Applications of Recursive Sequences and Generating Functions in Cryptography

Research Question:

How are ordinary generating functions (OGFs) used in generalizing recursive sequences, and to what extent are the applications of OGFs and recursive sequences in the device of linear feedback shift registers useful in cryptography?

Subject: Mathematics AA HL

Candidate Code: jfx450

Word Count: 3998

Contents

0. Defining Equations

Generating function/ formal power series:

$$G(x) = \sum_{k=0}^{\infty} a_k x^k = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \dots + a_k x^k$$

Sum of infinite geometric series:

$$S_\infty = \frac{1}{1-x} = 1 + x + x^2 + \dots \qquad |x| < 1$$

General formula of a recursive sequence of an LFSR:

$$s_j = \sum_{i=1}^{L} c_i s_{j-i} = c_1 s_{j-1} \oplus c_2 s_{j-2} \oplus \dots \oplus c_{L+1} s_{j-L+1} \oplus c_L s_{j-L}, \qquad j \geq L, c \in \{0, 1\}, s \in \{0, 1\}$$

General formula of a connection polynomial of an LFSR:

$$F(x) = \sum_{i=0}^{L} c_i x^i = c_0 \oplus c_1 x \oplus c_2 x^2 \oplus \dots \oplus c_{L-1} x^{L+1} \oplus c_L x^L, \qquad \text{where } c_0 = c_L = 1$$

1. Introduction

A recursive sequence is one whose terms are defined by one or more previous terms in the sequence (Weisstein). A generating function is "an elegant and powerful technique" of generalizing an infinite sequence, such as a recursive sequence, as a single mathematical object. In the text "Concrete Mathematics: A Foundation for Computer Science", generating functions are described as "the most important idea in the whole book" ("Generating Functions"). The concept of generating functions was invented by Abraham De Moivre, who is well-known for his contributions to mathematical topics including complex numbers, trigonometry, and probability (Dobrushkin).

I was introduced to this concept through a summer course which covered topics within combinatorics such as permutations and combinations, derangements, and generating functions. I found generating functions particularly interesting as they have applications in computer science, a field I hope to pursue in university. Specifically, generating functions and their relationship to recursive sequences have applications in cryptography. Although the majority of this essay will be outside of the AA HL syllabus, I will use the concepts of sequences and series and partial fractions from Number and Algebra when I find closed forms for recursive sequences.

An ordinary generating function (OGF) takes a sequence of numbers and makes it the coefficients of a formal power series. The general form of an OGF is:

$$G(x) = \sum_{k=0}^{\infty} a_k x^k$$

A power series is an infinite sum of terms where a_k represents the coefficient of x, or the kth term of the sequence. It refers to the expanded form of the summation in $G(x)$:

$$G(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + ... + a_kx^k$$

Examples of formal power series of sequences include:

1. $1, 1, 1,.. \Rightarrow 1 + x + x^2 + x^3 +...$
2. $1, 0, 0,.. \Rightarrow 1$
3. $0, 1, 2, 3, 0, 0, 0, ... \Rightarrow x + 2x^2 + 3x^3$

The adjective "formal" refers to x being an abstract indeterminate, and we will not be concerned about convergence of the series for specific values of x ("Generating Functions"). In other words, rather than an actual number, x is regarded a placeholder to keep track of the coefficients of x^k.

Many types of generating functions exist, including ordinary generating functions, probability generating functions, and exponential generating functions. This essay will focus on OGFs, recursive sequences and their applications in cryptography. Cryptography refers to the "science of using mathematics to hide data". It involves encoding information with a key one must have to access the original data ("Guide to Cryptography Mathematics"). Many cryptosystems require the generation of random numbers for a key. However, it is impossible for a computer to produce truly random numbers as it is a machine that follows a set of instructions. Therefore, we rely on devices such as linear feedback shift registers (LFSRs) that allow us to generate pseudo-random numbers (numbers that are generated from an algorithm but seem statistically random).

Thus, through this essay, I will answer the research question "How are ordinary generating functions (OGFs) used in generalizing recursive sequences, and to what extent are the applications of OGFs and recursive sequences in the device of linear feedback shift registers (LFSRs) useful in cryptography?" by exploring how OGFs allow us to find closed forms of recursive sequences and how they are used in cryptography.

2. Ordinary Generating Functions

To better understand OGFs, consider the example of determining the number of ways to pay a fixed amount of money. Note that some of the information that I will use is from notes I took while attending the previously mentioned summer course.

If we wish to find the number of ways to pay 15 QAR (Qatari Riyals) in terms of 3 and 9 QARs, we can represent paying 3 QARs as x^3, 9 QARs as x^9, and paying 15 QAR as x^{15}. Right away, we can see that paying 3 QARs 5 times is a solution:

$$\left(x^3\right)^5 = x^{15}$$

Another solution is paying 9 QAR once and 3 QARs twice:

$$\left(x^9\right)\left(x^3\right)^2 = x^{9+(3\times2)} = x^{15}$$

The step-by-step solution to this is to first represent all possible ways to pay any amount using 3 QARs:

$$\left(x^3\right)^0 + \left(x^3\right)^1 + \left(x^3\right)^2 + \ldots = 1 + x^3 + x^6 + x^9 + x^{12} + x^{15} \ldots = \sum_{k=0}^{\infty} x^{3k}$$

We do the same with 9 QARs:

$$\left(x^9\right)^0 + \left(x^9\right)^1 + \left(x^9\right)^2 + \ldots = 1 + x^9 + x^{18} + x^{27} + \ldots = \sum_{k=0}^{\infty} x^{9k}$$

Multiplying the two OGFs results in:

$$\left(\sum_{k=0}^{\infty} x^{3k}\right) * \left(\sum_{k=0}^{\infty} x^{9k}\right)$$

$$= (1 + x^3 + x^6 + x^9 + \ldots)(1 + x^9 + x^{18} + x^{27} + \ldots)$$

$$= 1 + x^3 + x^6 + 2x^9 + 2x^{12} + 2x^{15} + 3x^{18} + \ldots$$

This expression shows all possible ways of paying an amount with both 3 QARs and 9 QARs. The coefficients of x^n represent all the number of ways of paying n QARs. Thus, paying 15 QARs is possible in two ways, since the coefficient of x^{15} is 2. 15 QARs can be paid using five 3 QARs, or using one 9 QAR and two 3 QARs. Observe that paying 1 QAR or 2 QARs is not possible because there is no coefficient for x^2 or x, since there is no way to pay those amounts using 3 QARs or 9 QARs. However, there is a coefficient 1 for x^0, which is the first term, 1. This indicates how we can choose not to pay at all in one way, by not paying either 3 QARs or 9 QARs. Although it would have been simple to deduce this, as we did in the beginning, OGFs are useful for calculations with more complex investigations. This example thus proves that OGFs are powerful as they allow us to manipulate an infinite sequence of numbers.

3. <u>Generalizing Recursive Sequences using Generating Functions</u>

In a recursive sequence, terms are defined using previous terms. Finding the general formula of an arithmetic or geometric pattern is straightforward and can be done without OGFs because they have a common ratio or difference. However, with recursive sequences, we need to use OGFs to find a closed form. A closed form helps us find the kth term (a_k) by simply knowing the value of k.

The Fibonacci sequence,

$F(x) = 0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ \ldots$

is an example of a recursive sequence that holds great mathematical significance. I will use it to illustrate how OGFs can solve recursive sequences.

A term in the Fibonacci sequence can be expressed as:

$a_n = a_{n-1} + a_{n-2}$,

where $n > 2$, $a_0 = 0$, and $a_1 = 1$

The OGF for the sequence, with its values as the coefficients of x^n is:

$$A(x) = \sum_{n=0}^{\infty} a_n x^n = 0 + 1x + a_2 x^2 + a_3 x^3 + \ldots.\ a_n x^n$$

Multiplying $A(x)$ by x, results in an index shift, in other words, a shift in how a_n aligns with x^n:

$$xA(x) = x(a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots.)$$

$$= a_0 x + a_1 x^2 + a_2 x^3 + a_3 x^4 + \ldots$$

This can also be written as :

$$\sum_{n=0}^{\infty} a_n x^{n+1}$$

Or:

$$\sum_{n=1}^{\infty} a_{n-1} x^n$$

Similarly, shifting the index to $\sum_{n=2}^{\infty} a_{n-2} x^n$ can be achieved by multiplying $A(x)$ by x^2:

$$x^2A(x) = x^2(a_0 + a_1x + a_2x^2 + a_3x^3 + \ldots.)$$

$$= a_0x^2 + a_1x^3 + a_2x^4 + \ldots$$

$$= \sum_{n=2}^{\infty} a_{n-2}x^n$$

Evaluating $A(x)$ - $xA(x)$ - $x^2A(x)$ results in:

$$(a_0 + a_1x + a_2x^2 + a_3x^3 + \ldots) - (a_0x + a_1x^2 + a_2x^3 + a_3x^4 + \ldots) - (a_0x^2 + a_1x^3 + a_2x^4 + \ldots)$$

$$= a_0 + a_1x - a_0x + (a_2 - a_1 - a_0)x^2 + (a_3 - a_2 - a_1)x^3 + \ldots.$$

The coefficients in the parenthesis for $x^2, x^3, \ldots\ x^n$ cancel out due to the Fibonacci formula:

$$a_n = a_{n-1} + a_{n-2}$$

$$a_n - a_{n-1} - a_{n-2} = 0$$

Since said coefficients follow the pattern ($a_n - a_{n-1} - a_{n-2}$), they all add up to 0, and we are left with:

$$a_0 + a_1x - a_0x$$

We know that $a_0 = 0$, and $a_1 = 1$, therefore:

$$a_0 + a_1x - a_0x = 0 - 0x + x = x$$

The expression that we have evaluated to be equal to x is $A(x)$ - $xA(x)$ - $x^2A(x)$, which can also be written as :

$$A(x)\,(1 - x - x^2) = x$$

Isolating $A(x)$ gives the result:

$$A(x) = \frac{x}{1 - x - x^2}$$

This generating function can be converted into a closed form using partial fractions and sum of infinite series. We can factor $1 - x - x^2$ as:

$$1 - x - x^2 = (x + \alpha)(x + \beta) = x^2 + \alpha x + \beta x + \alpha\beta$$

To make the coefficient of x^2 equal to -1, we will multiply the factored form by -1:

$$1 - x - x^2 = -(x + \alpha)(x + \beta) = -x^2 - \alpha x - \beta x - \alpha\beta$$

By comparison, we can see that:

1. $\alpha\beta = -1$
2. $\alpha + \beta = 1$

Rewriting the second expression gives:

$$\alpha = 1 - \beta$$

Substituting this into the first expression gives:

$$\beta(1 - \beta) =- 1$$

$$\beta^2 - \beta - 1 = 0$$

Substituting this into the quadratic formula gives:

$$\beta = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

In this case $a = 1,\ b =- 1$, and $c =- 1$, therefore:

$$\beta = \frac{1 \pm \sqrt{1 - 4(1)(-1)}}{2}$$

$\beta = \frac{1+\sqrt{5}}{2}$ or $\beta = \frac{1-\sqrt{5}}{2}$

Notice that both the above roots add up to 1, just like $\alpha + \beta$:

$$\frac{1+\sqrt{5}}{2} + \frac{1-\sqrt{5}}{2} = \frac{1}{2} + \frac{1}{2} = 1$$

Since $\alpha + \beta = 1$, we can say that:

$$\alpha = \frac{1+\sqrt{5}}{2}\ ,\ \beta = \frac{1-\sqrt{5}}{2}$$

Thus, we can factor $1 - x - x^2$ as:

$$1 - x - x^2 = -(x + \alpha)(x + \beta) = -(x + \frac{1+\sqrt{5}}{2})(x + \frac{1-\sqrt{5}}{2})$$

For the sake of keeping the working out neat, I will use the symbols α and β rather than writing out $\frac{1+\sqrt{5}}{2}$and $\frac{1-\sqrt{5}}{2}$. Using partial fraction decomposition, we can say that:

$$A(x) = \frac{x}{1 - x - x^2} = \frac{-x}{(x+\alpha)(x+\beta)}$$

$$= \frac{P}{x+\alpha} + \frac{Q}{x+\beta} = \frac{P(x+\beta)+Q(x+\alpha)}{(x+\alpha)(x+\beta)}$$

Thus:

$$-x = P(x+\beta) + Q(x+\alpha)$$

Expanding the equation gives:

$$-x = Px + Qx + P\beta + Q\alpha$$

By comparison, we can say that:

1. $P + Q = -1$
2. $P\beta + Q\alpha = 0$

Rearranging equation 1 gives:

$$P = -1 - Q$$

Substituting this into equation 2 gives:

$$-\beta - \beta Q + Q\alpha = 0$$

$$Q(-\beta + \alpha) = \beta$$

Using the values for β and α to evaluate $(-\beta + \alpha)$ we get:

$$-\beta + \alpha = \frac{-1+\sqrt{5}}{2} + \frac{1+\sqrt{5}}{2}$$

$$= \frac{2\sqrt{5}}{2}$$

$$= \sqrt{5}$$

Therefore,

$$\beta = \sqrt{5}Q$$

Rearranging for Q gives us:

$$Q = \frac{\beta}{\sqrt{5}}$$

To calculate P, we repeat this process. We rearrange equation 1 to make Q the subject:

$$Q = -1 - P$$

Then, we substitute Q into equation 2:

$$P\beta + Q\alpha = 0$$

$$P\beta + (-1 - P)\alpha = 0$$

$$P\beta - P\alpha - \alpha = 0$$

$$P(\beta - \alpha) = \alpha$$

Using the values for β and α to evaluate $(\beta - \alpha)$ we get:

$$\beta - \alpha = \frac{1-\sqrt{5}}{2} + \frac{-1-\sqrt{5}}{2}$$

$$= -\frac{2\sqrt{5}}{2}$$

$$= -\sqrt{5}$$

Therefore,

$$\alpha = -\sqrt{5}P$$

Rearranging for P gives us:

$$P = -\frac{\alpha}{\sqrt{5}}$$

Having calculated $P = -\frac{\alpha}{\sqrt{5}}$ and $Q = \frac{\beta}{\sqrt{5}}$, we can substitute them into $A(x)$:

$$A(x) = \frac{P}{x+\alpha} + \frac{Q}{x+\beta}$$

$$= \frac{-\alpha}{\sqrt{5}(x+a)} + \frac{\beta}{\sqrt{5}(x+\beta)}$$

We can factor out $\sqrt{5}$:

$$A(x) = \frac{1}{\sqrt{5}}\left(\frac{\beta}{(x+\beta)} - \frac{\alpha}{(x+a)}\right)$$

We can use the formula for the sum of infinite series to find a closed form for the two expressions in the parenthesis (Rochford):

$$S_{\infty} = \frac{u_1}{1-r},\ |r| < 1$$

3.1. Deriving the Formula for Sum of Infinite Series:

Consider the sequence :

$$\sum_{n=0}^{k} a_0 r^n = S = a_0 r^0 + a_0 r^1 + a_0 r^2 + \dots + a_0 r^k$$

When we multiply S by r we get:

$$Sr = r(a_0 r^0 + a_0 r^1 + a_0 r^2 + \dots + a_0 r^k)$$

$$= a_0 r^1 + a_0 r^2 + a_0 r^3 + \dots + a_0 r^{k+1}$$

Subtracting Sr from S cancels out everything except a_0 and $a_0 r^{k+1}$:

$$S - Sr = (a_0 r^0 + a_0 r^1 + a_0 r^2 + \dots + a_0 r^k) - (a_0 r^1 + a_0 r^2 + a_0 r^3 + \dots a_0 r^n + a_0 r^{k+1})$$

$$= a_0 r^0 - a_0 r^{k+1}$$

Therefore:

$$S - Sr = a_0 - a_0 r^{k+1}$$

$$S(1 - r) = a_0(1 - r^{k+1})$$

$$S = \frac{a_0(1-r^{k+1})}{1-r}$$

When the absolute value of the common ratio is less than 1, the sequence is said to be converging:

$$\lim_{k\to\infty} r^{k+1} = 0 \text{, when } |r| < 1$$

Therefore, replacing 0 with r^{k+1} gives us:

$$S = \frac{a_0(1-0)}{1-r}$$

$$= \frac{a_0}{1-r}$$

Thus,

$$\sum_{n=0}^{\infty} a_0 r^n = \frac{a_0}{1-r} \text{, when } |r| < 1, n \to \infty$$

Convergence is a property of formal power series algebra ("Generating Functions"). Therefore we can say that $r = x$ and define the sum of the generating function $\sum_{n=0}^{\infty} x^n$ as :

$$\sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + \ldots = \frac{1}{1-x}$$

where $x \to \infty$ and $|x| < 1$

Going back to $A(x) = \frac{1}{\sqrt{5}}\left(\frac{\beta}{(x+\beta)} - \frac{\alpha}{(x+a)}\right)$ we can use the formula that we have derived to rewrite $\frac{\beta}{(x+\beta)}$ and $\frac{\alpha}{(x+a)}$ as generating functions. First, for $\frac{\beta}{(x+\beta)}$, we divide the numerator and the denominator of the term by β so that the numerator is equal to 1, as in the formula for the sum of infinite series:

$$\frac{\frac{\beta}{\beta}}{\frac{x}{\beta}+\frac{\beta}{\beta}} = \frac{1}{1+\frac{x}{\beta}}$$

From pg. 10, recall that:

$$\alpha\beta = \; -1$$

Rearranging for α gives:

$$\alpha = \frac{-1}{\beta}$$

We can substitute this into $\frac{1}{1+\frac{x}{\beta}}$:

$$\frac{1}{1+\frac{x}{\beta}} = \frac{1}{1-\alpha x}$$

Since the common ratio is αx, the formula for the sum of infinite series gives us:

$$\frac{1}{1-\alpha x} = \sum_{n=0}^{\infty} \alpha^{n} x^{n}$$

We repeat this process for $\frac{\alpha}{(x+a)}$:

$$\frac{\alpha}{(x+a)} = \frac{1}{1+\frac{x}{\alpha}}$$

$$= \frac{1}{1-\beta x}$$

Using the formula for the sum of infinite series gives:

$$\frac{1}{1-\beta x} = \sum_{n=0}^{\infty} \beta^{n} x^{n}$$

Thus:

$$\frac{β}{(x+β)} = \sum_{n=0}^{\infty} α^n x^n \quad , \quad \frac{α}{(x+a)} = \sum_{n=0}^{\infty} β^n x^n$$

We can substitute these generating functions into $A(x)$:

$$A(x) = \frac{1}{\sqrt{5}} \left(\frac{β}{(x+β)} - \frac{α}{(x+a)}\right)$$

$$A(x) = \frac{1}{\sqrt{5}} \left(\sum_{n=0}^{\infty} α^n x^n - \sum_{n=0}^{\infty} β^n x^n \right)$$

We can combine the two summations into:

$$A(x) = \frac{1}{\sqrt{5}} \left(\sum_{n=0}^{\infty} (α^n - β^n) x^n \right)$$

We can insert the constant $\frac{1}{\sqrt{5}}$ into the summation:

$$A(x) = \sum_{n=0}^{\infty} \frac{1}{\sqrt{5}} (α^n - β^n) x^n$$

Recall that the definition of $A(x)$ is:

$$A(x) = \sum_{n=0}^{\infty} a_n x^n$$

Where a_n represents the nth term of the Fibonacci sequence. The coefficient of x^n in the summation is given by $\frac{1}{\sqrt{5}} (α^n - β^n) x^n$:

$$A(x) = \sum_{n=0}^{\infty} a_n x^n = \sum_{n=0}^{\infty} \frac{1}{\sqrt{5}} (α^n - β^n) x^n$$

Therefore,

$a_n = \frac{1}{\sqrt{5}}(\alpha^n - \beta^n)$

Finally, we can substitute the previously calculated values for α and β to get the closed form of the Fibonacci:

$a_n = \frac{1}{\sqrt{5}}((\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n)$

We can test this closed form for a few values of the Fibonacci to ensure it works. I will test it out for a_0, a_1 and a_2, which we know to be 0, 1, and 1 respectively.

$a_0 = \frac{1}{\sqrt{5}}((\frac{1+\sqrt{5}}{2})^0 - (\frac{1-\sqrt{5}}{2})^0)$

$= \frac{1}{\sqrt{5}}(1-1)$

$= 0$

$a_1 = \frac{1}{\sqrt{5}}((\frac{1+\sqrt{5}}{2})^1 - (\frac{1-\sqrt{5}}{2})^1)$

$= \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2} + \frac{-1+\sqrt{5}}{2})$

$= \frac{1}{\sqrt{5}}(\frac{2\sqrt{5}}{2})$

$= 1$

$a_2 = \frac{1}{\sqrt{5}}((\frac{1+\sqrt{5}}{2})^2 - (\frac{1-\sqrt{5}}{2})^2$

$= \frac{1}{\sqrt{5}}(\frac{1+5+2\sqrt{5}}{4} + \frac{-1-5+2\sqrt{5}}{4})$

$$= \frac{1}{\sqrt{5}} \left(\frac{4\sqrt{5}}{4} \right)$$

$$= 1$$

This example reflects how powerful OGFs are in finding closed forms, and how their properties can be used to simplify the most complex of sequences. After exploring how OGFs help in finding the closed form of recursive sequences, I decided to look at a more real-life approach: the applications of recursive sequences and OGFs in cryptography.

4. Linear Feedback Shift Registers

A linear feedback shift register is a device that generates a pseudo-random sequence of binaries (0s and 1s). The input of an LFSR is dependent on its previous states. The pseudo-random numbers generated by LFSRs are used as keys in cryptographic systems. Since an LFSR is a type of recursive sequence, generating functions are used to represent cryptosystems that use LFSRs.

A Vernam cipher is a cipher in which each character is encrypted using its own key (Suzuki, "Vernam Cipher"). Using a Vernam cipher, I will demonstrate why random numbers are necessary in cryptography.

4.1. Vernam Ciphers:

In Vernam ciphers, binary plaintext (text that needs to be encrypted), is encrypted using the XOR operation, which is also called the modulo 2 addition operation. The XOR operation performs an exclusive disjunction on each plaintext character and binary key; they must be different values for the ciphertext to be a 1, if they are the same it is a 0 (Zaczyński). The arithmetic behind this operation involves a sum of the key and plaintext, and a modulo 2 operation on this sum. There are three possible sums:

1. $0 + 0 = 0$
2. $0 + 1 = 1$
3. $1 + 1 = 2$

A modulo operation performs division and returns the remainder (Zaczyński) and is represented as:

$p \bmod q$, where p is the dividend and q is the divisor. In modulo 2 operation, 2 is the divisor. An example of mod 2 operation is:

$5 \bmod 2 = 1$

because 1 is the remainder when 5 is divided by 2. When we divide a number by 2, it is either even and divides perfectly, returning a 0 as the remainder, or it is odd, and returns 1 as the remainder. The mod 2 operation only returns the binary values of 0 and 1, making it useful in computer science.

Going back to the three sums, the solutions when the mod 2 operation is performed on them are:

1. $(0 + 0) \bmod 2 = 0 \bmod 2 = 0$
2. $(0 + 1) \bmod 2 = 1 \bmod 2 = 1$
3. $(1 + 1) \bmod 2 = 2 \bmod 2 = 0$

This corresponds to performing exclusive disjunction; two same values in the plaintext and key produce a 0 in the ciphertext, but two different values produce a 1.

To better understand this, consider a message 101100 that we will encrypt using the key 011010. Table 1 demonstrates the encryption; I have shown the calculations for the ciphertext for two characters. Note that in computer science, the mod 2 addition operation is often represented using the symbol $\oplus$.

Plaintext	1	1	0	1	0	0
Key	0	1	1	0	1	0
Ciphertext	$1 \oplus 0 = 1$	$1 \oplus 1 = 0$	1	1	1	0

Table 1 : Encrypting 101100

4.1.1. Need for Pseudo-randomness for Vernam Ciphers:

Now, I will investigate why the randomness of its key makes the Vernam cipher "perfect" to explore why the generation of pseudo-random numbers is important. A random key is unpredictable; it contains an equal number of 0s and 1s so the probability of either number being a key character is $\frac{1}{2}$. A random key is important because it results in the ciphertext also being random and difficult to decrypt. I will now prove that the ciphertext is random even when the plaintext is not, as long as the key is random.

Let the probability of a key character being 0 or 1 be equal to $\frac{1}{2}$. Since the probability of a plaintext character being 0 or 1 does not have to be equal, let the probability of 0 be x and the probability of 1 therefore be $1 - x$. Calculating the probability of the ciphertext characters by multiplying the key and plaintext results in the values in Table 2:

Plaintext		Key		Ciphertext	
Character	Probability	Character	Probability	Character	Probability
0	x	0	$\frac{1}{2}$	0	$\frac{1}{2}x$
0	x	1	$\frac{1}{2}$	1	$\frac{1}{2}x$
1	$1 - x$	0	$\frac{1}{2}$	1	$\frac{1}{2}(1 - x)$
1	$1 - x$	1	$\frac{1}{2}$	0	$\frac{1}{2}(1 - x)$

Table 2: Proof for Randomness of Ciphertext

We can calculate that the probability of a ciphertext character being 1 or 0 is equal to:

$$\frac{1}{2}x + \frac{1}{2}(1 - x) = \frac{1}{2}$$

Since the probabilities are equal, the ciphertext looks like a random sequence, which proves that a random key generates a random ciphertext ("One-Time Pad or Vernam Cipher").

Although the Vernam cipher has perfect secrecy, it is impractical to use because it requires a key that is as long as the message, which makes it difficult to communicate securely. One solution is to transmit a formula to the key, such as using a linear feedback shift register (LFSR), rather than transmitting the key itself.

4.2. Linear Feedback Shift Registers:

An LFSR is a recursive sequence where the preceding terms are not raised to powers and where there are no added constants (Suzuki, "Linear Feedback Shift Registers, Part One"). The sequence

$s_j = s_{j-1} + s_{j-2}$ where $s_0 = 1$ and $s_1 = 1$

is an LFSR. However, the sequence

$s_j = (s_{j-1})^2 + 5$ where $s_0 = 1$

is not an LFSR, because s_{j-1} is squared and because of the added constant 5. The formula of a recursive sequence that defines an LFSR is:

$$s_j = \sum_{i=1}^{L} c_i s_{j-i} = c_1 s_{j-1} \oplus c_2 s_{j-2} \oplus \ldots \oplus c_{L+1} s_{j-L+1} \oplus c_L s_{j-L}$$

Where $j \geq L$, L is the order of the sequence, $\oplus$ is the XOR operation, c is the connection coefficient, $c \in \{0, 1\}$, $s \in \{0, 1\}$, and c_1 to c_L and s_0 to s_{L-1} are defined explicitly (Hell, "14.2 LFSR").

To better understand this, consider the example,

$s_j = c_1 s_{j-1} \oplus c_2 s_{j-2} \oplus c_3 s_{j-3}$ where $c_1 = 0,\ c_2 = 1,\ c_3 = 1$ and $s_0 = 1, s_1 = 0,\ s_2 = 1$

In this example, the term s_j is defined by three preceding terms, therefore $L = 3$, and the LFSR is of the 3rd order. The terms c_1, c_2, c_3 and s_0, s_1, s_2 are initialized so that terms s_3 and higher can be calculated:

$s_3 = c_1 s_{3-1} \oplus c_2 s_{3-2} \oplus c_3 s_{3-3}$

$= c_1 s_2 \oplus c_2 s_1 \oplus c_3 s_0$

$= 0 \oplus 0 \oplus 1$

$= 1$

$s_4 = c_1 s_{4-1} \oplus c_2 s_{4-2} \oplus c_3 s_{4-3}$

$= c_1 s_3 \oplus c_2 s_2 \oplus c_3 s_1$

$= 0 \oplus 1 \oplus 0$

$= 1$

$s_5 = c_1 s_{5-1} \oplus c_2 s_{5-2} \oplus c_3 s_{5-3}$

$= c_1 s_4 \oplus c_2 s_3 \oplus c_3 s_2$

$= 0 \oplus 1 \oplus 1$

$= 0$

And so on. Notice that coefficients are manipulated in a way that if c is 0, it does not affect s_j, whereas if it is 1, it affects s_j. In the example, the coefficient c_1 was initialized to 0, therefore while calculating term s_j, s_{j-1} does not impact its value. For instance, the value of s_3 was affected by the value of s_0 and s_1, and not s_2. Note that initialized values for c and s_j cannot all be set to 0 as this would result in all subsequent terms being 0.

Figure 1 is a diagram representing the LFSR device:

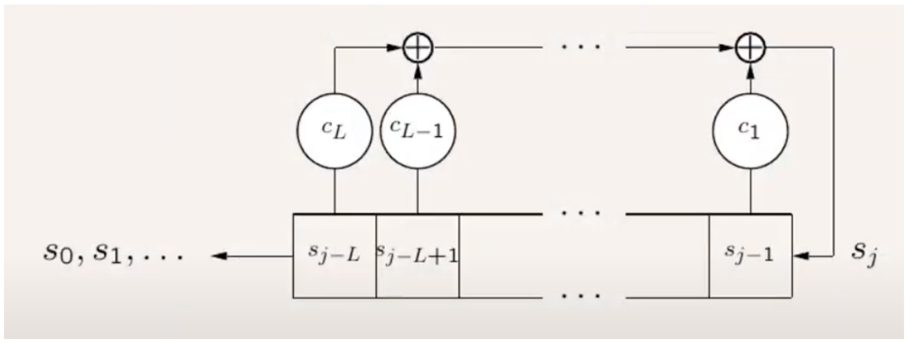
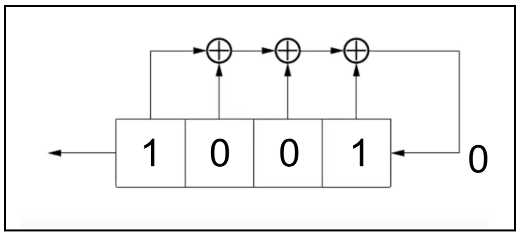


Figure 1: LFSR Diagram (Hell, "14.2 LFSR")

In computer science, the order of an LFSR is referred to as the number of registers, which contain terms s_{j-1} to s_{j-L}. LFSRs are "clock controlled"; an output is produced when the device is clocked and the bits in the registers shift once to the left. For instance, at the first clocking, the output is s_0, from s_{j-L}. The example in Figure 2 and Figure 3 help to visualize this:



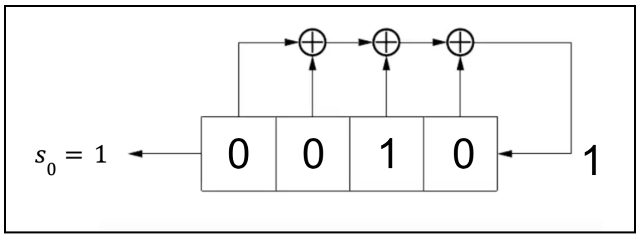


Figure 2: Initial State of LFSR

Figure 3: LFSR State After 1st Clocking

In this example, the sequence is:

$s_j = s_{j-1} \oplus s_{j-2} \oplus s_{j-3} \oplus s_{j-4}$

As shown in Figure 2,

$s_0 = 1$

$s_1 = 0$

$s_2 = 0$

$s_3 = 1$

Calculating s_4 results in 0. When the LFSR device is clocked for the first time, the values shift to the left, s_0 is outputted, and s_4 is inputted in the rightmost register.

The registers that affect the output (have a c of 1) are called taps. Taps are expressed in the form of an OGF, which is called a connection polynomial in the context of LFSRs. A connection polynomial is defined as:

$$F(x) = \sum_{i=0}^{L} c_i x^i = c_0 \oplus c_1 x \oplus c_2 x^2 \oplus \dots \oplus c_{L-1} x^{L+1} \oplus c_L x^L, \qquad \text{where } c_0 = c_L = 1$$

Where c represents connection coefficients, and x is a placeholder for the taps (Cherowitzo).

For example, the connection polynomial for the LFSR

$$s_j = s_{j-1} \oplus s_{j-4}$$

is written as:

$$F(x) = 1 \oplus x^1 \oplus x^4$$

Note that the "1" in the polynomial corresponds not to a tap, but to the input s_j, which is x^0 or 1 in the summation.

While investigating connection polynomials of LFSRs, I realized that it is similar to finding the generating function of the Fibonacci sequence (pg. 7 to 10). The general form of the Fibonacci is:

$$a_n = a_{n-1} + a_{n-2}$$

or

$$a_n - a_{n-1} - a_{n-2} = 0$$

And its generating function is expressed as:

$$A(x) = \frac{x}{1 - x - x^2}$$

The powers of x in the denominator correspond to the terms in the general form of the Fibonacci: a_n corresponds to 1, $-\ a_{n-1}$ corresponds to $-\ x$, and $-\ a_{n-2}$ corresponds to $-\ x^2$. This is similar to finding the connection polynomial of an LFSR, for instance, we calculated the connection polynomial for $s_j = s_{j-1} \oplus s_{j-4}$ as $1 \oplus x^1 \oplus x^4$. This was an interesting pattern to discover, and perhaps could be a shortcut to finding OGFs of recursive sequences like the Fibonacci, although it is unclear how the x in the numerator of $A(x)$ could be calculated this way.

4.2.1. Primitive Polynomials:

Expressing an LFSR as a connection polynomial has several uses. Although any polynomial can define an LFSR, if a polynomial is primitive, the LFSR produced has good statistical properties ("One-Time Pad or Vernam Cipher"). A primitive polynomial is irreducible, which means that it cannot be factored into other polynomials. It is also a polynomial that generates a sequence with a maximum period of $2^n - 1$, where n is the order of the polynomial (Cherowitzo). The period of a sequence is the number of terms that the sequence contains before it repeats. In the example in pg. 23,

$s_j = c_1 s_{j-1} \oplus c_2 s_{j-2} \oplus c_3 s_{j-3}$ where $c_1 = 0,\ c_2 = 1,\ c_3 = 1$ and $s_0 = 1, s_1 = 0,\ s_2 = 1$

Calculating the whole sequence produces the output:

$1,\ 0,\ 1,\ 1,\ 1,\ 0,\ 0,\ 1,\ 0,\ 1, 1,\ 1,\ 0,\ 0,\ \dots$

Observe that after the 7th term, the sequence begins to repeat itself. The sequence can also be represented as $[1,\ 0,\ 1,\ 1,\ 1,\ 0,\ 0]^{\infty}$ because it repeats infinitely. Since the sequence repeats 7 terms, the period of the sequence is 7. Since the taps are at c_2 and c_3, the connection polynomial of the sequence is:

$F(x) = 1 \oplus x^2 \oplus x^3$

$F(x)$ is a primitive polynomial because it produces a sequence with a period of $2^n - 1$:

$n = 3$

$2^3 - 1 = 7$

A period of $2^n - 1$ is the maximum period because there are 2 bit possibilities (0 and 1) for each register, and to calculate all the possible combinations in an LFSR state we have to multiply 2 by itself n times. 1 is subtracted from this because of the combination consisting of all zeroes, which produces a null sequence. Thus, if a polynomial is irreducible and produces a maximal sequence where all $2^n - 1$ possibilities occur, it is a primitive polynomial (Cherowitzo).

To check for irreducibility, we have to check two properties of the polynomial. Firstly, an irreducible polynomial has an odd number of terms (Kleitman). For instance, the following polynomials are irreducible:

$P(x) = 1 \oplus x \oplus x^2$

$P(x) = 1 \oplus x \oplus x^2 \oplus x^3 \oplus x^5$

This is because if a polynomial has an even number of terms, setting x as 1 results in 0. For instance, if we have the polynomial:

$F(x) = 1 \oplus x \oplus x^2 \oplus x^4$

Setting x as 1 results in:

$F(x) = 1 \oplus 1 \oplus 1 \oplus 1$

$F(x) = 0 \oplus 0$

$F(x) = 0$

Therefore, we know that $x = 1$ is a root of $F(x)$, which means that a factor of $F(x)$ is:

$(x - 1)$ mod 2

Thus, a primitive polynomial must have an odd number of terms as a polynomial with an even number of terms can be factorized.

Secondly, no polynomial whose terms all have even powers is irreducible (Kleitman). This is because such a polynomial can be written as the square of the polynomial whose powers are half of its powers. For example, $1 \oplus x^2 \oplus x^4$ can be written as $(1 \oplus x \oplus x^2)^2$.

$(1 \oplus x \oplus x^2)^2$

$= (1 \oplus x \oplus x^2)(1 \oplus x \oplus x^2)$

$= 1 \oplus x \oplus x^2 \oplus x \oplus x^2 \oplus x^3 \oplus x^2 \oplus x^3 \oplus x^4$

$= 1 \oplus 2x \oplus 2x^2 \oplus x^2 \oplus 2x^3 \oplus x^4$

The terms $2x,\ 2x^2$ and $2x^3$ cancel out during expansion due to the XOR operation:

$(1 \oplus x \oplus x^2)^2 = 1 \oplus x^2 \oplus x^4$

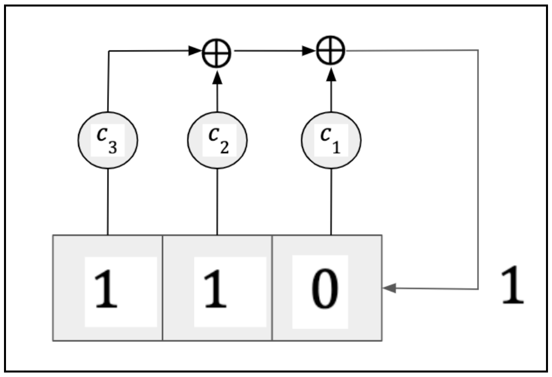
4.3. Cracking an LFSR:

Although it may be useful that the periodicity of LFSRs can be very high because it is exponentially related to order, this property makes them weak and decryptable. One way to decrypt an LFSR is using systems of equations. When analyzing the strength of cryptosystems, we have to provide every possible advantage to the hacker to ensure it is a good system (Suzuki, "Linear Feedback Shift Registers, Part Two"). Thus, while demonstrating decryption, I will decrypt knowing $2L$ terms of the sequence and the number of registers. We only need $2L$ terms of a sequence, where L is the order of the sequence, to find its connection polynomial (Suzuki, "Linear Feedback Shift Registers, Part Two"). This is because with $2L$ terms, we can create L equations, which is enough to find L number of coefficients.

I will explore decryption using the following sequence whose LFSR has 3 registers:

$s = 1, 1, 0, 1, 0, 0, \ldots$

To create the system of equations, we have to write every possible state of this LFSR. For example, Figure 4 and Figure 5 represent the first two states of the LFSR device that produces s:



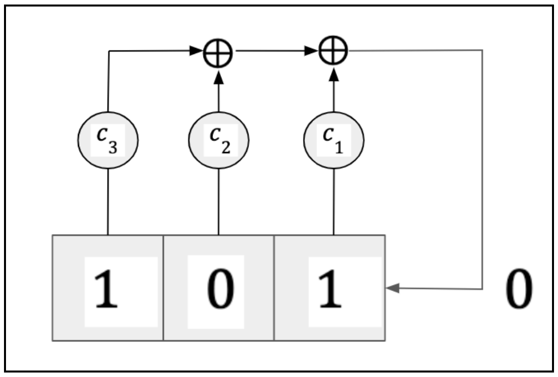


Figure 4: Initial State of LFSR

Figure 5: Second State of LFSR

All the states that we can extract from the sequence $s = 1, 1, 0, 1, 0, 0, \ldots$ are listed in Table 4:

j	s_{j-3}	s_{j-2}	s_{j-1}	s_j
3	1	1	0	1
4	1	0	1	0
5	0	1	0	0

Table 4: LFSR States

Recall that the formula of a sequence that defines an LFSR is:

$$s_j = \sum_{i=1}^{L} c_i s_{j-i} = c_1 s_{j-1} \oplus c_2 s_{j-2} \oplus \ldots \oplus c_{L+1} s_{j-L+1} \oplus c_L s_{j-L}, \ j \geq L, \ c \in \{0, 1\}, \ s \in \{0, 1\}$$

Using this formula helps generate our system of equations. We can write our first equation as:

$s_j = c_1 s_{j-1} \oplus c_2 s_{j-2} \oplus c_3 s_{j-3}$

$s_3 = c_1 s_2 \oplus c_2 s_1 \oplus c_3 s_0$

We can now replace the values for s_0, s_1, s_2 and s_3 from the first row where $j = 3$ into the equation:

$s_0 = s_{3-3} = 1$

$s_1 = s_{3-2} = 1$

$s_2 = s_{3-1} = 0$

$s_3 = 1$

$1 = 0c_1 \oplus 1c_2 \oplus 1c_3$

Therefore, the first equation is:

$1 = c_2 \oplus c_3$

The second equation is:

$s_4 = c_1 s_3 \oplus c_2 s_2 \oplus c_3 s_1$

$0 = 1c_1 \oplus 0c_2 \oplus 1c_3$

$0 = c_1 \oplus c_3$

The third equation is:

$s_5 = c_1 s_4 \oplus c_2 s_3 \oplus c_3 s_2$

$0 = 0c_1 \oplus 1c_2 \oplus 0c_3$

$0 = c_2$

Thus, the system of equations is:

$1 = c_2 \oplus c_3$

$0 = c_1 \oplus c_3$

$0 = c_2$

From this we have the value for c_2:

$0 = c_2$

We also know that:

$1 = c_2 \oplus c_3$

Which means that c_3 has to be 1. Knowing that:

$0 = c_1 \oplus c_3$,

We can also calculate c_1 to be 1.

We now know that the taps of the LFSR are at c_1 and c_3. Therefore, we can write the LFSR connection polynomial as:

$F(x) = 1 \oplus x \oplus x^3$

4.4. Evaluation of LFSRs:

Recursive sequences and OGFs are useful in the foundation of LFSRs. Using generating functions (connection polynomials) in LFSRs allows us to make use of properties such as primitivity. Primitive polynomials are ideal for LFSRs as they can create maximal sequences with the least number of registers. However, linearity is the "the curse of cryptographer" because properties such as periodicity hinders their security ("One-Time Pad or Vernam Cipher"). Another such property is that the sequence is not pseudo-random because the probability of a 0 or 1 occurring in the LFSR is not exactly $\frac{1}{2}$. Since an all 0 state is not possible for an LFSR, the number of 1s in a sequence is more than the number of 0s.

Since an LFSR alone does not provide much secrecy, mixing different LFSRs could result in greater security. For instance, in paired generators two or more LFSR sequences undergo mod 2 addition to produce a new sequence with a higher period. For example, if we have sequences

$P = 1, 0, 1, 1, 0, 1, 0$

And

$Q = 0, 1, 0, 1, 1, 0, 0$

We can add them mod 2 to get R, as shown in Table 5:

P	1	0	1	1	0	1	0

Q	0	1	0	1	1	1	0
R	1	1	1	0	1	0	0

Table 5: Paired Generator Producing a Pseudo-Random Sequence

Examples of other such generators include shrinking generators and alternating step generators (Suzuki, "Linear Feedback Shift Registers, Part Three").

5. Conclusion

To conclude, in this essay, I have explored how OGFs help find the closed form of recursive sequences, and how they apply to cryptography. To make the concept of OGFs coherent, I first used an example involving the number of ways to pay a certain amount of money. This example illustrates how OGFs are useful as they allow us to algebraically manipulate an infinite sequence of numbers.

Next, I demonstrated how OGFs can be used to find the closed form of a recursive sequence using the Fibonacci sequence. This was a fascinating example to investigate because I previously thought that finding a closed form can only be possible with sequences that have a common ratio or difference, but this process taught me that OGFs make it possible to even generalize recursive sequences that get infinitely large. It was also interesting to come across the mathematical number, $\frac{1+\sqrt{5}}{2}$, also known as the Golden Ratio, as I was solving for the roots of the quadratic equation in the denominator of the generalized Fibonacci sequence. Recalling that the golden ratio is the ratio between two successive Fibonacci terms, I realized that its appearance in the closed form of the Fibonacci is similar to how the common ratio or difference of an arithmetic or geometric sequence appears in its closed form. Although it might divert from the focus of the essay, it would have been interesting to elaborate on this by deriving a proof for the Golden Ratio.

Next, I explored the use of recursive sequences and OGFs in cryptography. I explored how LFSRs generate pseudo-random numbers for keys of ciphers, decrypting them, and ways to strengthen them. I was drawn to discussing pseudo-randomness as I came across this concept when I searched up why the “shuffle” feature on my Spotify, which is supposed to randomize the music that is played, seemed to play a pattern of the same songs rather than randomized ones. Thus, it was intriguing to explore the math behind this concept as it not only has applications in cryptography, but also appears in many parts of my daily life, from the music I listen to, to the video games I play. For further investigations, I would explore how primitivity in polynomials can be checked. There is a method for decrypting LFSRs involving the Euclidean algorithm (Hell, “14.3 LFSR”), which would also have made for an interesting read if I had more words.

All in all, I believe that I have substantially explored both parts of my research question; I have thoroughly investigated the relationship between OGFs and recursive sequences, as well as the extent of their use in cryptography.

6. Works Cited

Cherowitzo, William. “Linear Feedback Shift Registers.” *University of Colorado Denver*, http://www-math.ucdenver.edu/~wcherowi/courses/m5410/m5410fsr.html. Accessed 5 Dec. 2022.

Dobrushkin, Vladimir. “Part V: Generating Functions.” *Brown University*, https://www.cfm.brown.edu/people/dobrush/am33/Mathematica/ch5/gfunction.html. Accessed 11 Sep. 2022.

“Generating Functions.” 15-251: Great Theoretical Ideas in Computer Science, 18 Sept. 2008, *Carnegie Mellon University*, Lecture handout. https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15251-s09/Site/Materials/Handouts/generatingfunctions.pdf. Accessed 9 Sep. 2022.

“Guide to Cryptography Mathematics.” *Privacy Canada*, 5 July 2022, https://privacycanada.net/mathematics/. Accessed 25 Nov. 2022.

Hell, Martin. “14.2 LFSR - Connection Polynomial and Recursion.” Youtube, 27 Aug. 2020, https://www.youtube.com/watch?v=dwjZbi2QxTk. Accessed 18 Nov. 2022.

---. “14.3 LFSR Theorem and Euclid's Algorithm for LFSRs.” Youtube, 27 Aug. 2020, https://www.youtube.com/watch?v=Htj6-PQ1lMI. Accessed 29 Nov. 2022.

Kleitman, Daniel. “10. Polynomial Codes and Some Lore about Polynomials.” *MIT Mathematics*, https://math.mit.edu/~djk/18.310/18.310F04/polynomial_hamming_codes.html. Accessed 5 Dec. 2022.

"One-Time Pad or Vernam Cipher." *Root Me*, 12 Nov. 2014, https://repository.root-me.org/Cryptographie/Sym%C3%A9trique/EN%20-%20One-Time%20Pad%20or%20Vernam%20cipher.pdf. Accessed 11 Dec. 2022.

Rochford, Austin. "Generating Functions and the Fibonacci Numbers." Austin Rochford, 1 Nov. 2013, https://austinrochford.com/posts/2013-11-01-generating-functions-and-fibonacci-numbers.html. Accessed 11 Sep. 2022.

Suzuki, Jeff. "Linear Feedback Shift Registers, Part One." *Youtube*, 11 Sept. 2019, https://www.youtube.com/watch?v=iqGKAu5__lE&t=2s. Accessed 17 Nov. 2022.

---. "Linear Feedback Shift Registers, Part Three." *Youtube*, 23 Mar. 2020, https://www.youtube.com/watch?v=wr5Dg7vRGgk. Accessed 17 Nov. 2022.

---. "Linear Feedback Shift Registers, Part Two." *Youtube*, 12 Sept. 2019,https://www.youtube.com/watch?v=i8WQlBDJTbc. Accessed 17 Nov. 2022.

---. "Vernam Cipher." Youtube, 7 July 2019, https://www.youtube.com/watch?v=2R2AYeieoRw. Accessed 17 Nov. 2022.

Weisstein, Eric W. "Recursive Sequence." *Wolfram MathWorld,* https://mathworld.wolfram.com/RecursiveSequence.html. Accessed 9 Sep. 2022.

Zaczyński, Bartosz. “Bitwise Operators in Python.” *Real Python*, https://realpython.com/python-bitwise-operators/. Accessed 20 Nov. 2022.