Secure Communications on the Battlefield?

Gary Krahn

History

For two days in June 1942, the fate of the U.S. Fleet (rebuilt since Pearl Harbor) hung on the meaning of two letters - AF. Between December 1941 and June 1942, the Japanese Navy under the command of Admiral Yamamoto had swept from one victory to another. Yamamoto knew it was essential to achieve victories in rapid succession. Given time, America could starve Japan of fuel. At Midway he would deal the U.S. Fleet a crushing defeat that would put it permanently out of action.

On May 20, 1942 Yamamoto broadcast his plans for the final destruction of the U.S. Fleet in a series of orders to his own fleet. At the same time the U.S. Combat Intelligence Unit at Pearl Harbor started to break into the stream of five-digit code groups that concealed the admiral's intentions. The projected battle began to emerge, however, the when and where remained hidden in mysterious letter groups that defied analysis. The key location was coded: AF.

The U.S. crypto unit employed one of the oldest tricks in the book. Using a code that they knew the Japanese had already broken, they arranged for the U.S. Garrison on Midway to broadcast the news that they were short of fresh water. Two days later, Yamamoto broadcasts to his fleet, "AF is short of freshwater."

Admiral Nimitz was now able to get his fleet to Midway and surprise the Japanese. The Americans destroyed all four of the big Japanese aircraft carriers that had attacked Pearl Harbor. As Admiral Nimitz later observed: "Midway was a victory of intelligence."

Introduction

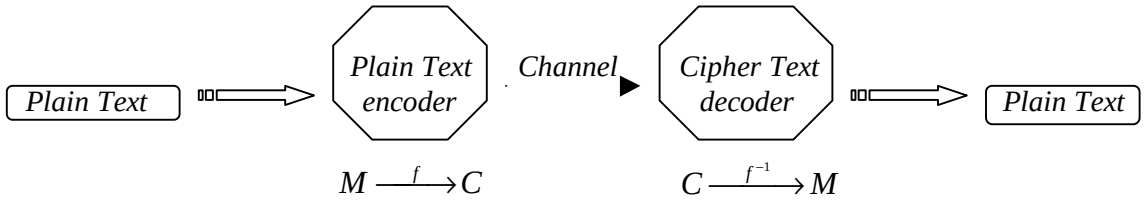
On the battlefield we must have the ability to transmit information quickly and securely. Information management is a combat multiplier that is critical to implementing the principals of maneuver, surprise, and initiative on the modern battlefield. Successful military units must be able to send information in a disguised form so that only the intended recipients can remove the disguise and read the message. Whether positioning units,

disseminating logistics, or coordinating an attack, secure information is vital for military operations.

The Word *Cryptography* is from the Greek 'kryptos' (hidden) and 'graphein' (to write). The traditional goal of cryptography has been to ensure privacy in communication by transforming data to render it unintelligible to all but the intended recipient. This can be achieved through the use of an encoding scheme that relies on a secret *key* known only by the sender and intended recipient. The message we want to send is called the *plaintext* and the disguised message is called the *ciphertext*. The message may be formed by using only the familiar symbols A – Z. If we don't include blanks, however, then all of the words are run together and the messages are harder to read. The process of converting a plaintext to a ciphertext is called *enciphering* or *encryption*, and the reverse process is called *deciphering*. Suppose the information we are to transmit comes from the set of symbols {A, B, C, . . . , Z}. Using binary communications we can associate a sequence of 0's and 1's with each of these symbols. For example,

A → 00000
B → 10100
C → 00001
D → 01010
E → 10011
G → 11111.

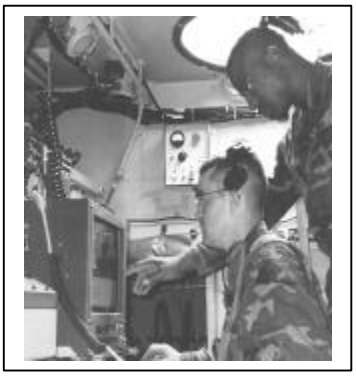
Hence, a 10100 00000 01011 sequence represents the message "BAD." In the hostile environment of the battlefield we want information to remain secret. Furthermore, we want to disguise the message to the enemy; however, we want our friendly units to be able to convert the disguised message into its original form. We can represent this process by the following simple model:



An enciphering transformation is a *function* that takes a plaintext message and gives us a ciphertext message. In other words, it is a mapping f from the set $\boldsymbol{M}$ of all possible plaintext messages to the set $\boldsymbol{C}$ of all possible ciphertext messages. We can represent this transformation schematically by the diagram: $\mathbf{M} \xrightarrow{f} \mathbf{C} \xrightarrow{f^{-1}} \mathbf{M}$, where f^{-1} is the map for deciphering.

Exercise 0: If the map (*f*) is not a one-to-one mapping (or function), what difficulties may the receiver encounter when using the map f^{-1} during deciphering.

Solution: Since the map (f) is not one-to-one then f^{-1} is not a function and it is possible during deciphering that an enciphered letter is mapped to more than one plaintext letter. Therefore, the receiver may not be able to determine with absolute certainty the actual plaintext message.



Modern high-speed communication systems handle information in binary form. To disguise or encipher a binary message during transmission one could *randomly* change the bits of the message. An efficient technique to encipher is to add, bit by bit modulo 2, a *random* binary sequence, ***S***, to the message, ***M***, generating a ciphertext. The receiver, also knowing ***S***, can then add ***S*** to the ciphertext bit by bit modulo 2 to retrieve the original message, ***M***. This *random* binary sequence ***S*** can not be truly random. It should, however, possess properties associated with a random process.

Note: Modulo 2 addition is defined as follows: $1 \oplus 1 = 0$; $1 \oplus 0 = 1$; $0 \oplus 1 = 1$; and $0 \oplus 0 = 0$.

Example 1: Suppose ***M*** is the message 10100 00000 01011 (representing the word BAD). We define a function f from the message set to the cipher set by

$$f(M) = M \oplus 10101\ 01011\ 01011.$$

In other words, f adds the sequence (or key) **S** =10101 01011 01011 bit by bit modulo 2 to ***M*** to form ***C***.

$$\begin{array}{ll} 10100\ 00000\ 01011 & (\boldsymbol{M}) \\ \oplus\ \underline{10101\ 01011\ 01011} & (\boldsymbol{S}) \\ 00001\ 01011\ 00000 & (\boldsymbol{C}) \end{array}$$

The receiver deciphers the ciphertext ***C*** by adding the key **S** modulo 2 to **C** to recreate the message **M**.

$$\begin{array}{ll} 00001\ 01011\ 00000 & (\boldsymbol{C}) \\ \oplus\ \underline{10101\ 01011\ 01011} & (\boldsymbol{S}) \\ 10100\ 00000\ 01011 & (\boldsymbol{M}) \end{array}$$

If the key, S, is random-looking then the resulting ciphertext, C, tend to be random-like. We may apply the key to many messages throughout some period of operations. Therefore, there is a need for high-speed techniques to generate random-like sequences. One of the simplest and most efficient devices for generating or modeling deterministic, random looking sequences of 0's and 1's is the *shift-register*.

The usefulness of sequences generated by a shift-register depends in large part on their randomness properties. In a sense, no finite sequence is truly random.

In particular, no sequence that depends on a few parameters, such as the feedback connections of a linear feedback shift-register, can be considered truly random. Solomon W. Golomb introduced the name *pseudo-random* for periodic binary sequences because they satisfy the three randomness properties of balance, runs, and correlation. We will discuss these properties later.

Feedback Shift-Register Design

A *binary shift-register* of span *n* is a collection $\{w(i): i = 0,1, \ldots, n\text{-}1\}$ of *n*-storage registers each capable of holding a value from the set {0,1}. The contents of the
n-storage registers, the *n*-tuple $(s_j, s_{j+1}, \ldots, s_{j+n-1})$, is denoted as the *state* of the register at time *j* for each $j \geq 0$. An example is shown in Figure 1.

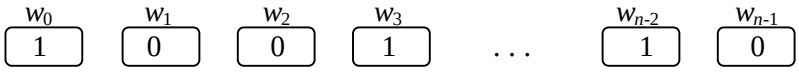


Figure 1: *n* Storage Devices

There is a feedback rule computed from the contents of the *n*-storage registers called the *feedback function* (Figure 2). If the feedback function is

$$f(s_j, s_{j+1}, \ldots, s_{j+n-1}) = s_{j+n} = c_0 s_j \oplus c_1 s_{j+1} \oplus \ldots \oplus c_{n-1} s_{j+n-1} = \sum_{k=0}^{n-1} c_k s_{j+k}, \quad 0 \leq j,$$

where the coefficients $c_0, c_1, \ldots,$ and c_{n-1} are 1's or 0's, and the summation is modulo 2 addition the shift register is called *linear*.

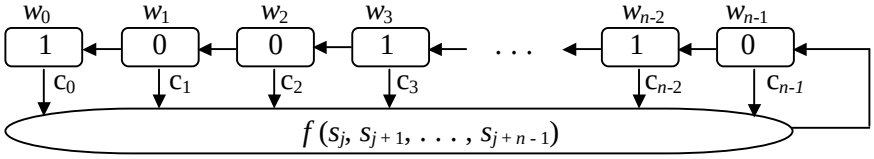


Figure 2: Feedback Shift-Register Model

At the pulse of an external clock the content of the storage register w_{i+1} is shifted into w_i for i =0, 1,. . . , *n*-2 and the value of the computed feedback function
$f(s_j, s_{j+1}, \ldots, s_{j+n-1})$ is shifted into storage register w_{n-1}.

Example 2: Using the above notation, let $n = 4$, $c_0 = c_1 = 1$, $c_2 = c_3 = 0$. Thus, $s_{j+4} = s_j \oplus s_{j+1}$ modulo 2. The wiring of this linear feedback shift-register (LFSR) is

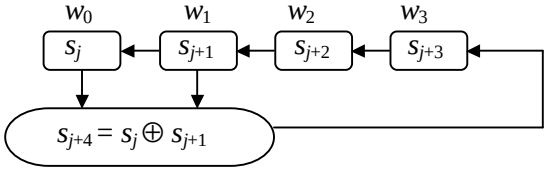
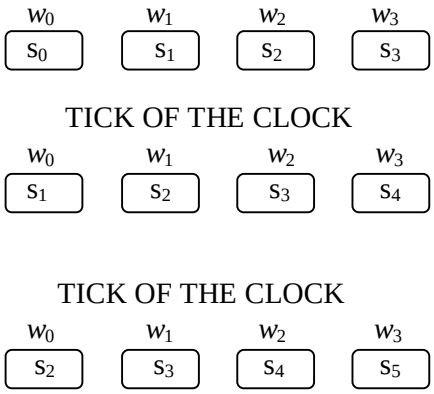


Figure 3: A Linear Feedback Shift-Register Model

Note: the connections from s_{j+2} and s_{j+3} are open since $c_2 = 0$ and $c_3 = 0$. Successive iterations from the initial configuration in Figure 3 look like:



The successor of the register fill vector $s_0, s_1, \ldots, s_{n-1}$ is the vector $s_1, s_2, \ldots, s_n$, where the value s_n is the computed feedback function $f(s_0, s_1, \ldots, s_{n-1})$. To generate successive states we iterate this procedure. With each tick of the clock, the register completes another step through a sequence of states. If we set the initial fill of the register to be 0001, such that $s_0 = 0$, $s_1 = 0$, $s_2 = 0$, and $s_3 = 1$, then the successive states of the register are:

0	0	0	<u>1</u>
0	0	1	<u>0</u>
0	1	0	<u>0</u>
1	0	0	<u>1</u>
0	0	1	<u>1</u>
0	1	1	<u>0</u>
1	1	0	<u>1</u>
1	0	1	<u>0</u>
0	1	0	<u>1</u>
1	0	1	<u>1</u>
0	1	1	<u>1</u>
1	1	1	<u>1</u>
1	1	1	<u>0</u>
1	1	0	<u>0</u>
1	0	0	<u>0</u>
0	0	0	1

For this example, the state sequence of register fills repeats with a period of 15. The history of stage w_3, which is underlined, is the sequence {100110101111000}, periodically repeated. If the contents of the register are initially not all 0's, the linear shift-register will cycle through all 15 different states before repeating itself. In general, it is possible to construct a span *n* linear

shift-register of the form in Figure 2 that will cycle through all 2^n-1 different states before repeating itself for every $n \geq 1$.

Associated with each linear feedback shift-register (LFSR) is the *characteristic polynomial*

$$C(x) = f(x) = x^n + \sum_{j=0}^{n-1} c_j x^j .$$

In Example 2 the characteristic polynomial is $x^4 + x + 1$.

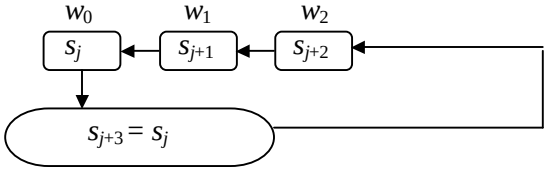
Exercise 1: Explain why any linear shift-register of span *n* will generate a sequence that is ultimately periodic with a period $p \leq 2^n$-1. (Sequences being the history of a selected storage register.)

> **Solution**: Each state of the shift register is completely determined by the previous state and the feedback function. Since there are only a finite number of states, some state must eventually repeat. Working with *n*-stage shift registers with 0's and 1's, the size of the possible state set is 2^n. The linearity of the feedback function guarantees that the all zero state is always its own successor. Therefore, the longest possible shift register cycle contains all of the nonzero states, and is periodic of period 2^n –1.

A sequence generated by a linear feedback shift-register of span *n* having a period
2^n –1 is called a maximum length linear shift register sequence or more simply an
m-sequence.

Exercise 2: Construct the LFSR with the characteristic polynomial $x^3 + 1$. Furthermore, if the initial state of this shift-register is 001, write the periodic sequence that it generates.

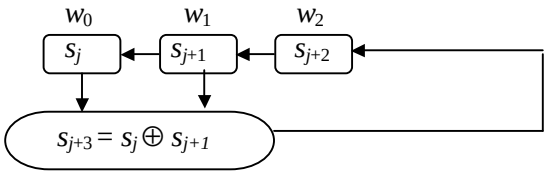
> **Solution**: : Using the above notation, let *n* = 3, $c_0 = 1$, $c_2 = 0$.
> Thus, $s_{j+3} = s_j$ modulo 2. The wiring of this LFSR is as follows:



> The sequence is 100 of period 3. This LFSR does not generate an *m-seq*uence of length 7.

Exercise 3: Construct the LFSR with the characteristic polynomial $x^3 + x + 1$. Furthermore, if the initial state of the shift-register is 001, write the periodic sequence that is generated.

Solution: Using the above notation, let $n = 3$, $c_0 = 1$, $c_2 = 1$. Thus, $s_{j+3} = s_j \oplus s_{j+1}$ modulo 2. The wiring of this LFSR is as follows:



The sequence is 1011100 of period 7. This LFSR does generate an *m-seq*uence of length 7.

Randomness Properties of *m-seq*uences

The usefulness of *m-seq*uences depends in large part on their having nearly ideal randomness properties. The randomness properties that we would like a sequence to have are given below.

The Balance Property: In a sequence the number of ONEs is the same as the number of ZEROs.

The Run Property: Among the runs of ONEs and ZEROs in a sequence, one-half the runs are of length one, one-fourth are of length two, one-eighth are of length three, and so on, as long as these fractions give meaningful number of runs.

The Correlation Property: When a sequence is compared term–by–term with a cyclically shifted version of the same sequence, the number of agreements equals the number of disagreements.

We will discover that *m-seq*uences satisfy the above randomness properties as closely as possible for each period of the sequence.

Exercise 3: Given that every non-zero *n*-tuple is seen as a state within a LFSR exactly once during the generation of an *m-seq*uence, explain why the balance property holds for every *m-seq*uence. In particular, there are 2^{n-1} ONEs and $2^{n-1} - 1$ ZEROs in an *m-seq*uence of length $2^n - 1$.

Solution: An n stage shift-register generating an *m-seq*uence cycles through all $2^n - 1$ states before it repeats. In decimal notation, each state can be thought of representing an integer from 1 to $2^n - 1$. From 1 to $2^n - 1$ there are 2^{n-1} odd integers and 2^{n-1} even integers. Thus, an

*m-seq*uence contains 2^{n-1} ONEs and 2^{n-1} –1 ZEROs and will therefore always possess the balance property.

Example 3: Given below is an *m-seq*uence of length 63 generated from a span *n* = 6 LFSR. Marked on the sequence are the runs of length greater than 2.

$$\underset{6}{\underline{111111}}0101\underset{2}{\underline{11}}00\underset{2}{\underline{11}}0\underset{3}{\underline{111}}0\underset{2}{\underline{11}}0100\underset{3}{\underline{111}}000101\underset{4}{\underline{1111}}00101000\underset{2}{\underline{11}}0000100000$$

Exercise 4: In Example 3 there are a total of 32 runs, including the singleton runs. One half of the runs (16) are of length one, one fourth (8) are of length two, one eighth (4) are of length three, and one sixteenth (2) are of length four. For each of these lengths there are equally many runs of ZEROs and ONEs. Finally, there is one run of five consecutive ZEROs and one of six consecutive ONEs. Explain why the run property holds for all *m-seq*uences.

Solution: In an *m-seq*uence of period 2^n –1, every non-zero *n*-tuple occurs exactly once. If the *n*-tuple 000…0 occurs, the sequence will remain forever in the state 000…0 . Therefore, the *n*-tuple, 000…0, nevers occurs in an *m-seq*uence. Since every non-zero *n*-tuple occurs, the *n*-tuple 111…1 must occur exactly once with a zero preceding and following this all ONE *n*-tuple. The (n+2)-tuple, 011…10, contains the *n*-tuples [011…1]10 and 01[1…110]. These *n*-tuples are not repeated elsewhere in the sequence. Hence, there can be no run of *n*-1 ONEs in the sequence. However, there is exactly one run of length *n*-1 of ZEROs, represented in the consecutive *n*-tuple 10…00 and 00…01. Now consider the runs of ONEs and length *r* where $0 < r \le n-2$. Each such run can be made to correspond to *n*-tuples of the type

$$0\underset{r}{\underline{11\ldots1}}0\,\underset{n-r-2}{\underline{xx\ldots x}}$$

where the *x*'s are arbitrary bits. The number of such *n*-tuples of run length *r* is 2^{n-r-2}. A similar argument gives the same number of runs of ZEROs of length *r*. This completely determines the *run* structure of *m-seq*uences.

Experiment: Compare an *m-seq*uence $\{s_i\}$ with a cyclic shift of itself. This comparison is called *autocorrelation*. Suppose an *m-seq*uence is added to itself bit by bit using modulo 2 addition. When the sequence is added in phase with itself, we get the null sequence. When the sequence is added to each out of phase shift of the sequence a miracle occurs. What is this miracle? Explain. Do *m-seq*uences have the correlation property? Explain.

Solution: When the sequence is added to each out of phase shift of the sequence, the resulting sequence is a non-zero sequence that is a shifted version of the original sequence. This property is known as the *Shift-and-Add property*. For example, 1011100, denoted by $\{s_i\}$, is an *m-seq*uence. Let $\{s_{i+1}\}$ denoted the sequence $\{s_i\}$ shifted to the left by one bit. We find that $\{s_{i+1}\} = 0111001$. When $\{s_{i+1}\}$ and $\{s_i\}$ are added modular 2 (bit by bit) the resulting sequence is 1100101. A (1) in the resulting sequence means that $\{s_{i+1}\}$ and $\{s_i\}$ disagreed in that position while a (0) means that $\{s_{i+1}\}$ and $\{s_i\}$ agreed in that position. Since the resulting sequence is an *m-seq*uence it follows that the number of agreements and disagreements between an *m-seq*uence and a shifted version of the same *m-seq*uence are just about equal. Hence, *m-seq*uences possess the correlation property.

Examples of the shift-and-add property are given below:

s_0	1011100	s_0	1011100	s_0	1011100
s_1	$\oplus$ 0111001	s_2	$\oplus$1110010	s_4	$\oplus$1001011
s_3	1100101	s_6	0101110	s_5	0010111

We say that the shift-and-add pairs for the sequence 1011100 are (1,3), (2,6), and (4,5), e.g., the sequence 1011100 added modulo 2 to a shifted (by 3) version of itself produces a shifted (by 1) version of itself.

Because *m-seq*uences satisfy the three randomness properties they are sometimes referred to as *pseudo-random* or *pseudo-noise* sequences. Moreover, *m-seq*uences are the only deterministic sequences that have the randomness properties and distinct *n*-tuples in each period of length $2^n - 1$.

Exercise 5: Let us say a segment from an unknown *m-seq*uence (**S**) is added to a shifted (by 3) version of itself:

(**S**)	101011101100. . .0111110011	Unknown *m-seq*uence
($\mathbf{S}_3$) $\oplus$	011101100011. . .1110011010	shift of 3
($\mathbf{S}_5$)	110110001111. . .1001101001	shift of 5

We can see that (3,5) is a shift and add pair for the sequence (S). Furthermore, the sum $\mathbf{S} \oplus \mathbf{S}_3 \oplus \mathbf{S}_5 = 000\ldots0000$, the all zero sequence.

(**S**)	101011101100. . .0111110011	Unknown *m-seq*uence
($\mathbf{S}_3$)	011101100011. . .1110011010	shift of 3
($\mathbf{S}_5$) $\oplus$	110110001111. . .1001101001	shift of 5
	000000000000000000000000	**SUM**

What if the unknown *m-seq*uence (**S**) was copied by a person who on the average made 10% errors in the sequence at *random* as she records the sequence (**S'**) from a digital communication network. The mod 2 sum of (**S'**) and the sequence of shifts of 3 and 5, i.e., $\mathbf{S'} \oplus \mathbf{S'}_3 \oplus \mathbf{S'}_5 = \mathrm{S}$ would no longer be the zero sequence. In fact, it would give us a sequence (S) with some percentage of ones. How would you estimate the density of ones in (S) if (3,5) was truly a shift-and-add pair of (**S**)?

Solution: The sequence (**S**) is subjected to random errors at a rate of 10%. That is, the probability that each bit in the original sequence (**S**), and in the shifted sequences, is in error is (.10). If the shifts of 3 and 5 constitute a valid shift and add pair then $\mathbf{S} \oplus \mathbf{S}_3 \oplus \mathbf{S}_5$ is the zero sequence. The rows of the matrix [A] below are composed of **S**, $\mathbf{S}_3$, and $\mathbf{S}_5$.

$$A = \begin{bmatrix} 101011101100\ldots0111110011 \\ 011101100011\ldots1110011010 \\ 110110001111\ldots1001101001 \end{bmatrix}$$

The probability that column (i) in matrix A contains :

1. One error is (.1)(.9)(.9) C(3,1) = .243
2. Two errors is (.1)(.1)(.9) C(3,2) = .027
3. Three errors is (.1)(.1)(.1) = .001
4. Zero errors is (.9)(.9)(.9) = .729

Note: C(*n*,*r*) = (*n* !) / *r* !(*n*-*r*)! is the number of different ways of selecting (*r*) elements from a set with (*n*) elements. Since the summation is mod 2, one or three errors in column (*i*) will generate a one in the sequence (S). Therefore, the density of ones in (S) is approximately 24.4%.

Exercise 6: Given that someone has injected 20% errors in an *m-seq*uence (**S**), how would you verify that (*a*,*b*) is a shift-and-add pair for (**S**)?

Solution: Using the same analysis as above, if (*a*,*b*) is a shift-and-add pair for the sequence **S**, we would find that $\mathbf{S} \oplus \mathbf{S}_3 \oplus \mathbf{S}_5$ would have 39.2% errors, since (.2)(.8)(.8)C(3,1) + (.2)(.2)(.2) = .392. If (*a*,*b*) is not a shift-and-add pair $\mathbf{S} \oplus \mathbf{S}_3 \oplus \mathbf{S}_5$ would have nearly 50% errors.

As we have seen, *m-seq*uences have some very interesting properties. For nearly four decades the idea of using shift registers to generate sequences of 1'and 0's has been explored, developed, and refined. There are, however, many more properties to be discovered. Bon appetite!

Exercises

1. Suppose $x^5 + x^3 + 1$ is the characteristic polynomial of a linear feedback shift-register. Wire the shift-register and generate the *m-seq*uence.

2. Find the set of shift-and-add pairs for the sequence.

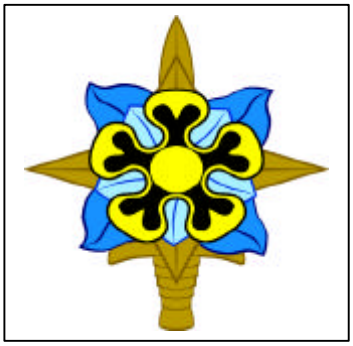
3. Consider the summation of the three sequences **S**, $\mathbf{S}_j$, and $\mathbf{S}_k$ $(\mathbf{S} \oplus \mathbf{S}_j \oplus \mathbf{S}_k)$ where *j* and *k* are shift-and-add pairs for the sequence **S**. Determine the percentages of ones in the sequence $\mathbf{S} \oplus \mathbf{S}_j \oplus \mathbf{S}_k$ when there is a 20% error rate in **S**.

4. Write a paragraph that explains how shift-and-add pairs may be used to determine the generator of a sequence?

Capstone Historical Vignette

ULTRA was the name given for the Allies ability to listen in on the most private communications of the Nazi leaders during World War II. The German Enigma device was an electrical machine used to encode messages. Its keyboard was similar to a typewriter, but the letters struck rotary disks that selected substitute letters at random to create encoded messages. England's survival of German air attacks, Dunkirk, North African Campaign, El Alamein, and V-1 rockets program all have a rich history of ULTRA and code breaking. The final and crucial contribution of ULTRA was in Operation Overlord, the Allied invasion of mainland Europe from the British Isles.

In 1943 the code-breakers had learned that General K. von Rundstet feared the Allies would arrive on the continent by way of Calais. To foster that notion, the Allied powers mounted a massive diversionary exercise, putting an entire ghost army opposite the far northern beaches of France. By the Spring of 1944, the ULTRA teams listened in to the high-level debate between Hitler and his hard-pressed generals on the best way to meet the powerful Allied attack. The rest is history, but the real story is mathematics.



References

[1] Golomb, S. W., Shift Register Sequences, Aegean Park Press, Laguna Hills, CA, 1982.

[2] Kahn, David, Kahn on Codes, Macmillan Publishing Company, New York, 1983.

[3] Kahn, David, Code Breakers, Macmillan Publishing Company, New York, 1967.

[4] Koblitz, Neal, A Course in Number Theory and Cryptography, Springer-Verlag, 1988.

[5] Way, Peter, Ciphers and Codes, Crown Publishers, United Kingdom, 1977.