bound on the error propagation factor can be determined by noting that

$$EPF \approx \sum_{i=1}^{4} \sum_{k=0}^{\infty} P^*(k) \cdot P_i(k) \delta_i(k) \tag{32}$$

resulting in (13). The upper bound on EPF is found to be best for small L and for large n. This can be observed in the results presented in Section 5.

ACKNOWLEDGMENT

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

[1] National Institute of Standards and Technology, *Advanced Encryption Standard*, Federal Information Processing Standards (FIPS) Publication 197, Nov. 2001.

[2] O. Jung and C. Ruland, "Encryption with statistical self-synchronization in synchronous broadband networks", *Proceedings of Cryptographic Hardware and Embedded Systems - CHES '99*, Lecture Notes in Computer Science, vol. 1717, Springer, pp. 340-352, 1999.

[3] H.M. Heys, "Analysis of the Statistical Cipher Feedback Mode of Block Ciphers", *IEEE Tranactions on Computer Engineering*, vol. 52, no. 1, Jan. 2003.

[4] U.M. Maurer, "New Approaches to the Design of Self-Synchronizing Strean Ciphers", *Advances in Cryptology - Proceedings of Eurocrypt '91*, Lecture Notes in Computer Science, vol. 547, Springer, pp. 458-471, 1991.

[5] J. Daemen and P. Kitsos, "The Self-Synchronizing Stream Cipher MOUSTIQUE", *New Stream Cipher Designs*, Lecture Notes in Computer Science, vol. 4986, Springer, pp. 210-223, 2008.

[6] eSTREAM project website: www.ecrypt.eu.org/stream

[7] S. Babbage, C. De Canniere, A. Canteaut, C. Cid, H. Gilbert, T. Johansson, M. Parker, B. Preneel, V. Rijmen, and M. Robshaw, "The eSTREAM Portfolio", available at the eSTREAM project website: www.ecrypt.eu.org/stream, Apr. 15, 2008.

[8] W. Stallings, *Cryptography and Network Security*, Pearson Prentice Hall, 4th ed., 2006.

[9] I. Verbauwhede, P. Schaumont, and H. Kuo, "Design and Performance Testing of a 2.29 GB/s Rijndael Processor", *IEEE Journal of Solid-State Circuits*, vol. 38, no. 3, pp. 569-572, Mar. 2003.

[10] A. Hodjat and I. Verbauwhede, "Area-Throughput Trade-offs for Fully Pipelined 30 to 70 Gbits/s AES Processors", *IEEE Transactions on Computers*, vol. 55, no. 4, pp. 366-372, Apr. 2006.

[11] K. Burda, "Resynchronization Interval of Self-Synchronizing Modes of Block Ciphers", *International Journal of Computer Science and Network Security*, vol. 7, no. 10, pp. 8-13, Oct. 2007.

[12] A. Alkassar, A. Geraldy, B. Pfitzmann, and A.-R. Sadeghi, "Optimized Self-Synchronizing Mode of Operation", *Fast Software Encryption (FSE 2001)*, Lecture Notes in Computer Science, vol. 2355, Springer, pp. 78-91, Apr. 2001.

[13] K. Burda, "Modification of OCFB Mode for Fast Data Links", *International Journal of Computer Science and Network Security*, vol. 7, no. 12, pp. 228-232, Dec. 2007.

[14] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, "A Concrete Security Treatment of Symmetric Encryption: Analysis of DES Modes of Operation", *Proceedings of 38th Annual Symposium on*

Foundations of Computer Science, IEEE, pp. 394-403, 1997.

[15] J. Daemen, J. Lano, and B. Preneel, "Chosen Ciphertext Attack on SSS", available at eSTREAM project website: www.ecrypt.eu.org/stream, 2005.

[16] A. Joux and F. Muller, "Chosen-Ciphertext Attacks Against MOSQUITO", *Fast Software Encryption - FSE 2006*, Lecture Notes in Computer Science, vol. 4047, Springer, pp. 390-404, 2006.

Howard M. Heys obtained a BESc degree in electrical engineering in 1984 from the University of Western Ontario in London, Ontario, Canada, and a PhD degree in computer engineering in 1994 from Queen's University, Kingston, Ontario, Canada. He worked for several years as a software designer in Ottawa and Toronto. Dr. Heys is now a professor of electrical and computer engineering at Memorial University of Newfoundland. His current research interests include cryptography, digital hardware design, and communication networks.

Liang Zhang obtained a BEng in 2002 from Tianjin University, Tianjin, China. Subsequently, he obtained MASc and MEng degrees in computer engineering from Memorial University, Newfoundland, Canada. He is currently working as a hardware design engineer in Avalon Microelectronics.