# The Role of Decimated Sequences in Scaling Encryption Speeds Through Parallelism

Edward L. Witzke
RE/SPEC Inc.
4775 Indian School Road N.E., Suite 300
Albuquerque, New Mexico 87110
elwitzk@respec.com
(505)268-2661

Sandia National Laboratories
Lyndon G. Pierson
Mail Stop 0806
P.O. Box 5800
Albuquerque, New Mexico 87185-0806
lgpiers@sandia.gov
(505)845-8212

## Abstract

Encryption performance, in terms of bits per second encrypted, has not scaled well, as network performance has increased. The authors felt that multiple encryption modules, operating in parallel would be the cornerstone of scalable encryption. One of the major problems with parallelizing encryption is ensuring that each encryption module is getting the proper portion of the key sequence at the correct point in the encryption or decryption of the message. Many encryption schemes use linear recurring sequences, which may be generated by a linear feedback shift register. Instead of using a linear feedback shift register, the authors describe a method to generate the linear recurring sequence by using parallel decimated sequences, one per encryption module. Computing decimated sequences can be time consuming, so the authors have also described a way to compute these sequences with logic gates rather than arithmetic operations.

## Introduction

End-to-end encryption can protect proprietary information as it passes from one end of a complex computer network to another, through untrusted intermediate systems. Encryption performance, in terms of bits per second encrypted, has not scaled well, as network performance has increased. Encryption performance in terms of long term secrecy also suffers as computer performance is scaled, cracking previously secure algorithms.

The overall problem addressed in the authors' research is: How can end-to-end encryption technology be scaled for high performance in the Gigabit per second networking arena? The authors, along with other members of the project team, identified and analyzed current research efforts in scalability of encryption and interoperability of scaled and unscaled encryptors.

## Approach

The authors found that multiple encryption modules operating in parallel would provide an alternative for creating a scalable encryption architecture able to keep pace with the needs of modern, high performance communication networks. Varying numbers of encryption modules could be combined with control circuitry to produce an encryption unit for a specific computer system or class of systems with similar capabilities. The number of encryption modules making up a parallel encryption unit would be determined by the speed at which the computer system is expected to communicate to networks. A supercomputer may require 8 or 16 encryption modules operating in parallel to keep up with its network communications demand, while a workstation may only need 2 or 4 encryption modules in parallel to accommodate its workload. A PC may only require 1 encryption module.

These encryption units with different degrees of parallelism must interoperate with each other across networks, while remaining synchronized. Encryption functions with feedback based on a combination of key and plaintext or ciphertext do not scale and interoperate. There are ways to design an encryption unit of parallel modules incorporating this feedback, which would provide a scalable solution. The drawback is that the units would not interoperate with other units having a different degree of parallelism, i.e. an 8-way encryptor would not interoperate with a 2-way encryptor. This eliminates the feedback modes of the Data Encryption Standard (DES) as candidates for the individual encryption modules, the building blocks of the parallel encryption unit.

A linear recurring sequence can be used to feed a nonlinear encryption function. Because of the ease of implementing linear feedback shift registers (LFSRs) in hardware, one may use the concept of parallel LFSRs to feed parallel nonlinear encryption modules. Now the problem becomes a question of how to ensure that each encryption module is getting the proper portion of the linear recurring sequence at the correct point in the encryption or decryption of the message.

A $n$ th order homogeneous linear recurring sequence in a field $F_2$ satisfies the linear recurrence relation $S_n = A_{n-1}S_{n-1} + A_{n-2}S_{n-2} + \ldots + A_0S_0$ and would correspond to a left shifting (Fibonacci) linear feedback shift register. $S_0,\ldots, S_{n-1}$ correspond to the cells in the LFSR, while the nonzero coefficients $(A_0,\ldots, A_{n-1})$ correspond to the taps of the LFSR. Lidl and Niederreiter[4] provide an excellent treatise of linear recurring sequences, feedback shift registers, and characteristic polynomials.

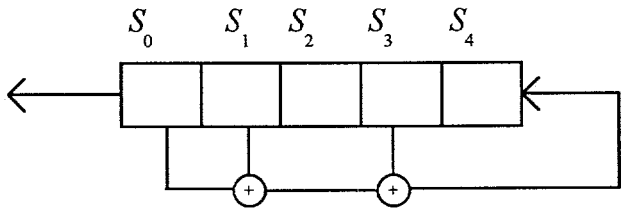
To decimate a sequence is to replace an integer sequence $\bar{a} = \{ a_1, a_2, a_3, a_4, \ldots, a_{2^n-1} \}$ with the sequence $\bar{a}_k = \{ a_k, a_{2k(\mathrm{mod}\,2^n)}, a_{3k(\mathrm{mod}\,2^n)}, a_{4k(\mathrm{mod}\,2^n)}, \ldots, a_{k(2^n-1)(\mathrm{mod}\,2^n)} \}$, thereby producing a sequence of every $k^{\mathrm{th}}$ element of the main sequence. Using decimated sequences in parallel, one can reduce the amount of time it takes to generate the main sequence, $\bar{a}$, by a factor of $k$. For $k$ parallel encryptors fed by $k$ decimated sequences, it is now possible to send the first element of the sequence $\bar{a}$ to the first encryptor, the second sequence element to the second encryptor, and so on, wrapping around feeding elements $k$+1, $2k$+1, $3k$+1,... to the first encryptor, $k$+2, $2k$+2, $3k$+2,... to the second encryptor and so on. By doing this, the same portion of the linear recurring

sequence is matched with the same portion of plaintext/ciphertext in the encryption/decryption units without regard to the degree of parallelism within communicating units.

In general, a companion matrix for an n cell, left shift register, would be a zero filled n x n matrix, with 1's along the diagonal just below the main diagonal, and feedback coefficients down the last column. Companion matrices allow manipulation of linear feedback shift registers using matrix arithmetic. The companion matrix corresponding to 1 shift of a 5 element, left shifting LFSR would be as shown. (This could also be considered as a decimation by 1 matrix.)

$$\begin{bmatrix} 0 & 0 & 0 & 0 & A_0 \\ 1 & 0 & 0 & 0 & A_1 \\ 0 & 1 & 0 & 0 & A_2 \\ 0 & 0 & 1 & 0 & A_3 \\ 0 & 0 & 0 & 1 & A_4 \end{bmatrix}$$

The elements of vector $A$ would contain a 1 if there was a tap at the corresponding element of the LFSR, and a 0 otherwise. The ones just below the main diagonal have the effect of shifting the elements left by one, that is, LFSR element $S_0$(new) = $S_1$(old), $S_1$(new) = $S_2$(old), etc. The new element of the sequence (the new $S_4$) will be an "Exclusive-or" of various elements in the register, based on possible taps at locations 0 through 4, as specified by vector $A$. A linear feedback shift register, 5 elements in length, with taps at elements 0, 1, and 3, would look like:




and be represented by vector the $A$ = [ 1 1 0 1 0 ]. This would yield the companion matrix shown.

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

To get the second decimation, or a matrix that would produce every other element in the sequence (2 shifts of the LFSR), square the companion matrix (mod 2). To get the fourth decimation, a matrix that would produce every fourth element in the sequence (shifting the LFSR 4 times), raise the companion matrix to the $4^{th}$ power (mod 2).

$$M^2 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix} \qquad M^4 = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Multiplying that resulting decimation matrix (as raised to the 4th power) by the initial state vector ($I_0$), mod 2, gives the vector that would be obtained after shifting the LFSR 4 times ($I_4$). Now, every 4th vector can be produced by multiplying the current vector ($I$) by $M^4$ mod 2.

The application of this is we can now seed 4 of these matrix multiplies, all in parallel, with 4 different initial vectors ($I_0$, $I_1$, $I_2$, $I_3$), representing the first 4 states of the linear recurring sequence. This will allow us to have 4 concurrent streams (each one generating every fourth element) of the same sequence.

In the linear feedback shift register shown above, the sequence generated by $S_m = S_{m-2} \oplus S_{m-4} \oplus S_{m-5}$ is given in the following table.

| | |
|---|---|
| State 0: | [ 1 0 0 0 0 ] |
| State 1: | [ 0 0 0 0 1 ] |
| State 2: | [ 0 0 0 1 0 ] |
| State 3: | [ 0 0 1 0 1 ] |
| State 4: | [ 0 1 0 1 0 ] |
| State 5: | [ 1 0 1 0 0 ] |
| State 6: | [ 0 1 0 0 1 ] |
| State 7: | [ 1 0 0 1 1 ] |
| State 8: | [ 0 0 1 1 0 ] |
| State 9: | [ 0 1 1 0 1 ] |
| State 10: | [ 1 1 0 1 1 ] |
| State 11: | [ 1 0 1 1 1 ] |

Table 1. State Vectors From 5 Element Linear Feedback Shift Register.

By using the first 2 states in the table above as our initial states, $I$, we can generate the next states by multiplying $IM^2$ mod 2 as shown in Table 2.

| | Generator 1 | Generator 2 |
|---|---|---|
| State 0: | [ 1 0 0 0 0 ] | |
| State 1: | | [ 0 0 0 0 1 ] |
| State 2: | [ 0 0 0 1 0 ] | |
| State 3: | | [ 0 0 1 0 1 ] |
| State 4: | [ 0 1 0 1 0 ] | |
| State 5: | | [ 1 0 1 0 0 ] |
| State 6: | [ 0 1 0 0 1 ] | |
| State 7: | | [ 1 0 0 1 1 ] |
| State 8: | [ 0 0 1 1 0 ] | |
| State 9: | | [ 0 1 1 0 1 ] |
| State 10: | [ 1 1 0 1 1 ] | |
| State 11: | | [ 1 0 1 1 1 ] |

Table 2. Five Element State Vectors From Decimation by 2 Matrix Multiplication.

As shown in Table 2, the first sequence generator is seeded with vector 0 and produces vectors 2, 4, 6, 8, 10, 12,... The second generator, seeded with vector 1, produces vectors 3, 5, 7, 9, 11, 13,...

By using the first 4 states in the table above as our initial states, $I$, we can generate the next states by multiplying $IM^4$ mod 2 as shown in Table 3.

| | Generator 1 | Generator 2 | Generator 3 | Generator 4 |
|---|---|---|---|---|
| State 0: | [ 1 0 0 0 0 ] | | | |
| State 1: | | [ 0 0 0 0 1 ] | | |
| State 2: | | | [ 0 0 0 1 0 ] | |
| State 3: | | | | [ 0 0 1 0 1 ] |
| State 4: | [ 0 1 0 1 0 ] | | | |
| State 5: | | [ 1 0 1 0 0 ] | | |
| State 6: | | | [ 0 1 0 0 1 ] | |
| State 7: | | | | [ 1 0 0 1 1 ] |
| State 8: | [ 0 0 1 1 0 ] | | | |
| State 9: | | [ 0 1 1 0 1 ] | | |
| State 10: | | | [ 1 1 0 1 1 ] | |
| State 11: | | | | [ 1 0 1 1 1 ] |

Table 3. Five Element State Vectors From Decimation By 4 Matrix Multiplication.

As illustrated in Table 3, the first sequence generator is seeded with vector 0 and produces vectors 4, 8, 12,... The second generator, seeded with vector 1, produces vectors 5, 9, 13,..., and so on. This can be repeated $k$ times (all operating in parallel) to attain the desired encryption rate.

At this point we should note that the sequences in Tables 1, 2, and 3 are identical. This shows that as long as the blocks are sent through the network in the proper order, units having differing degrees of parallelism will be able to interoperate.

Whereas this technique looks useful in concept, in reality it is not practical in most general purpose computers due to the amount of time necessary to carry out the multiplication (mod 2) of a 1 by $n$ vector with an $n$ by $n$ matrix. What can be done to make this technique practical, is to use logic gates rather than arithmetic operations to implement the matrix multiplication. Since addition modulo 2 can be represented by a logical "Exclusive-or" and multiplication modulo 2 can be represented by a logical "And", the matrix multiplication reduces to a set of logic gates, many of which can be wired in parallel. The general case would have each element in a column of the decimation matrix (say $M^4$) wired to an "And" gate along with the corresponding element of the current state vector. The output of these "And" gates can be fed into a cascade of "Exclusive-or" gates. This effectively performs the multiplications (mod 2) concurrently and then sums the results (mod 2) to produce the corresponding element of the new state vector. This is repeated (in parallel) for each column of the decimation matrix. This entire group of logic gates will have to be replicated $k$ times, once for each encryption module in the parallel encryption unit.

In each encryption module (of a unit containing $k$ encryption modules operating in parallel), for any characteristic polynomial of order $n$, $n^2$ "And" gates could be executed concurrently. There could also be $n$ cascades of "Exclusive-or" gates operating in parallel. These logic gates will also be operating $k$-times in parallel with the other encryption modules in the parallel encryption unit. This will result in the generation of the next state vector for use with each nonlinear encryption/decryption function, while incurring very few gate delays.

This process can be sped up further, if the LFSR taps are not part of the key material and can be defined at implementation time. In our example above, with taps at $S_0$, $S_1$, and $S_3$, and a decimation of 4, the new state vector for each generator could be constructed from the old state vector $[S_0, S_1, S_2, S_3, S_4]$ as: $S = [S_4 \quad S_0 \oplus S_1 \oplus S_3 \quad S_1 \oplus S_2 \oplus S_4 \quad S_0 \oplus S_1 \oplus S_2 \quad S_1 \oplus S_2 \oplus S_3]$. This logic would be repeated four times, once for each encryptor in the unit. The output from these logical operations, seeded as before with the initial vectors ($I_0$, $I_1$, $I_2$, $I_3$), will produce the same values as found in Table 3.

## Conclusion

This method of generating linear recurring sequences in parallel could be applied anywhere LFSRs are currently used, in order to achieve increased performance. As applied to increasing encryption rates, one could now design an encryption unit using $k$ decimated linear feedback shift register sequences feeding $k$ nonlinear encryption functions operating in parallel. Since these units will interoperate with each other, as higher encryption rates are needed, the number of encryption modules operating in parallel ($k$) within a unit can be scaled, without rendering previous versions obsolete. These slower units can still be used in portions of the network where top encryption speeds are not necessary.

## Acknowledgements

The work described in this paper was performed by Sandia National Laboratories and RE/SPEC Inc. under RE/SPEC contract number 56-4484 to Sandia National Laboratories and Sandia

contract number DE-AC04-94AL85000 to the United States Department of Energy. The authors would like to thank Jim Davis (ret.) of Sandia National Laboratories and Jed Greene of Northwestern University for their contributions to this work.

## Bibliography

1. "DES Modes of Operation" (FIPS PUB 81), Federal Information Processing Standards Publication 81, U. S. National Bureau of Standards, Washington, D. C., December 2, 1980.

2. Golomb, Solomon W., Shift Register Sequences, Holden-Day Inc., San Francisco, 1967.

3. Lempel, Abraham, and W. L. Eastman, "High Speed Generation of Maximal Length Sequences," IEEE Transactions on Computers, Vol. C-20, February, 1971.

4. Lidl, Rudolf, and Harald Niederreiter, Finite Fields, Vol. 20, Encyclopedia of Mathematics and Its Applications, Addison-Wesley, Reading, Massachusetts, 1983.