

# Analysis of the Statistical Cipher Feedback Mode of Block Ciphers

Howard M. Heys

**Abstract**—In this paper, we examine a recently proposed mode of operation for block ciphers which we refer to as statistical cipher feedback (SCFB) mode. SCFB mode configures the block cipher as a keystream generator for use in a stream cipher such that it has the property of statistical self-synchronization, thereby allowing the stream cipher to recover from bit slips in the communication channel. Statistical self-synchronization involves feeding back ciphertext to the input of the block cipher similar to the conventional cipher feedback (CFB) mode, except that the feedback only occurs when a special synchronization pattern is recognized in the ciphertext. In the paper, we examine the efficiency, resynchronization, and error propagation characteristics of SCFB and compare these to conventional modes such as CFB and output feedback (OFB). In particular, we study these characteristics of SCFB as a function of the synchronization pattern size. As well, we examine implementation issues of SCFB, focusing on the buffer requirements and resulting delay for a practical realization of the cipher. We conclude that SCFB mode can be used to provide practical, efficient, self-synchronizing implementations for stream ciphers. In particular, SCFB mode is best used in circumstances where slips are a concern and where implementation efficiency is a high priority in comparison to encryption latency.

**Index Terms**—Cryptography, stream ciphers, block cipher modes, synchronization, error propagation.

## 1 INTRODUCTION

IN this paper, we discuss a structure for self-synchronizing stream ciphers, recently proposed in [1]. Stream ciphers are used to encrypt one symbol, typically one bit, at a time. They are usually used when error propagation must be minimized or when the communication channel suffers from periodic slips. The basic form of a stream cipher involves the generation of a *keystream*—a keyed, pseudorandom, unpredictable sequence of bits—that is XORed bit by bit with the plaintext to generate the ciphertext at the transmitter [2]. At the receiver, the plaintext is recovered by generating the identical keystream such that it is exactly synchronized with the received ciphertext stream. Hence, the XOR of the keystream bits and received ciphertext bits produces the original plaintext bits.

We shall concern ourselves with stream ciphers which are derived from block ciphers such as the Data Encryption Standard (DES) [3] and the Advanced Encryption Standard (AES) [4]. Unlike stream ciphers, which conceptually operate on bits individually, block ciphers operate on a fixed size block of plaintext bits to produce a block of ciphertext bits. When configuring a block cipher for use as a stream cipher, the keystream is generated by the output of the block cipher. There are several conventional modes of operation of block ciphers that allow their use as stream ciphers including output feedback (OFB) mode and cipher feedback (CFB) mode [2]. In this work, we focus on an unconventional mode which we shall refer to as *statistical cipher feedback (SCFB) mode*. SCFB mode has been proposed

[1] to provide physical layer security for a SONET/SDH environment and is suitable for many other applications as well. It has the benefits of being self-synchronizing and yet being more efficient in its implementation than conventional cipher feedback mode. In this paper, the characteristics of SCFB mode are thoroughly examined and its merits are discussed.

## 2 BACKGROUND

There are several conventional block cipher modes of operation that may be applied to derive a stream cipher, each with its own advantages and disadvantages. We briefly review the important modes of output feedback and cipher feedback here. In our notation, we let  $B$  represent the block size in bits of the block cipher. For example, for DES,  $B = 64$  and, for AES,  $B = 128$ .

*Output feedback (OFB) mode*, a standardized mode of operation for block ciphers such as DES [5], generates the keystream by directly feeding back the  $B$ -bit output of the block cipher to the input. An implementation of OFB is parameterized by  $j$ ,  $1 \leq j \leq B$ , where every block cipher operation produces  $j$  bits of keystream which can be XORed with  $j$  bits of plaintext to produce  $j$  bits of ciphertext. For efficiency purposes, it is convenient to let  $j = B$ , thereby making use of all possible  $B$  output bits of the block cipher for every block cipher operation. This basic configuration is illustrated in Fig. 1. The primary advantage of output feedback mode is that error propagation is minimized. In fact, a single bit error in the ciphertext in the communication channel results in only a single bit error in the recovered plaintext.

The most significant disadvantage of OFB is that the system relies on the maintaining of synchronization between the transmitter and receiver. For example, if a slip

• The author is with Electrical and Computer Engineering, Faculty of Engineering and Applied Science, Memorial University of Newfoundland, St. John's, NF, Canada A1B 3X5. E-mail: howard@enr.mun.ca.

Manuscript received 10 Aug. 2001; revised 26 Feb. 2002; accepted 22 Apr. 2002.

For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 114746.

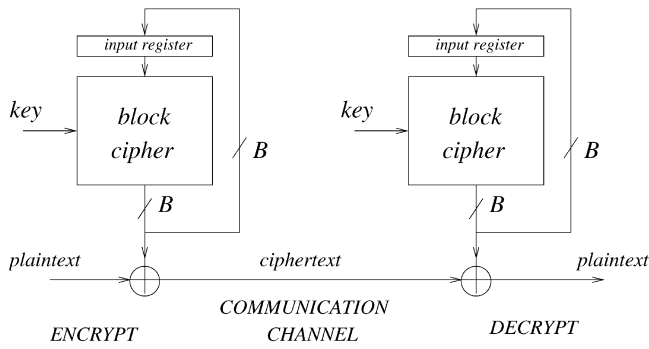


Fig. 1. OFB mode.

occurs (i.e., one or more bits are eliminated from the received ciphertext stream), synchronization loss will occur between the transmitter and receiver and half the bits following the slip are expected to be in error until synchronization is recovered. Resynchronization can be achieved by periodically sending an *initialization vector (IV)* from the transmitter to the receiver through the signaling channel of the communication system. Obviously, the price of such a scheme involves extra messaging overhead and the associated delays while synchronizing. As well, the rate at which synchronization messages are sent must balance the overhead of sending such messages frequently with the penalty of losing synchronization for a long period of time should the messages be sent too infrequently. However, OFB (not considering the resynchronization messaging overhead) can be implemented as efficiently as straight block encryption by having  $B$  bits of keystream produced from one block cipher output.

*Cipher feedback (CFB) mode* [5] allows for automatic resynchronization should slips occur in the communication channel and, hence, CFB stream ciphers fall into the category of self-synchronizing stream ciphers. There are several parameters that may be set for an implementation of CFB, however, a typical application would have the input to the block cipher driven by ciphertext data which is fed back into a shift register at the input of the block cipher in groups of  $m$  ( $\leq B$ ) bits at a time, as shown in Fig. 2. The plaintext is encrypted by XORing  $m$  bits with  $m$  bits of the block cipher output.

Since the input to the block cipher is being generated from the ciphertext, which is available to both ends of the communication, it is possible to recover from slips using CFB. For example, if any single or multiple bit slip occurs, the CFB cipher with  $m = 1$  can recover synchronization since the next  $B$  bits will be shifted into the register at the input to the block cipher and, at this point, the receiver will again be synchronized with the transmitter. Resynchronization therefore requires only  $B$  bits in CFB mode with  $m = 1$ .

Unfortunately, the self-synchronization property achieved by the CFB mode is costly in terms of implementation efficiency. For CFB with  $m = 1$ , only one keystream bit is generated from a  $B$ -bit block cipher output and, hence, the cipher is only capable of operating at  $1/B$  times the rate of block encryption and, consequently, can only be implemented at  $1/B$  times the best rate of OFB. This can be improved by increasing  $m$ , but, if  $m > 1$  and a single bit

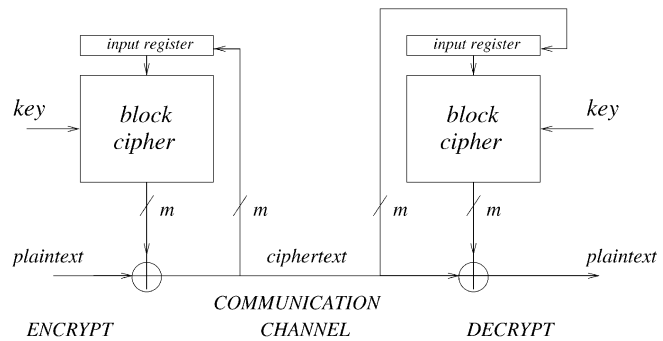


Fig. 2. CFB mode.

slip occurs, the input to the block cipher at the receiver will become misaligned and resynchronization will not occur. In fact, slips must occur as multiples of  $m$  bits or resynchronization will not occur. Hence, usually for CFB mode,  $m = 1$  is the desirable configuration.<sup>1</sup>

Finally, it should be noted that the error propagation advantage of OFB is no longer applicable to CFB mode. This occurs because, at the receiver, a bit error must work its way through the shift register at the input to the block cipher. As a result, for CFB mode with  $m = 1$ , a single bit error in the communication channel will result in the corrupted bit plus the next  $B$  bits being randomly decrypted due to the receiver's corrupted keystream. So, a bit error is expected to result, on average, in  $B/2 + 1$  errors in the recovered plaintext.

### 3 DESCRIPTION OF SCFB

In [1], the concept of statistical self-synchronization is proposed as a mechanism to provide physical layer security for a SONET/SDH environment.<sup>2</sup> Operation of the mode is illustrated in Fig. 3, where  $E$  represents the block cipher encryption operation with block size  $B$  and both the encryption and decryption of SCFB mode are illustrated. Essentially, the concept of statistical self-synchronization involves a hybrid of OFB<sup>3</sup> and CFB modes: The cipher operates in OFB mode, while scanning the ciphertext for a special *sync pattern* of  $n$  bits in length. When this pattern is recognized, the next  $B$  bits are stored for a new initialization vector (IV) and, after all  $B$  bits have been collected, the input register for the block cipher is loaded with the new IV. The cipher then proceeds in OFB mode until the next  $n$  bit sync pattern is received. During the collection of  $B$  bits for the new IV, the sync pattern scanning is turned off so that any  $n$  bits matching the sync pattern are ignored until the IV collection phase is complete. This process follows for both encryption and decryption and, since both the transmitter

1. There are cases where  $m > 1$  makes sense. For example, CFB with  $m = 8$  can be used to encrypt an asynchronous communication link so that an 8-bit character can be encrypted with each block cipher output. Here, CFB mode is used to recover from losses of synchronization due to asynchronous characters being lost, as opposed to individual bit slips.

2. This scheme appears to have also been invented earlier and is referred to in [6].

3. SCFB can also be implemented as a hybrid of counter mode [2] and CFB mode. However, in this paper, we focus our description on the OFB-based configuration only.

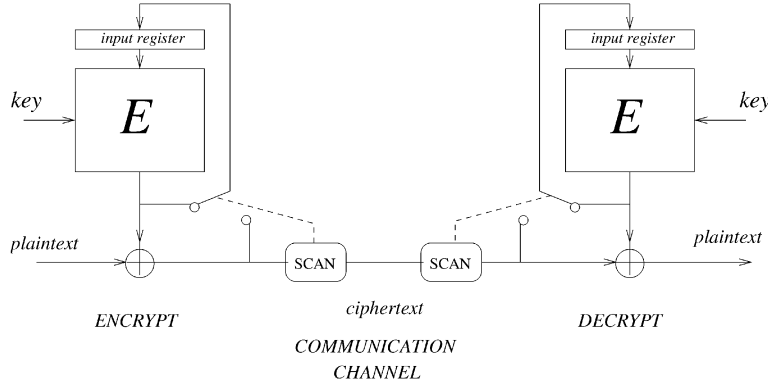


Fig. 3. SCFB mode.

and receiver are examining the ciphertext, synchronization is achieved.

To provide enough detail for precise clarity of the operation of SCFB mode, a pseudocode representation for encryption at the transmitter using SCFB is given in Fig. 4. The sync pattern is given by  $Q_0 \dots Q_{n-1}$  and  $W_0 \dots W_{n-1}$  represents the window of  $n$  bits that is currently being compared to the sync pattern. In order for the algorithm, as presented, to work with the initialization of  $W_0 \dots W_{n-1}$  to all zeros,  $Q_0$  must be 1. The function  $E_K(\cdot)$  represents the

```

loading_IV ← false
 $X_0 \dots X_{B-1} \leftarrow$  initial value
 $W_0 \dots W_{n-1} \leftarrow 0 \dots 0$ 
 $j \leftarrow 0$ 
do
   $Y_0 \dots Y_{B-1} \leftarrow E_K(X_0 \dots X_{B-1})$ 
  new_IV ← false
   $i \leftarrow 0$ 
  do
     $C_{j+i} \leftarrow P_{j+i} \oplus Y_i$ 
    if loading_IV then
       $Z_k \leftarrow C_{l+k}$ 
       $k \leftarrow k + 1$ 
      if  $k = B$  then
        loading_IV ← false
        new_IV ← true
         $X_0 \dots X_{B-1} \leftarrow Z_0 \dots Z_{B-1}$ 
         $W_0 \dots W_{n-1} \leftarrow 0 \dots 0$ 
      else
         $W_0 \dots W_{n-2} W_{n-1} \leftarrow W_1 \dots W_{n-1} C_{j+i}$ 
        if  $W_0 \dots W_{n-1} = Q_0 \dots Q_{n-1}$  then
          loading_IV ← true
           $l \leftarrow j + i + 1$ 
           $k \leftarrow 0$ 
         $i \leftarrow i + 1$ 
        if  $i = B$  and not new_IV then
           $X_0 \dots X_{B-1} \leftarrow Y_0 \dots Y_{B-1}$ 
        while  $i < B$  and not new_IV
           $j \leftarrow j + i$ 
        while true

```

Fig. 4. SCFB pseudocode.

block cipher encryption (using key  $K$ ).  $Z_0 \dots Z_{B-1}$  is used to collect the IV bits. The flags *loading\_IV* and *new\_IV* are used to indicate that IV is currently being collected (and sync pattern scanning is therefore suspended) and collection of IV has just completed, respectively. Note that the initial block cipher input  $X_0 \dots X_{B-1}$  is given an initial value known to both the transmitter and receiver at the beginning of the communication.

From the pseudocode, it may be seen that the encryption of a bit would encounter a significant delay whenever a new block encryption is required since  $E_K$  can be expected to take much longer than any other operation in the algorithm. Hence, in practice, for an efficient synchronous system, an implementation would need a buffer in order to ensure that plaintext bits can be accepted at a uniform rate and ciphertext bits can be produced at a uniform rate at the output of the encryption process.

Since both the transmitter and receiver are using recognized bits within the ciphertext as a cue to resynchronize the stream cipher, SCFB mode is capable of self-synchronization and will clearly perform better in an environment where slips occur than OFB mode. Also, although individual bit errors will typically only cause one bit error if the bit is not part of the sync pattern or the initialization vector, there is the possibility that a bit error will cause a synchronization to be missed, an incorrect IV to be used, or a false synchronization to be detected at the receiver. In these cases, a single bit error in the communication channel will result in many bit errors at the output of the decryption as synchronization will be lost until the next sync pattern is properly detected. Hence, clearly, the error propagation characteristics of SCFB will be worse than OFB mode.

It should be noted that a mode similar in nature to SCFB was recently proposed in [7] and is referred to as Optimized Cipher Feedback (OCFB). As in SCFB, OCFB uses recognition of a synchronization pattern in the ciphertext to resynchronize the states of the transmitter's and receiver's keystream generators. Hence, the characteristics of SCFB and OCFB are very similar and, in our work, we focus on SCFB.

Although the SCFB mode was proposed in [1], the cipher characteristics were not fully examined. Notably, the effect of the sync pattern size on the cipher properties was not discussed. In the following sections, we shall consider the

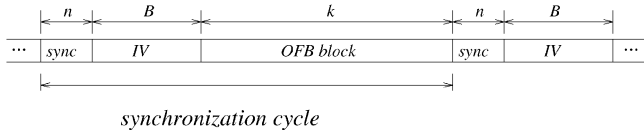


Fig. 5. Synchronization cycle.

efficiency, resynchronization, and error propagation characteristics of SCFB and, in particular, make a comparison to other conventional cipher modes. Further, we shall discuss the implementation issues associated with a practical realization of the SCFB cipher mode and briefly comment on the security of the scheme in relation to the likelihood of the keystream repeating.

#### 4 THEORETICAL EFFICIENCY

The principal advantage of implementing SCFB versus conventional CFB is that the efficiency (and, hence, the potential speed) of the implementation can approach that of straight block encryption, depending on the sync pattern size. Letting  $D$  represent the number of bits transmitted, we can define the *theoretical efficiency* for a stream cipher based on a block cipher core as:

$$\eta = \lim_{D \rightarrow \infty} \frac{D/B}{E\{\#\text{ block cipher operations for } D \text{ bits}\}}, \quad (1)$$

where  $E\{\cdot\}$  represents the expectation operator. The numerator represents the number of block cipher operations required for straight block encryption and the denominator represents the expected number of block cipher operations required in SCFB mode (or other mode of interest). Hence, the theoretical efficiency is essentially a measure of the rate at which the stream cipher can encrypt in comparison to block encryption. In reality, the theoretical efficiency represents an upper bound on the efficiency: As we shall see in Section 7, real implementations place constraints on the system so that the practical efficiency (which we shall refer to as *full-queue efficiency*) is marginally smaller.

For OFB mode, when all  $B$  bits are used in the XOR operation,  $\eta = 1$  and, for conventional CFB with  $m = B$ ,  $\eta = 1$ . However, if we are to be guaranteed to correct slips of any number of bits, conventional CFB must operate with  $m = 1$  and, in this case,  $\eta = 1/B \ll 1$ . So, conventional CFB is very inefficient in comparison to block encryption.

Consider now SCFB. We can assume that, for SCFB, the ciphertext bits transmitted in the communication channel can be categorized as illustrated in Fig. 5, where it is clear that some bits belong to the sync pattern ( $n$  bits), some belong to the subsequent IV ( $B$  bits), and the remaining bits, which we shall refer to as the *OFB block*, occur between the end of the IV and the beginning of the next sync pattern. We shall refer to the set of bits from the beginning of the sync pattern to the beginning of the next sync pattern as a *synchronization cycle* and, hence, a synchronization cycle consists of  $n + B + k$  bits, where  $k$  is the size of the OFB block.

The size  $k$  of the OFB block is variable and dependent on the position of the next sync pattern. Since we assume that the block cipher used in the SCFB configuration displays

strong randomness properties (or else it would be insecure),  $k$  is a random variable with a probability distribution determined by assuming that each bit of ciphertext is equally likely to be a 0 or 1 and that each bit is independent. Strictly, the distribution of  $k$  is dependent on the sync pattern used (e.g., 11...11, or 10...00, etc.).

The value of  $k$  is determined by the process of taking samples of  $n$  bits and comparing each to the sync pattern where  $k$  is the number of samples taken before the sync pattern is found. Now, if we assume that each  $n$ -bit sample is independent, then the value of  $k$  follows the geometric distribution where the probability that the sync pattern occurs in a sample is  $1/2^n$ . In this case, we have the probability distribution for  $k$  given by

$$P(k) = (1 - 1/2^n)^k \cdot 1/2^n, \quad (2)$$

the expected value of  $k$  given by

$$E\{k\} = 2^n - 1, \quad (3)$$

and the second moment of  $k$  given by

$$E\{k^2\} = 2^{2n+1} - 3 \cdot 2^n + 1. \quad (4)$$

Based on the geometric distribution, we can define the average synchronization cycle size to be represented by  $\mu$ , where

$$\mu = n + B + 2^n - 1. \quad (5)$$

For SCFB mode where  $k$  actually represents the size of the OFB block as determined by the next occurrence of the sync pattern, consecutive samples overlap in  $n - 1$  bits and, hence, samples are not independent and  $k$  does not follow the geometric distribution exactly. However, we have verified experimentally that, for most sync patterns, the probability distribution of  $k$  can be approximated by the geometric distribution. We therefore use this distribution in our development and defer a more detailed discussion of the effect of the selection of the sync pattern on the distribution of  $k$  to the Appendix.

Consider now the scenario for Fig. 5 where  $k + n + B$  is not a multiple of  $B$ . In this case, after the second IV is collected, the input register of the block cipher will be loaded and a new block cipher output will be produced after a block cipher operation that is used for encrypting only a subset of a full  $B$ -bit block of plaintext. For example, let  $k + n + B = \alpha B + \gamma$ , where  $\alpha$  and  $\gamma$  are integers and  $\gamma < B$ . Hence, the block cipher must execute  $\alpha + 1$  block encryptions from the beginning of the OFB block to the end of the second IV, producing  $(\alpha + 1)B$  bits, to encrypt only  $\alpha B + \gamma$  plaintext bits (i.e., only  $\gamma$  bits of the last block cipher output are used in the XOR). Therefore, in SCFB mode, the block cipher must be run at a rate slightly greater than for straight block encryption and a buffer must be used to accommodate scenarios where only partial outputs of the block cipher are used by the XOR operation.

Using (1) as the basic definition, it is possible to define the theoretical efficiency of SCFB as

$$\eta = \frac{E\{\text{sync cycle size}\}/B}{E\{\#\text{ block cipher operations per sync cycle}\}}. \quad (6)$$

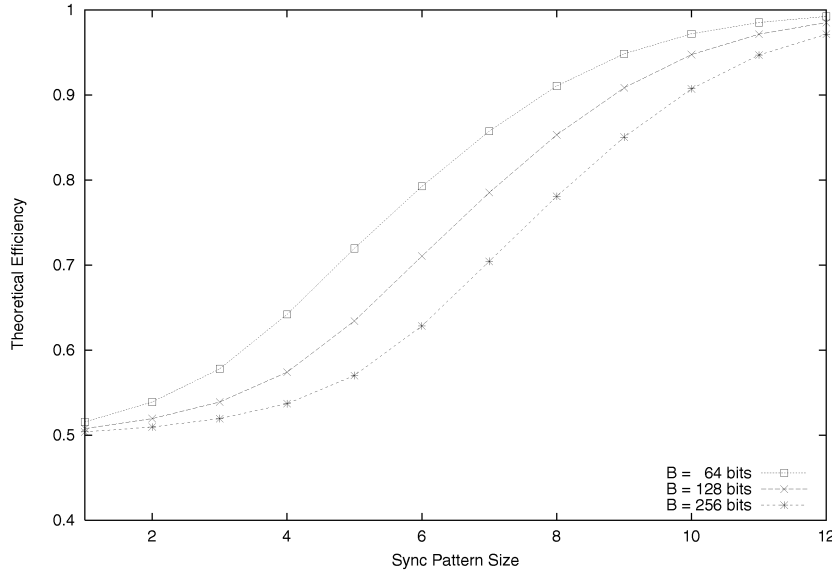


Fig. 6. Theoretical efficiency vs. sync pattern size.

This leads to

$$\eta = \frac{\mu/B}{\sum_{k=0}^{\infty} P(k) \cdot \lceil (k+n+B)/B \rceil}. \quad (7)$$

Consider now the computation of the denominator of (7). The expression for the denominator can be rewritten to be

$$\text{denom} = \sum_{k=0}^{\infty} P(k) \cdot \left\lceil \frac{k+n}{B} \right\rceil + 1. \quad (8)$$

Further, considering the effect of the ceiling operator, it may be shown that

$$\text{denom} = d_A + d_B + 1, \quad (9)$$

where

$$d_A = \sum_{k=0}^{B-n} P(k) \quad (10)$$

and

$$d_B = \sum_{j=2}^{\infty} \sum_{k=(j-1)B-n+1}^{jB-n} j \cdot P(k). \quad (11)$$

Recall that  $P(k)$  is given by (2). Letting  $\sigma = (1 - 1/2^n)$  and noting that  $d_A$  is represented by the sum of a geometric series, it is easily derived that

$$d_A = 1 - \sigma^{B-n+1}. \quad (12)$$

As well, it can be shown that

$$\begin{aligned} d_B &= \sum_{j=2}^{\infty} j \cdot (\sigma^{(j-1)B-n+1} - \sigma^{jB-n+1}) \\ &= (1 - \sigma^B) \sigma^{-B-n+1} \sum_{j=2}^{\infty} j \sigma^{jB} \\ &= (1 - \sigma^B) \sigma^{-B-n+1} \left[ \frac{\sigma^B}{(1 - \sigma^B)^2} - \sigma^B \right]. \end{aligned} \quad (13)$$

Now, substituting for  $d_A$  and  $d_B$  leads to the denominator of (7) given by

$$\text{denom} = 2 + \frac{\sigma^{B-n+1}}{1 - \sigma^B}. \quad (14)$$

Finally, the efficiency of SCFB mode can be straightforwardly computed from  $n$  and  $B$  using

$$\eta = \frac{\mu/B}{2 + \frac{(1-1/2^n)^{B-n+1}}{1-(1-1/2^n)^B}}. \quad (15)$$

We have computed the theoretical efficiency  $\eta$  for block sizes of 64, 128, and 256 bits and these are plotted in Fig. 6 as a function of the sync pattern size. It is obvious from the graph that, as  $n$  increases, the efficiency of the implementation increases and, for large  $n$ , the efficiency approaches 100 percent, implying that the stream cipher can be run at a rate very nearly equivalent to the speed of block encryption. In fact, a cipher can be run at a rate approaching  $\eta$  times the block encryption rate with a suitably large enough buffer to account for the scenario of several resynchronizations within a short time frame. This is discussed in detail in Section 7.

Note that all efficiencies for SCFB mode are greater than 50 percent. This occurs because at least one full block is used in each synchronization cycle since  $B$  bits are associated with the IV. For small values of  $n$ , SCFB is significantly less efficient than straight block encryption. For example, for  $n = 1$ , the

cipher is resynchronizing after an expected OFB block size of 1 and, as a result, virtually every second block cipher output is used only for a small number of bits in the XOR operation. Hence, the efficiency is only about 50 percent. For conventional CFB mode with  $m = 1$  to accommodate sync recovery from a slip of any number of bits, the efficiency would be  $< 2\%$  and  $< 1\%$  for a block size of  $B = 64$  and  $B = 128$ , respectively. In contrast, for OFB mode, efficiencies are 100 percent when all  $B$  bits of block cipher output are used in the keystream.

Also from the graph, it is clear that SCFB mode applied to a block cipher with a large block size suffers in efficiency. For example, for  $n = 8$ , the theoretical efficiencies are 91.1 percent, 85.3 percent, and 78.1 percent for 64, 128, and 256 bit blocks, respectively. However, these efficiencies are still many times more than the conventional CFB mode and, in fact, the ratio of SCFB efficiency to conventional CFB efficiency increases as  $B$  increases.

## 5 RESYNCHRONIZATION

A fundamental requirement of a self-synchronizing stream cipher is that resynchronization occurs quickly to minimize the corruption of data due to a sync lost condition. Conventional CFB mode with  $m = 1$ , for example, will synchronize within  $B$  bits of the end of a slip of one or more bits. Resynchronization delay for an OFB cipher relying on signaling messages to provide synchronization information will generally be very large since synchronization usually relies on a low rate signaling channel and synchronization messages are exchanged relatively infrequently.

In this section, we examine the resynchronization properties of SCFB mode. We shall consider the metric of interest to be the *synchronization recovery delay* (SRD), defined as the expected number of bits following a sync loss due to a slip before synchronization is regained. It is important to note that, in our analysis, when we refer to the occurrence of a slip, we are referring to the position representing the termination of the bits lost in the slip. Hence, we consider SRD to represent the number of bits for which the receiver is out of sync following the termination of the slip. That is, SRD does not include the bits that are lost directly due to the slip and no explicit assumptions are made about the number of bits lost in the slip.

In our work, we will present lower and upper bounds on SRD and experimental measurements of SRD for varying values of  $n$ . Our analysis shows that, except for very small sync pattern sizes (i.e.,  $n \leq 4$ ), SRD is approximately  $2^n$ , a value that can be significantly larger than SRD for CFB for large  $n$ . This fact encourages the use of modest size values of  $n$  in SCFB mode.

### 5.1 Lower Bound on SRD

We begin by considering a lower bound on the sync recovery delay. Assume that a slip randomly occurs such that there are no other slips in the synchronization cycle in which the slip terminates. The probability that the slip occurs within a synchronization cycle of size  $n + B + k$  is given by

$$P^*(k) = \frac{(n + B + k) \cdot P(k)}{\mu}, \quad (16)$$

where  $P(k)$  and  $\mu$  are given by (2) and (5), respectively. Assuming that the receiver resynchronizes at the next sync pattern, i.e., at the end of the next IV, it will take an average of  $(n + B + k)/2 + n + B$  bits to resynchronize. This is determined by the average position of a slip within the synchronization cycle plus the  $n + B$  bits required at the beginning of the next synchronization cycle to resynchronize. Now, if we consider the average over all synchronization cycle sizes, then the synchronization recovery delay is lower bounded as in

$$\begin{aligned} SRD &> \sum_{k=0}^{\infty} \frac{3(n + B) + k}{2} \cdot P^*(k) \\ &= \frac{3}{2}(n + B) + \frac{1}{2} \sum_{k=0}^{\infty} \frac{(n + B)kP(k) + k^2P(k)}{\mu} \\ &= \frac{3}{2}(n + B) + \frac{1}{2\mu} [(n + B)E\{k\} + E\{k^2\}], \end{aligned} \quad (17)$$

where  $E\{k\}$  and  $E\{k^2\}$  are given by (3) and (4), respectively. For large  $n$ , the sync recovery delay lower bound of (17) is approximated by  $2^n$ .

Equation (17) represents a lower bound because it is possible that the position of the slip can result in scenarios which prevent resynchronization at the next sync pattern. For example, a slip could occur in such a way that the new sequence of ciphertext bits results in a false synchronization. It is possible for this to occur near the end of the OFB block so that the receiver will interpret the next valid sync pattern bits as part of the false IV and will ignore them. As a result, resynchronization will be delayed until the next sync pattern. For small OFB block sizes, this could even happen in a manner such that several proper sync patterns are misinterpreted as part of the initialization vectors of several false synchronizations. This phenomenon is particularly prevalent for small values of  $n$  since small OFB block sizes are much more likely.

### 5.2 Upper Bound on SRD

Now, consider an upper bound on SRD. In our analysis, we consider two regions in which a slip may occur within a sync cycle of size  $n + B + k$ , as illustrated in Fig. 5. If a slip occurs such that the first bit following the slip is *not* within  $n + B$  bits of the sync pattern for the next cycle, then synchronization is lost until the valid sync pattern is detected for the next cycle. The probability of a slip occurring in this region is  $k/(n + B + k)$ . When a slip occurs within the last  $n + B$  bits of a 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 occurring in this region is given by  $(n + B)/(n + B + k)$  and, for simplicity, in deriving the upper bound for SRD, we shall assume that any such occurrence of a slip causes a false synchronization. Once a false synchronization has occurred, we assume that the next valid synchronization will be missed due to a portion of the valid sync pattern lying within the false IV. Subsequently, false synchronizations may occur at the receiver if the receiver misinterpretes a sync pattern appearing within an

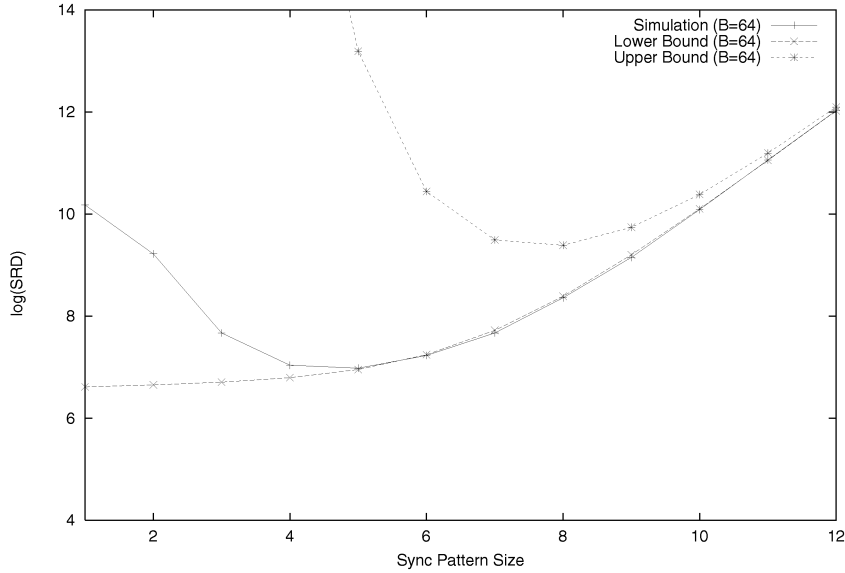


Fig. 7. Sync recovery delay vs. sync pattern size ( $B = 64$  bits).

actual IV to be a proper sync pattern. However, we can say that synchronization must be regained when the OFB block size  $k$  exceeds  $n + B$ , as the end of a false IV must then fall within an OFB block and the next valid sync pattern will be properly detected and synchronization regained. Synchronization will also be regained when the sequence of bits in the actual IV does not contain the sync pattern causing a false synchronization. However, since we are considering an upper bound, to make the problem tractable we shall ignore this possibility and consider the likelihood that synchronization is regained based on the probability that a sync cycle has  $k \geq n + B$ .

Based on these concepts, we derive an upper bound on SRD as given by

$$SRD < \sum_{k=0}^{\infty} \left[ \left( 2(n+B) + \frac{k}{2} \right) \frac{k}{n+B+k} + (2(n+B) + \mu\lambda) \frac{n+B}{n+B+k} \right] P^*(k), \quad (18)$$

where  $\lambda$  represents the expected number of sync cycles before  $k \geq n + B$  and is given by

$$\lambda = \frac{1}{\sum_{k=n+B}^{\infty} P(k)} = \frac{1}{\left(1 - \frac{1}{2^n}\right)^{n+B}}. \quad (19)$$

Making use of (16), the sum of (18) can be manipulated and the upper bound given as

$$SRD < \frac{1}{\mu} \left[ 2(n+B)E\{k\} + \frac{1}{2}E\{k^2\} + 2(n+B)^2 + \mu\lambda(n+B) \right]. \quad (20)$$

For small values of  $n$ , many sync cycles occur before  $k \geq n + B$  and, in reality, resynchronization will be achieved much more rapidly since it is possible that the

end of a false IV lies close to the end of an actual IV and it is likely, in this case, that no sync pattern is detected before the OFB block begins. Hence, the upper bound is very loose for small values of  $n$ , but is much tighter for larger values of  $n$ . It can be shown that, as  $n$  gets large, the upper bound on SRD approaches  $2^n$ .

### 5.3 Experimental Results for SRD

We have examined experimentally the sync recovery delay versus different values of  $n$  by running simulations with a sync pattern of the form  $10\dots00$ . It is reasonable to assume that slips are infrequent so that it is very unlikely to have more than one slip in a synchronization cycle, and, in our simulations, we have used a slip rate of 1 bit slip every  $10^5$  bits. For convenience, we have simulated individual bit slips only. However, the results for multiple bit slips would be no different. The results of the simulations for  $B = 64$  (using DES) and  $B = 128$  (using AES) are shown in Fig. 7 and Fig. 8, respectively. The lower and upper bounds of (17) and (20) are also illustrated in the figures. For convenience, the graphs present a plot of the logarithm base-2 of the sync recovery delay.

We can conclude that the synchronization recovery delay increases in an exponential manner. As  $n$  gets large, the synchronization recovery delay simulation results and the lower and upper bounds converge. It should be noted that the bounds are based on the geometric distribution for  $k$ , whereas the simulations are based on a sync pattern for which  $k$  follows closely, but not exactly, the geometric distribution. (See the Appendix for discussion of this point.) Also, as expected, the upper bound on SRD is very loose for small values of  $n$ , but appears to be quite tight for  $n \geq 10$ .

For values of  $n \leq 4$ , the effects mentioned previously cause resynchronization problems and we find from the simulation results that the sync recovery delay is actually minimized, in both cases, for a value of  $n = 5$  with SRD values of 126 and 225 for  $B = 64$  and  $B = 128$ , respectively. This may be compared to the synchronization recovery

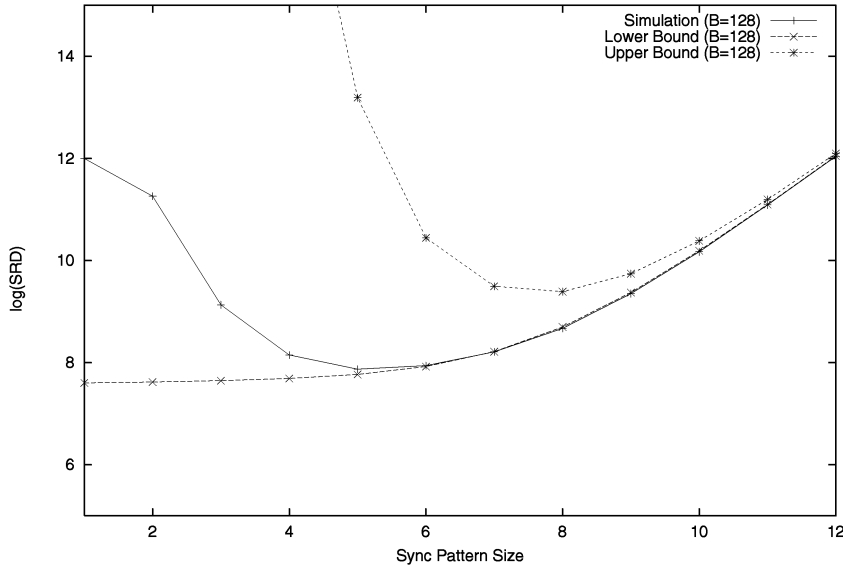


Fig. 8. Sync recovery delay vs. sync pattern size ( $B = 128$  bits).

delay of  $B$  for conventional CFB mode with  $m = 1$ . For large values of  $n$ , the synchronization recovery delay is much greater than for conventional CFB. For example, for  $n = 10$ , the sync recovery delay for SCFB is about  $2^{10}$  bits, irrespective of the block size.

## 6 ERROR PROPAGATION

In this section, we consider the effect of SCFB mode on the error characteristics at the output of the decryption. As we shall observe, the error propagation is on the same order as for conventional CFB mode and, hence, is much poorer than OFB mode.

We shall consider the *error propagation factor* (EPF) to be the bit error rate at the output of the decryption divided by the probability of a bit error in the communication channel (i.e., in the ciphertext). It is assumed that bit errors occur randomly and independently in the communication channel. In our work, we outline a lower bound, an upper bound, and simulation results for EPF for varying values of  $n$ .

### 6.1 Lower Bound on EPF

We consider first a simple lower bound on the error propagation factor. Let  $P_e$  represent the probability of a bit error in the ciphertext. The probability of an error occurring in either the sync pattern or the IV of a synchronization cycle is given by

$$\begin{aligned} P(\text{sync/IV error}) &= 1 - (1 - P_e)^{n+B} \\ &\approx (n + B) \cdot P_e, \end{aligned} \quad (21)$$

where the approximation is valid if  $n + B \ll 1/P_e$ , which would typically be true for reasonable values of  $n$ ,  $B$ , and  $P_e$ . When an error occurs in the sync/IV portion of a synchronization cycle, it is expected that half of the bits corresponding to the OFB block and the bits corresponding to the next sync/IV bits will be in error. This is irrespective of the size  $k$  of the OFB block portion of the synchronization cycle and, so, we may conclude that the expected bit error

rate at the output of the decryption process at the receiver will be lower bounded as

$$P(\text{error after decrypt}) > \frac{(n + B) \cdot P_e}{2}. \quad (22)$$

Hence, the lower bound on the error propagation factor is given by

$$EPF > \frac{n + B}{2}. \quad (23)$$

We might expect that other scenarios for bit errors (other than in the sync/IV bits) will only result in one bit error, in which case, this simple lower bound will suffice as a good estimate of EPF. However, this is not necessarily the case. Consider the list of error scenarios and the resulting effects as shown in Table 1. Note that Cases 3 and 4 involve false detection of the sync pattern. This can occur if a bit error results in a sync pattern in the received ciphertext. The probability that a bit error causes a false synchronization is less than  $n/(2^n - 1)$ . For small values of  $n$ , Cases 3 and 4 become much more likely and the lower bound for the error propagation factor of  $(n + B)/2$  is very loose. But, as  $n$  increases, Cases 1 and 2 become the most significant since the probability of a bit error occurring such that false sync patterns are generated decreases dramatically as  $n$  increases. Clearly, the most likely scenario becomes Case 2 since most of the ciphertext bits will correspond to the OFB block. However, it is actually Case 1 which is the cause of the majority of bit errors after decryption. Although the occurrence of a bit error corresponding to Case 1 becomes significantly less likely than Case 2 for larger  $n$  due to the larger OFB block sizes, this is offset by the fact that the number of bit errors produced at the output of the decryption due to Case 1 increases in proportion to the decrease, i.e., about 1/2 of the bits in the larger OFB block are in error. Hence, Case 1 becomes the significant factor in determining the error propagation factor for larger  $n$ .

TABLE 1  
Effects of Different Error Scenarios

Case	Error Scenarios	Effect
1	error in $n + B$ bits of sync/IV block	sync lost for entire cycle ( $\sim \frac{k+n+B}{2}$ expected bit errors)
2	error in OFB block such that no sync pattern is falsely generated	one bit error generated in recovered plaintext
3	error in OFB block such that false sync generated in first $k - (n + B)$ bits of OFB block	$i/2$ bit errors generated, where $i$ is number of bits between end of false IV and end of next IV
4	error in OFB block such that false sync generated in last $n + B$ bits of OFB block	next sync pattern can be missed because it is interpreted to be part of false IV causing $1/2$ bits in error until next valid sync
5	error while sync already lost at receiver	generates a possible bit error at the corresponding position of recovered plaintext

## 6.2 Upper Bound on EPF

Consider now, an upper bound on EPF based on approximating bounds on the likelihood and effects of the cases in Table 1. Consider a sync cycle of size  $n + B + k$ . The probability that a bit error belongs to Case 1 is

$$P_1(k) = \frac{n + B}{n + B + k} \quad (24)$$

and the resulting expected number of bit errors is given by

$$\delta_1(k) = 1 + \frac{n + B + k}{2}. \quad (25)$$

For Case 2, the expected number of bit errors is  $\delta_2(k) = 1$  and the probability is typically quite high and, hence, 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  $n/(2^n - 1)$ . This is useful in determining the upper bound on EPF since the more likely it is that a false synchronization occurs, the larger EPF. Hence, Case 3 occurs with a probability upper bounded by

$$P_3(k) < \frac{n}{2^n - 1} \cdot \frac{k}{n + B + k}. \quad (26)$$

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

$$\delta_3(k) < n + B + \frac{k}{4}, \quad (27)$$

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

The upper bound on the probability that Case 4 occurs is given by

$$P_4(k) < \frac{n}{2^n - 1} \cdot \frac{n + B}{n + B + k}. \quad (28)$$

For this scenario, we assume that at least the next sync pattern is missed and it could take several sync cycles to recover synchronization (similarly to the loss of sync due to a slip in the last  $n + B$  bits of an OFB block). Hence, the expected number of errors caused by a Case 4 bit error can be large and is upper bounded by

$$\delta_4(k) < \frac{n + B}{2} + \frac{\mu \cdot \lambda}{2}, \quad (29)$$

where  $\mu$  and  $\lambda$  represent the expected number of bits in a sync cycle and the expected number of cycles until resync is achieved and are given in (5) and (19), respectively. The  $(n + B)/2$  term represents an upper bound on the expected number of bit errors between the end of the false IV and the end of the IV of the next sync cycle.

In our analysis, we assume that errors and slips are infrequent enough that an error occurs in isolation. Hence, the effect of an error is not influenced by other errors or slips and we, therefore, ignore Case 5. As a result, an estimate of the upper bound on the error propagation factor is given by

$$\begin{aligned} EPF &\approx \sum_{i=1}^4 \sum_{k=0}^{\infty} P^*(k) \cdot P_i(k) \cdot \delta_i(k) \\ &< \frac{n + B}{2} + \frac{n + B}{\mu} + 1 + \frac{n}{\mu(2^n - 1)} \\ &\quad \left[ (n + B)E\{k\} + \frac{1}{4}E\{k^2\} + \frac{1}{2}(n + B)^2 + \frac{1}{2}\mu\lambda(n + B) \right]. \end{aligned} \quad (30)$$

From this expression, it may be shown that, as  $n$  gets large, the upper bound approaches  $n + B/2 + 1$ .

## 6.3 Experimental Results for EPF

We have investigated the error propagation factor experimentally and the results are illustrated in Fig. 9 and Fig. 10, where the error propagation factor is given as a function of  $n$  for a value of  $P_e = 10^{-5}$  for  $B = 64$  and  $B = 128$ . The simulation results are generated by randomly generating bit errors in the channel such that the effects of all cases listed

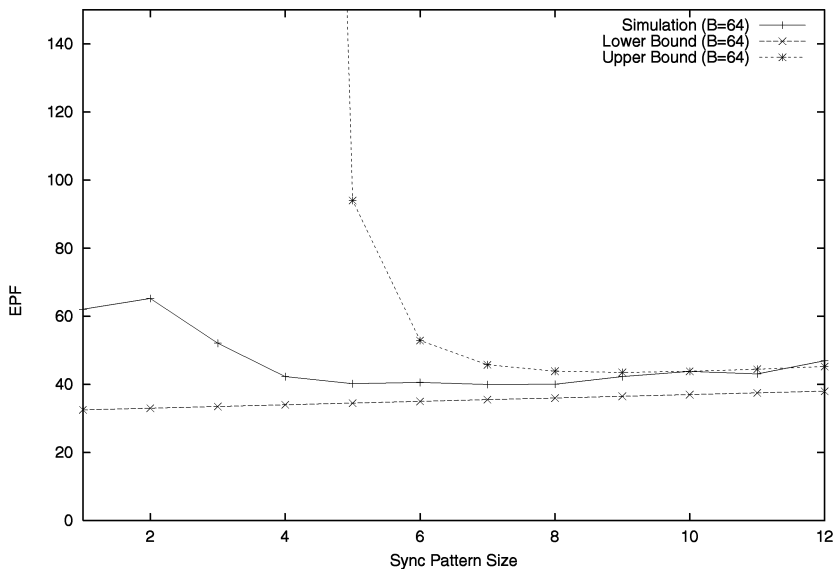


Fig. 9. Error propagation factor vs. sync pattern size ( $B = 64$  bits).

in Table 1 would be incorporated into the results. In order to get realistic statistical samples with the available computing resources, the maximum size of  $n = 12$  was selected for the simulation. The lower and upper bounds for the two cases are also illustrated. Once again, the block ciphers used in the simulations are DES and AES and the sync pattern used is of the form  $10\dots00$ . From the simulations it can be seen that the error propagation factor for most values of  $n$  is close to the case of conventional CFB mode with  $m = 1$ , where the error propagation factor is 33 for DES and 65 for AES.

It is perhaps surprising that the error propagation factor follows the conventional CFB mode very closely. As  $n$  increases, since most of the synchronization cycle becomes the OFB block and virtually all bit errors will occur within

the OFB block, one might expect that, on average, only about one bit error is produced at the decryption output for each bit error that occurs in the communication channel. This is true for most bit errors. However, as the probability of the occurrence of a bit error in the sync pattern and IV decreases, the resulting number of bit errors due to such an error increases proportionally so that, as indicated by the lower and upper bounds, the expected number of bit errors after decryption is much greater than one and the SCFB error propagation factor follows CFB more closely than OFB, even for large  $n$ .

Note that, in the analysis of the sync recovery delay and error propagation factor, we have focused on the effect of isolated events such as individual bit errors or slips. Of course, in practice, although it would be very infrequent,

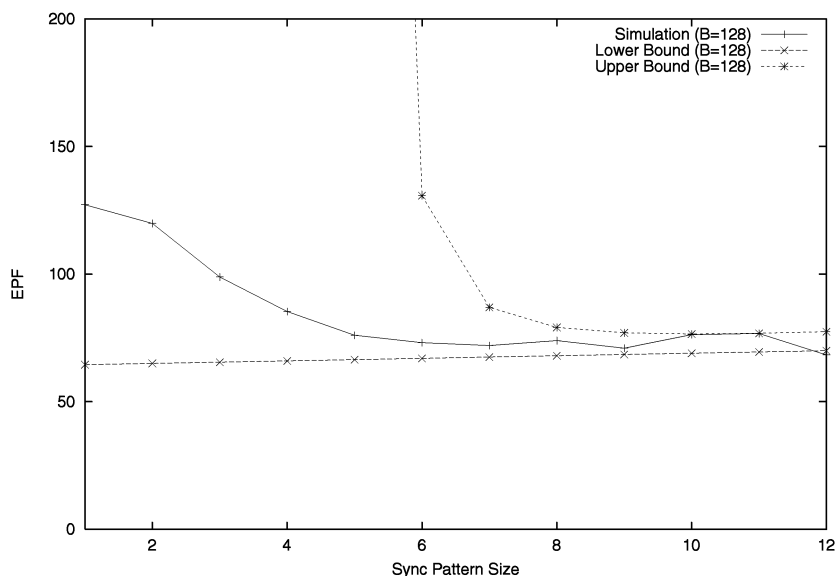


Fig. 10. Error propagation factor vs. sync pattern size ( $B = 128$  bits).

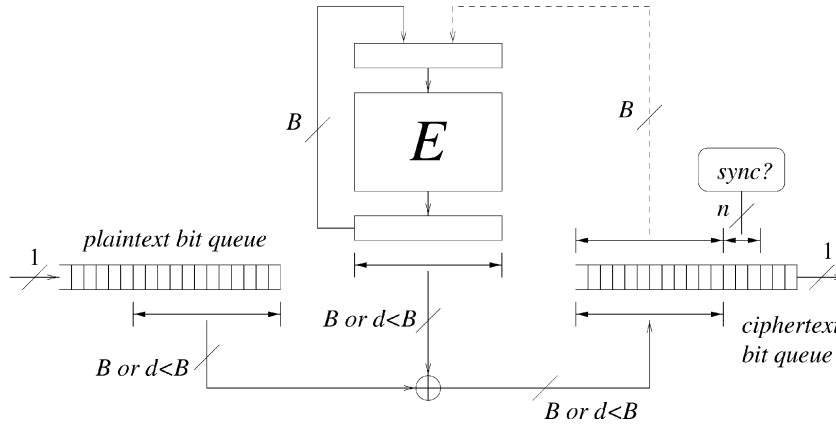


Fig. 11. SCFB encryption system.

combinations of slips and errors may occur. It would be a complex task to identify and analyze all such cases. However, it should be noted that, in such cases, generally the effect of the error or slip will be no more severe than the effect of an isolated error or slip. For example, a bit error that occurs while synchronization has been lost due to a slip is highly unlikely to influence the state of the cipher and is an event of no consequence in comparison to the sync loss. For these reasons, we are really most interested in the effect of isolated events.

## 7 IMPLEMENTATION ISSUES

In this section, we investigate the relationship of cipher speed (as measured by the efficiency of SCFB mode with respect to straight block encryption) and the delay caused by the SCFB mode of operation. The efficient operation of the SCFB mode is conditioned on the appropriate buffering of data to ensure that, should many resynchronizations occur closely in time, there is enough buffer available to avoid buffer overflow while storing plaintext data and there is enough ciphertext buffered to avoid underrun when production of ciphertext gets delayed due to the many required block encryptions in a short time span. We shall examine the buffering requirements of an implementation of SCFB mode. As we shall see, the buffering requirements are directly equivalent to the delay of data as it is transferred through the SCFB encryption system.

### 7.1 Implementation of an SCFB System

For the purposes of our work, we consider SCFB mode to be implemented for encryption as shown in Fig. 11. (A similar system can be envisioned for decryption.) In the system, there are two queues: one for incoming plaintext bits and one for outgoing ciphertext bits. Both queues are assumed to have buffer space of size  $M$  bits. When the SCFB system is operating in OFB mode (i.e., no sync pattern is detected), bits will collect in the plaintext queue until a block cipher output block is available and the number of bits in the queue is equal to the block size  $B$ . At this point, the  $B$  plaintext bits are XORed with the output of the block cipher and all  $B$  bits are placed into the ciphertext bit queue.

While bits are being collected in the plaintext queue, bits are being removed from the head of the ciphertext queue at exactly the same rate. The ciphertext queue is initialized with arbitrary data so that it is in the full state when the plaintext queue is empty. As the plaintext queue fills up, the ciphertext queue empties and, if the plaintext queue gets full, the ciphertext queue will be empty. Consequently, the condition of an overflow in the plaintext queue and an underrun in the ciphertext queue are exactly the same and we need only consider the plaintext queue overflow condition.

Let  $h$  represent the number of bits in the plaintext bit queue. Under conditions of no resynchronizations,  $h$  will increment from 0 to  $B$  bits and then drop back down to 0, repetitively. The number of bits in the ciphertext queue is given by  $M - h$ . Hence, the delay through the system experienced by any bit is given by  $h + (M - h) = M$  bit times.

In order to minimize delay, it is clearly desirable to keep the buffer size  $M$  as small as possible. However,  $M$  must be greater than  $B$  and must be large enough to deal with the build up of data that can occur in the plaintext queue when resynchronizations occur. When the sync pattern is detected in the bits being placed into the ciphertext queue, the system continues encrypting until all  $B$  bits of IV are collected from the ciphertext. The last bit of IV is unlikely to occur on the boundary of a block cipher output block and is, in fact, equally likely to occur anywhere within a block cipher output block. However, since all bits following the last bit of IV must be encrypted using the new output of the block cipher as a result of the block cipher processing of IV, only part of the block (i.e., up to the last IV bit) needs to be XORed and, although the XOR of this partial block can be executed as soon as the bit corresponding to the last IV bit is available in the plaintext queue, there is a delay in XORing the following bits due to the block cipher processing of IV. During this delay, incoming plaintext bits are buffered into the plaintext bit queue.

If many resynchronizations occur close together, the plaintext queue will fill up and may overflow, resulting in lost data bits. (This will manifest itself as underruns in the ciphertext queue.) In order to keep resynchronizations from continually growing the queue, the plaintext bits are removed from the queue at a rate greater than the rate of

bits entering the queue. The size of the queues must be large enough to ensure that the probability of an overflow is sufficiently small so as not to create a large number of bit errors. Hence, there is a relationship between the rate at which the block cipher is run (and the corresponding rate at which bits are removed from the plaintext queue) and the size required for the buffer.

There are other structures that may be envisioned to suitably implement an SCFB system. One straightforward approach to the implementation would be to place a buffer at the output of the block cipher and to remove the plaintext and ciphertext queues. The next keystream block would be generated while the current block is being used to encrypt the plaintext. This would clearly minimize delay through the system. However, although, during the OFB phase of a synchronization cycle, the block cipher need only be run at a rate equal to the incoming data, when a resynchronization occurs, a block cipher output must be generated within one bit time, implying that the block cipher must be ready to operate at  $B$  times the channel rate, equivalent to an efficiency of  $1/B$ , i.e., the same as conventional CFB. Hence, there is little value in this approach. Note that differing implementations of SCFB will not differ in the analysis of the sync recovery and error propagation characteristics, although they do vary in implementation efficiency and in delay due to queuing.

## 7.2 Practical Efficiency Concepts

In Section 4, the concept of theoretical efficiency is introduced as an indication of the rate at which the block cipher must operate in order to be able to process the incoming plaintext bits quickly enough to avoid queues growing without bounds. We now turn our attention to practical efficiency concepts related directly to the implementation of Fig. 11 and define *full-queue efficiency* to represent the efficiency at which the system operates while there is data in the plaintext queue to process. That is, if the queue has greater than  $B$  bits in it, bits are removed from the queue in blocks of  $B$  bits at a rate of  $(1/\alpha) \times R/B$  blocks per unit time, where  $\alpha$  is the cipher full-queue efficiency and  $R$  is the rate at which bits enter the plaintext queue in bits per unit time. Since data is arriving at the plaintext queue at a rate of  $R/B$  for each block of  $B$  bits, the full-queue efficiency represents the incoming plaintext data rate divided by the rate at which bits are removed from the plaintext queue. For example, for a plaintext queue with some data in it, an efficiency of  $\alpha = 50\%$  implies that, for every  $B$  bits entering the queue,  $2B$  bits are removed from the queue, XORed with the output of the block cipher, and placed in the ciphertext queue.

In practice, there will be periods when the plaintext queue has fewer than  $B$  bits available at the completion of a block cipher operation and the system must wait until the queue has  $B$  bits. Hence, the *average* efficiency is greater than the *full-queue* efficiency since, during the full-queue situation, the block cipher is operating at its peak rate and the corresponding efficiency is at a minimum. Assuming that we have a stable system (i.e., the plaintext queue does not grow without bound), the average and full-queue efficiencies must both be less than the theoretical efficiency given in Section 4.

Assuming the time to execute a block cipher operation is fixed, a higher efficiency implies that the system is capable of operating at higher rates, i.e., larger values of  $R$ . The full-queue efficiency,  $\alpha$ , gives a direct indication of the operating rate of the system since  $R = \alpha \cdot r_{enc} \cdot B$ , where  $r_{enc}$  represents the encryption rate of the block cipher while there is data to process in the plaintext queue in blocks per unit time. The parameter  $r_{enc}$  is generally constrained by the technology used to implement the block cipher.

## 7.3 Buffering and Delay Characteristics

In our work, we investigated the buffer requirements  $M$  (or, equivalently, the delay measured in bit times) of the SCFB encryption system in Fig. 11 by executing a number of simulations for systems with different buffer sizes and full-queue efficiencies,  $\alpha$ . As the output of the simulations, we examined the probability of an overflow of the plaintext queue (or, equivalently, an underrun of the ciphertext queue). We observed that, although approaching speeds very close to the theoretical efficiency,  $\eta$ , defined in Section 4, requires much buffer space and a resulting large delay, using speeds reasonably close (e.g., with full-queue efficiencies of about 80-90 percent of theoretical efficiency) requires modest buffering and causes small delays. As an example, consider Fig. 12 and Fig. 13, both of which are simulation results based on 64-bit DES with an 8-bit synchronization pattern of "10000000".

Fig. 12 illustrates the effect on the probability of buffer overflow as a function of the buffer size for fixed full-queue efficiencies. Although, in theory, the efficiency can be as high as 91.1 percent, it is clear that running the system at close to that efficiency causes significant buffer overflows for buffer sizes up to 256 bits. However, for full-queue efficiencies of 78.1 percent and 84.4 percent, buffer overflow becomes small for buffers of about 120 and 160 bits, respectively.

Fig. 13 shows that the probability of overflow increases dramatically as the full-queue efficiency approaches the theoretical efficiency of 91.1 percent. However, for large buffer sizes of 160 and 192, it can be seen that full-queue efficiencies of 80 percent to 85 percent can be achieved before significant overflow occurs. Even for small buffer sizes, such as 96, full-queue efficiencies can be 70 percent with little buffer overflow.

Results of a similar nature have been observed for the 128-bit AES cipher. Hence, we conclude that it is feasible to run the system at full-queue efficiencies approaching theoretical efficiencies and still maintain modest buffer sizes and resulting delays equivalent to a few blocks.

Of particular note is the efficiency value of 50 percent. For any sync pattern size, running the system at rates corresponding to 50 percent (or less) efficiency guarantees that there is no buffer overflow for a buffer size of  $B$ . This occurs because the block cipher is being run at a rate that is two times (or more) the rate of incoming bits and, since all synchronization cycles consist of the IV collection phase followed by OFB encryption based on the IV, the worst case scenario requires no more than two block cipher encryptions for any  $B + l, n \leq l \leq B$ , bits. This is a significant point because it implies that running SCFB at 50 percent full-queue efficiency guarantees no overflow and a delay of

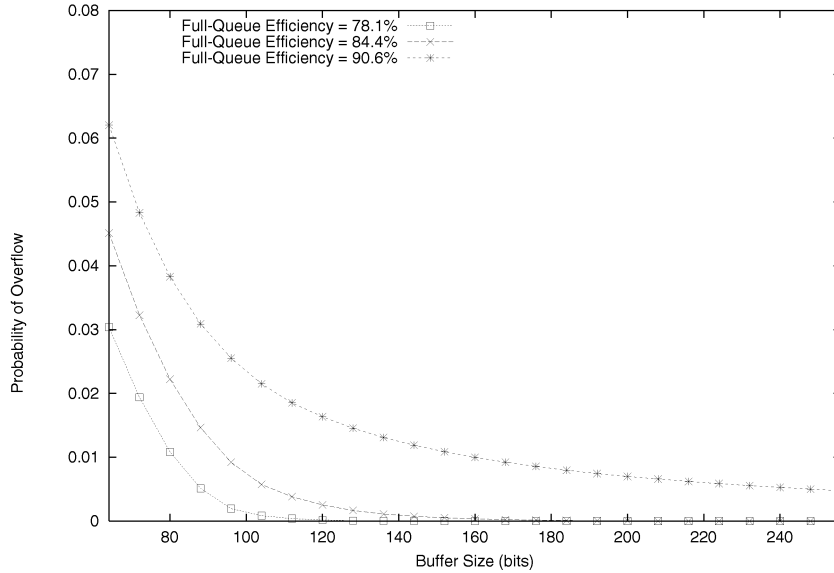


Fig. 12. Probability of overflow vs. buffer size.

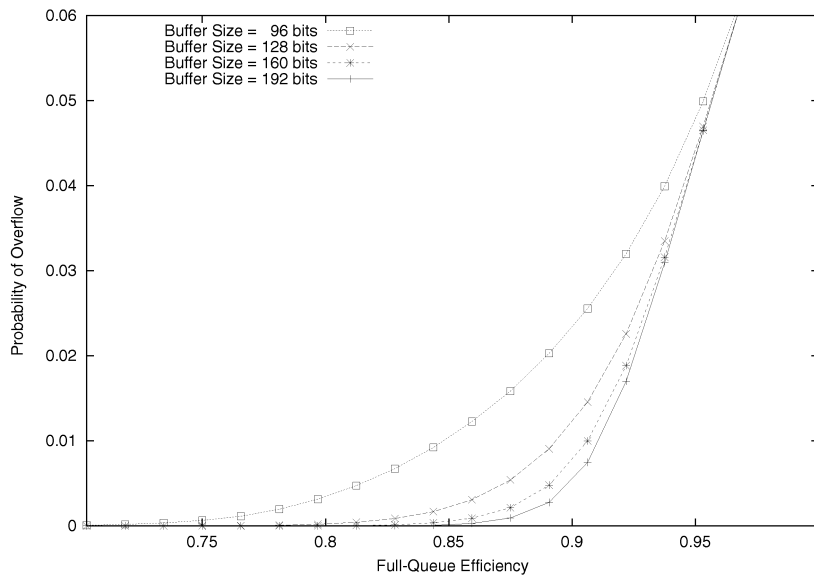


Fig. 13. Probability of overflow vs. encryption efficiency.

exactly  $B$  bit times. This may be compared to 1) conventional CFB, which, while being capable of providing minimal delays, results in an efficiency of only  $1/B$ , and 2) OFB which can provide minimal delay but has no self-synchronization capabilities.

### 8 SECURITY ISSUES

In this section, we explore some of the security issues associated with the SCFB mode of operation. In particular, we consider the susceptibility of the mode to passive attacks which look for repetitions in the keystream.

Since IV is essentially a random block of  $B$  bits, one concern for the security of SCFB mode is that an IV value might be the same as a block cipher input from another synchronization cycle. This could result in a large sequence

of bits that is encrypted with the same keystream sequence as another sequence of plaintext bits. When a significant portion of the keystream repeats, knowledge of some plaintext might compromise the security of other encrypted unknown plaintext bits. It should be noted that this situation also potentially exists for conventional OFB mode: When resynchronizations are executed by sending a new IV through a signaling channel, it is generally good practice to avoid reusing IVs.

This security concern is eliminated through the use of large block sizes. We shall investigate the issue by considering the probability of getting an IV that has already been used at the input to the block cipher in a previous sync cycle or getting a block cipher input that has been used as an IV for a previous sync cycle. It is possible to derive an upper bound on the probability and therefore be able to make a statement regarding the security of SCFB with

respect to a repeating keystream. The upper bound is derived by considering the likelihood that each input to the block cipher is different than all previous inputs, regardless of whether the input is coming from the previous output of the block cipher or the IV taken from the ciphertext. The assumption that each input block is randomly selected from all possible blocks is used. Hence, the probability of getting at least two identical input blocks in  $t$  blocks is given by [8]:

$$\begin{aligned} P(\text{repeated keystream}) &< 1 - \prod_{i=0}^{t-1} \left(1 - \frac{i}{2^B}\right) \\ &\approx 1 - \prod_{i=0}^{t-1} e^{-i/2^B} \\ &= 1 - e^{-t(t-1)/2^{B+1}} \\ &\approx 1 - e^{-t^2/2^{B+1}}, \end{aligned} \quad (31)$$

where we have used the reasonable assumptions that  $t \ll 2^B$  and  $1 - x \approx e^{-x}$  for  $x \ll 1$  in the first approximation and  $t \gg 1$  in the second approximation. Now, if we assume that  $t^2 \ll 2^{B+1}$  and again using  $1 - x \approx e^{-x}$ , we expect that

$$P(\text{repeated keystream}) < \frac{t^2}{2^{B+1}}. \quad (32)$$

For SCFB mode, a block cipher operation, on average, results in the encryption of  $\eta B$  plaintext bits where  $\eta$  is the theoretical efficiency given by (15). Hence, for a block size of  $B = 128$  and  $n = 8$ , being able to store about 1 terabit is the equivalent amount of data corresponding to about  $2^{33}$  block cipher operations, resulting in two identical input blocks with a probability of less than  $10^{-19}$ . For  $B = 64$ , the approximation of (32) does not hold, but, by considering the ‘‘birthday paradox’’ [8], it can be rationalized that, for  $n = 8$ , about 1/2 terabit is required to be stored before any two block cipher input values used in SCFB encryption can be expected to be the same with high probability (i.e., on the order of 50 percent). It is reasonable to conclude, therefore, that repetition of the keystream has negligible probability in SCFB mode with  $B = 128$ . However, for  $B = 64$ , there is a risk that a keystream repetition may be exploited if a large amount of data can be stored. This risk can be mitigated by keeping the amount of data encrypted under one block cipher key small enough.

In this section, we have considered the security of SCFB mode with respect to a passive attack based on repetition of the keystream. Although there is intuitive justification for believing that the SCFB mode would possess similar security to conventional cipher feedback and output feedback modes, in this paper, we do not pursue formal proofs of concrete security for SCFB such as those presented in [9] for cipher block chaining and counter modes. We leave, as an open problem, the formal proof of the security of SCFB mode with respect to more sophisticated cryptanalysis methods such as adaptive chosen plaintext attacks.

## 9 CONCLUSIONS

In the paper, we have examined the efficiency, synchronization recovery delay, and the error propagation characteristics of a recently proposed encryption mode that we refer to as the statistical cipher feedback (SCFB) mode. Most significantly,

we have examined these characteristics as a function of the synchronization pattern size. We conclude that SCFB mode provides self-synchronization with a recovery time from sync losses that is dependent exponentially on the sync pattern size. The major advantage of SCFB is that, as the sync pattern size increases, the efficiency of SCFB mode approaches 100 percent and, even for modest  $n$  (e.g.,  $n = 8$ ), the efficiency is large. The error propagation characteristics are demonstrated to be similar to CFB mode and not heavily dependent on the sync pattern size. Hence, for a particular application, the actual sync pattern size can be selected by determining a suitable balance between synchronization recovery and efficiency.

We have also examined the practical implementation aspects of SCFB mode and can conclude that SCFB should be used in circumstances where slips are a concern (thereby necessitating self-synchronization) and where implementation efficiency is a high priority in comparison to latency. In comparison to OFB mode, SCFB can provide similar implementation efficiencies and, while the error propagation characteristics of SCFB are poor compared to OFB, SCFB provides self-synchronization capabilities, which OFB does not. In comparison to conventional CFB mode, SCFB provides far better implementation efficiencies, with similar error propagation characteristics and poorer synchronization recovery delays. Further, efficient implementation of SCFB does necessitate some encryption latency due to the required buffering to deal with resynchronization occurrences, whereas both OFB and CFB can be implemented with minimal latency.

## APPENDIX A

### DISCUSSION ON SYNCHRONIZATION CYCLE LENGTH

In the paper, we have used the geometric distribution to characterize the probability distribution of the OFB block size  $k$ . However, the sync pattern is scanned for by essentially sliding along a window of  $n$  bits and comparing this to the sync pattern. Hence, there will be an overlap of windows and the detection of a sync pattern at a particular position is not independent of the sync pattern occurring at other locations.

In fact, assuming that each bit of ciphertext is equally likely to be a 0 or 1 and that each bit is independent, it is found that the distribution of the random variable  $k$  is actually dependent on the sync pattern. We illustrate this by examining, for varying sync pattern sizes  $n$ , two cases: sync patterns of the form  $10 \dots 00$  and of the form  $11 \dots 11$ .

#### A.1 Sync Pattern $10 \dots 00$

The probability distribution of  $k$  can be shown to be

$$P_a(k) = \begin{cases} [1 - \sum_{i=0}^{k-n} P_a(i)] \cdot \frac{1}{2^n}, & k > 0 \\ \frac{1}{2^n}, & k = 0 \\ 0, & k < 0. \end{cases} \quad (33)$$

Equation (33) is explained as follows: First, it should be noted that  $P_a(k)$  is equivalent in meaning to the probability that the next sync pattern ends  $k + n$  bits after the last bit of the previous IV. It is trivially true that  $k$  must be nonnegative. The case for  $k = 0$  occurs when the first  $n$  bits

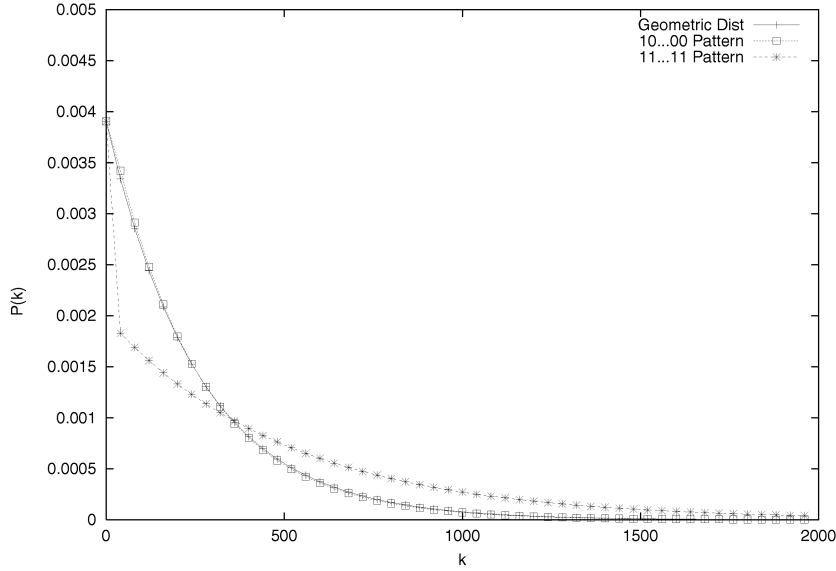


Fig. 14. Probability distribution comparison,  $n = 8$ .

following the last bit of IV correspond to the  $n$  bits of the sync pattern. The probability for the case of  $k > 0$  comes from the fact that, in order to finish the sync pattern at the  $(k + n)$ -th, the  $(k + 1)$ -th to  $(k + n)$ -th bits must each have values corresponding to the bits of the sync pattern: The probability of this is  $1/2^n$ . As well, there must be no sequence of bits equivalent to the sync pattern in the first  $k$  bits: The probability of this is given by the complement of the probability that the next sync pattern finishes at bit  $n$  or bit  $n + 1$ , etc., which is simply the sum of the probabilities up to  $P_a(k - n)$ . The resulting probability for the case of  $k > 0$  is given by the product of the probability that the pattern does not appear in the first  $k$  bits and the probability that it does appear in the bits  $k + 1$  to  $k + n$ . Particularly, it should be noted that the pattern cannot appear in bits  $k +$

$1 - i$  to  $k + n - i$ ,  $1 \leq i \leq n - 1$ , and also in bits  $k + 1$  to  $k + n$  because of the format of the bit pattern. This is accounted for in the calculation of the probability. (This is quite different than in the calculation of the geometric distribution, where it is assumed that if the pattern finishes at  $k + n$ , the probability that it does not finish at  $k + 1 - i$  to  $k + n - i$  must be accounted for.)

We have verified computationally that the expected value of  $k$  for the sync pattern  $10 \dots 00$  based on probability distribution  $P_a$  is given by

$$E_a\{k\} = 2^n - n, \tag{34}$$

which differs from the value for the geometric distribution given by (3). However, this difference is small.

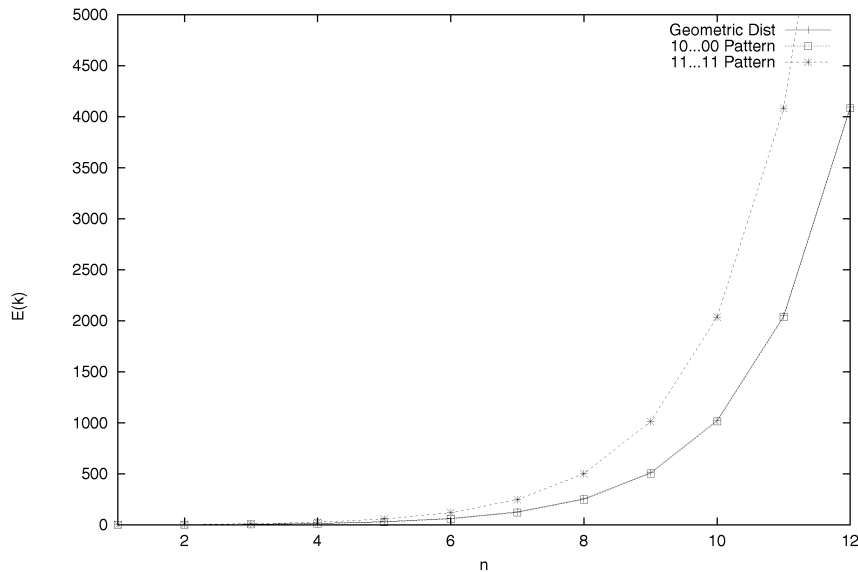


Fig. 15. Expected value comparison.

## A.2 Sync Pattern 11...11

In this case, the probability distribution can be shown to be

$$P_b(k) = \begin{cases} [1 - \sum_{i=0}^{k-n-1} P_b(i)] \cdot \frac{1}{2^{n+1}}, & k > 0 \\ \frac{1}{2^n}, & k = 0 \\ 0, & k < 0. \end{cases} \quad (35)$$

The case for  $k > 0$  varies from  $P_a(k)$  of (33). It is derived from the fact that, in order to finish the sync pattern at the  $(k+n)$ -th bit, the  $k$ th bit must have a value of 0 and the  $(k+1)$ -th to the  $(k+n)$ -th must each have values of 1: The probability of this is  $1/2^{n+1}$ . As well, there must be no sequence of bits equivalent to the sync pattern in the first  $(k-1)$ -th bits: The probability of this is given by the complement of the probability that the next sync pattern finishes at bit  $n$  or bit  $n+1$ , etc.. The product of these probabilities results in the case for  $k > 0$  in (35).

For sync pattern 11...11, the expected value of  $k$  is given by

$$E_b\{k\} = 2 \cdot (2^n - 1) - n, \quad (36)$$

which differs significantly from the expected value in the geometric distribution.

## A.3 Comparison of Distributions

For a comparison of the probability distributions between the two cases of sync patterns and the geometric distribution, see Fig. 14 for which  $n = 8$ . As well, Fig. 15 illustrates the expected values for the different distributions as a function of  $n$ .

It is clear that the sync pattern 10...00 follows the distribution and the resulting expected values of the geometric distribution very closely. (In fact, the distribution for  $n = 8$  and the expected value plots follow virtually exactly the geometric distribution from the perspective of a visual interpretation of Fig. 14 and Fig. 15.) We have found this to be the case for most sync patterns. However, the sync pattern 11...11 has a distribution significantly different than the geometric distribution and has an expected value for  $k$  that is about twice the expected value of the geometric distribution. In our experiments, we have used the sync pattern 10...00 and, so, we expect that the theoretical values developed from the geometric distribution should be a reasonable approximation of the values for the actual sync pattern.

## ACKNOWLEDGMENTS

This work was supported by the Natural Sciences and Engineering Research Council of Canada. The material in this paper was presented in part at IEEE Infocom 2001, Anchorage, Alaska, April 2001, and at the 2001 IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing (PACRIM '01), Victoria, B.C., August 2001.

## REFERENCES

- [1] O. Jung and C. Ruland, "Encryption with Statistical Self-Synchronization in Synchronous Broadband Networks," *Proc. Cryptographic Hardware and Embedded Systems (CHES '99)*, pp. 340-352, 1999.
- [2] A.J. Menezes, P. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997.

- [3] Nat'l Inst. of Standards and Technology, "Data Encryption Standard (DES)," Federal Information Processing Standard 46, 1977.
- [4] Nat'l Inst. Standards and Technology, "Advanced Encryption Standard (AES)," Federal Information Processing Standard 197, 2001.
- [5] Int'l Organization for Standardization (ISO), "Information Technology—Security Techniques—Modes of Operation for an  $n$ -bit Block Cipher," ISO/IEC 10116, 1997.
- [6] U.M. Maurer, "New Approaches to the Design of Self-Synchronizing Stream Ciphers," *Advances in Cryptology—Proc. EURO-CRYPT '91*, pp. 458-471, 1991.
- [7] A. Alkassar, A. Gerald, B. Pfitzmann, and A.-R. Sadeghi, "Optimized Self-Synchronizing Mode of Operation," *Fast Software Encryption Workshop (FSE 2001)*, Apr. 2001.
- [8] D.R. Stinson, *Cryptography: Theory and Practice*, second ed. CRC Press, 2002.
- [9] M. Bellare, A. Desai, E. Jorjipii, and P. Rogaway, "A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation," *Proc. 38th Ann. Symp. Foundations of Computer Science*, pp. 394-403, 1997.



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

► For more information on this or any computing topic, please visit our Digital Library at <http://computer.org/publications/dlib>.