Cryptography: Course Project

Instructed by *Wenfei Wu*

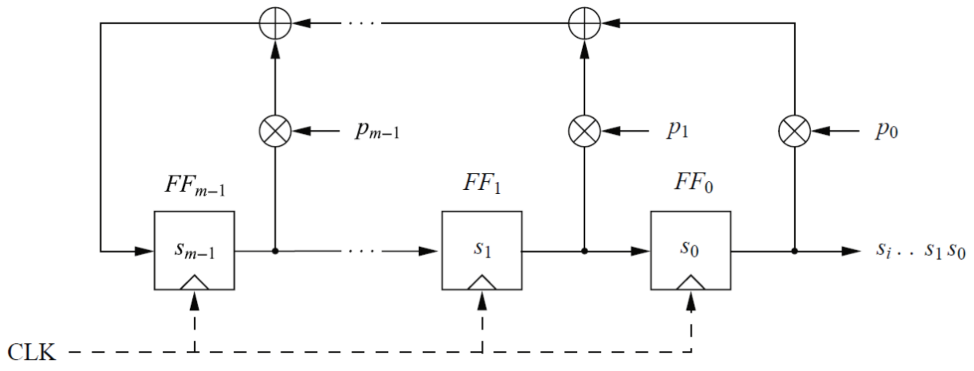
Due on January 10, 2020

Wenjun Yu & Yiheng Shen YaoClass 70 2017011072 & 2017010340

Introduction

1 Background

Our project mainly focus on the LFSR theme. LFSR is the abbreviation of linear-feedback shift register. In computing, a linear-feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state. the structure of an LFSR is as the following:



Each time the LFSR calculates the outcome of 1 bit, it right-shift all the rest bits once and use a linear function on the output bit and the previous bits to calculate the blank bit generated by the right-shifting in the new round. Usually, The most commonly used linear function of single bits is exclusive-or (XOR), as is shown in the previous graph. Thus, an LFSR is most often a shift register whose input bits are driven by the XOR of some bits of the overall shift register value.

The initial value of the LFSR is called the seed, and because the operation of the register is deterministic, the stream of values produced by the register is completely determined by its current (or previous) state. Likewise, because the register has a finite number of possible states, it must eventually enter a repeating cycle. However, an LFSR with a well-chosen feedback function can produce a sequence of bits that appears random and has a very long cycle.

Applications of LFSRs include generating pseudo-random numbers (usually is for stream ciphers), pseudo-noise sequences, fast digital counters, and whitening sequences. Both hardware and software implementations of LFSRs are common.

The mathematics of a cyclic redundancy check, used to provide a quick check against transmission errors, are closely related to those of an LFSR.

A classical LFSR string generator (as shown in the figure) with m registers could be represented by the following equation:

$$s_{i+m} = \sum_{j=0}^{m-1} p_j s_{i+j} \bmod 2.$$

We could use the characteristic polynomial of the recursion equation above to denote this LFSR as:

$$P(x) = x^m + \sum_{i=0}^{m-1} p_i x^i.$$

2 Related Work

We have reviewed the work of Tian Ye and the work of Runzhou Tao, Kaifeng Lv. In Tao's work they made a study about polynomial libraries that are widely used in practice and made a survey on the security of other stream ciphers, including A5/1 and RC4. Basically, Tao et al.'s work more focus on the application layer.

Ye's work is very similar to what we are going to do (Actually, he is required to solve exact the same two problems as ours). However, we found that some of the proofs in his report is written sloppily or missing (e.g. Corollary 2) and not correctly (e.g. Theorem 5). Moreover, the algorithm to find irreducible polynomials in his report turns out to be wrong. We discussed the algorithms with Ye again and he admitted the mistakes. In this report, we fixed the errors in his report and we reorganized the structure of the solution to these problems in order to make it clearer for the readers.

3 Our contributions

Since the construction of the LFSR is closely realted to irreducible polynomials, our report consists of two parts. The first part of our report solves two problems about irreducible polynomials, one is to find the relation between irreducible polynomials and primitive polynomials; the second is to work out an algorithm to find out all irreducible polynomials with degree n. In the second part, we read a paper on recent work of LFSR and stream cipher. We extract the main ideas and contributions from this paper and we gave some comments on it.

Problem 1

First we give a formalized definition of the *primitive*:

Definition 1. *A polynomial $f(x) = x^n + \sum_{i=0}^{n-1} p_i x^i$ over $\mathbb{F}_2$ is called primitive iff for all nonzero bit-strings $A = \{a_i\}_{i\geq 0}$ satisfying a recursion relationship that $a_{i+n} = \sum_{j=0}^{n-1} p_j a_{i+j}$ the period will be $2^n - 1$.*

Note that this kind of definition just explain what it said in the material more expolicitly, we can just use this to show our result. For the first direction, we have the following proof.

Proof. Suppose there is a reducible polynomial $f(x) = s(x)t(x)$, where $s(x)$ and $t(x)$ are both polynomials with degree larger than 1. Note in the definition, it provides a recursion equation and we can use it here to represent the reducibility. Assume the character polynomial is $f(x) = x^n + \sum_{i=0}^{n-1} p_i x^i$, we have the equation

$$\sum_{i=0}^{n} p_i a_{i+\Delta} = 0, \ \forall\, \Delta \in \mathbb{Z}.$$

Problem 1 continued on next page...

W.L.O.G, assume $s(x) = x^m + \sum_{i=0}^{m-1} l_i x^i$ and $t(x) = x^k + \sum_{i=0}^{k-1} r_i x^i$ and of course, $k + m = n$. Then it's obvious that let $b_j = \sum_{i=0}^{m} l_i a_{i+j}$ and for this sequence it always holds $\sum_{i=0}^{k} r_i b_{i+j} = 0$. According to the pigeonhole principle, since the total number of configurations of b is at most $2^k - 1$, so is the period, we denote it by T_B. Moreover, we can define $c_i = a_{i+T_B} - a_i$, then $\sum_{i=0}^{m} l_i c_{i+j} = b_{j+T_B} - b_j = 0$. And we can then follow the similar idea prove that the period of C is at most $2^m - 1$ and denotes by T_C.

Claim 1. *With two sequences defined above, the period T_A of A satisfies*

$$T_A | 2T_B T_C.$$

Proof. For an arbitrary integer i, find a set of $2T_C + 1$ corresponding $\{a_{i+lT_B}\}_{l=0}^{2T_C}$. Then each a_{i+lT_B} minus the next one, and we get the set of $2T_C$ $\{c_{i+sT_B}\}_{s=0}^{2T_C-1}$. Note that the latter half of these c is just equal to the corresponding one in the first half. The sum of these $\{c\}$ is exact 0 in $\mathbb{F}_2$. Therefore, represent these c by the differences a, we get $a_{i+2T_C T_B} - a_i = 0 \; \forall\, i \in \mathbb{Z}$. □

Thus if $T_A = 2T_B T_C$, we know that $T_A < 2^n - 1$ since parities keep different. If $T_A < 2T_B T_C$, then $T_A \le T_B T_C \le (2^m - 1)(2^k - 1) < 2^{m+k} - 1$. □

For the other direction, we can use an example to disprove it.

Proof. Consider the polynomial $f(x) = x^8 + x^4 + x^3 + x + 1$ over $\mathbb{F}_2$, and it's no doubt an irreducible polynomial and moreover it is a factor of polynomial $p(x) = x^{51} - 1$. Then we need to show this is not a primitive one with our version of primitive.
To understand it, we can regard the notation x^i as an operator over a bit-string A given a index j, which returns A_{i+j}. Thus A with character polynomial $f(x)$ means for all j, if we implement the linear combination of operators, $f(x)$, it will return 0. Note $p(x) = t(x)f(x)$ for some non-trivial $t(x)$, so if we implement $p(x)$ to the string with an arbitrary index, it always returns 0, which means for all index i $a_i = a_{i+51}$. And it's obvious $f(x)$ is not primitive under our definition. □

Remark 1. In order to check the proposed polynomial $f(x)$ is indeed a factor of $p(x)$, we use a computer program to find the remainder of this division in $\mathbb{Z}[x]$ and it is

$$r(x) = 504x^7 + 380x^6 - 64x^5 - 416x^4 - 362x^3 - 404x^2 - 692x - 386.$$

Note this $r(x)$ is just equal to 0 in $\mathbb{F}_2[x]$.

Definition 2. *A polynomial $f(x) = x^n + \sum_{i=0}^{n-1} p_i x^i$ over $\mathbb{F}_2$ is called primitive iff there exists one root α s.t., $\forall\, 0 < k < 2^n - 1$ $\alpha^k \neq 1$.*

Claim 2. *A polynomial over $\mathbb{F}_2$ is primitive in definition 1 iff it is primitive in definition 2.*

In the proof part of the problem 2, we use the classical definition of the primitive poly (Definition 2). We will prove that these two definitions are equivalent in appendix.

An algorithm for Polynomial Factorization

In this section, we give a polynomial algorithm which could factorize any polynomial $f(x) \in \mathbb{F}_2(x)$ into irreducible polynomials. The algorithm is as follows:

1 If $f'(x) = 0$, we know that $f(x)$ has only the even power terms, them we could denote $f(x)$ as $g(x^2)$, then we get $g(x)^2 = f(x)$, since the cross-terms of $g(x)^2$ has even coefficients. We repeat this until we get $f'(x) \neq 0$.

An algorithm for Polynomial Factorization continued on next page. . .

2 If $f(x)$ has square terms, we get $(f(x)', f(x)) \neq 1$. We use the Euclidean algorithm to calculate $h(x) = (f(x)', f(x))$ and divide $f(x)$ by $h(x)$. We repeat this step until $f(x)$ has no repeated roots.

3 Suppose after the previous two steps, the polynomial $f(x)$ has been tured into $v(x)$. We then use the Berlekamp's algorithm to factorize $v(x)$ into irreducible factors. The details would be shown in the presentation.

Problem 2

In this problem, we first give a theorem to verify whether an irreducible poly is primitive, then we try to find out all the irreducible polynomials with degree n.

Theorem 1. *If Given an irreducible polynomial $f(x)$ with degree n, it is primitive polynomial if and only if there is no such prime factor of $2^n - 1$, denoted by p, that $f(x)|x^{(2^n-1)/p} - 1$.*

Proof. By the definition of primitive polynomials, we know that if $f(x)$ is primitive, then there should be no solution for k less than $2^n - 1$ for the equation $f(x)|x^k - 1$. On the other side, if there is no such p, we first prove that there is no solution for this equation less than $2^n - 1$. If there is, namely k', we got $f(x)|x^{k'} - 1$, then $f(x)|x^{k'} - 1, x^{2^n-1} - 1$, $f(x)|x^{(k',2^n-1)} - 1$, so we get $(k', 2^n - 1) \neq 1$ and made a contradiction. Then we prove that if there is no solution with less than $2^n - 1$, $f(x)$ is primitive. If $f(x)$ is not primitive, there should be a root α with $\alpha^{k'} = 1 (k' < 2^n - 1)$. α satisfies $\alpha^{k'} - 1 = 0$. Since $f(x)$ is the minimal poly for the root α (since it is irreducible), we know that $f(x)|x^{k'} - 1$ and it leads to a contradiction, so we have proven the theorem. □

Then our algorithm to find out all the irreducible polys is as follows.

1 The first step is to generate n different polynomials of degree n. And we use the polynomial factorization algorithm and the previous theorem to test whether they are primitive poly or not, and if we find an primitive poly, we stop the generation. We denote this polynomial as $f(x)$.

2 We let α be a root of $f(x)$. We let $\{p_1, p_2, \cdots, p_t\}$ be the set of prime factors of $2^n - 1$. We let the set S be the set as follows: $S = \left\{x \in \{1, 2, \cdots, 2^n - 1\} \,|x \nmid 2^{n/p_i} - 1, \forall i \in \{1, 2, \cdots, t\}\right\}$

3 For each $i \in S$, we compute the minimal poly of the element α^i. Then we could get all the irreducible polynomials of degree n.

We first prove the correctness of the algorithm, that is the following theorem:

Theorem 2. *The algorithm could success with probability at least $1 - 1/e$.*

Proof. More precisely, we prove that the first step would succeed with probability at least $1 - 1/e$ and if the first step succeeds, then the algorithm would work.

First we know that the roots of the primitive elements lies in the primitive elements of the filed $\mathbb{F}_{2^n}$, that is, suppose that a primitive of this field is g, then we know that g^w, where $(w, 2^n - 1) = 1$ should also be the primitives. So the number of all the primitive poly roots is $\phi(2^n - 1)$. Since primitive polys are irreducible polys, their roots do not overlap. A primitive poly with degree n should have exactly n roots. Therefore the total number of primitives with degree n is $\phi(2^n - 1)/n$. By the lower bound of $\phi(x)/x$ is $\frac{1}{\log \log x}$, we know that the ratio of primitive polys is larger than $1/(n \log \log(2^n - 1)) > 1/(n \log n)$. If we pick $n \log n$ polys randomly, the probability to have no primitive poly among them is less than $(1 - 1/n \log n)^{n \log n} < 1/e$.

So we could guarantee that the first step would succeed with prob at least $1 - 1/e$. Then it comes to the correctness of the next two steps. We first prove that all the roots of irreducible polys are a power of a

Problem 2 continued on next page...

primitive of $\mathbb{F}_{2^n}$.

We suppose that the splitting field of $x^{2^n} - x$ is $\mathbb{L}_{2^n}$ and one of the root of this irreducible is α. By the Galois's Theorem, we get $[\mathbb{F}_2(\alpha) : \mathbb{F}_2] = n$, therefore the size of field $\mathbb{F}_2(\alpha)$ should be 2^n. Since all the field with size 2^n is unique(uniqueness of finite field), we get all the irreducible with degree n should has their roots in this field. we denote this field as $\mathbb{F}_{2^n}$ for convenience. Since all the primitive polys are irreducible and any root of a primitive would be able to generate the field $\mathbb{F}_{2^n}$, we could know that any root of a polynomial should be the a power of this primitive. And moreover, we have the relation that $f(x)|x^{2^n} - x$ for all irreducible n-degree poly $f(x)$.

Then we prove that the polynomial $x^{2^n} - x$ is the multiplication of all the irreducible polys whose degrees are the factors of n.

Since we have proved that all the irreducible polys have their roots in the field $\mathbb{F}_{\nvDash\ltimes}$. Then any element in the field $\mathbb{F}_{2^n}$ with minimal poly with degree d should satisfy the relation $d|n$. Because suppose that this element β is the k-power of a primitive α, we have $\beta^{2^d-1} = 1$ and $\beta^{2^n-1} = 1$, then $\beta^{(2^d-1,2^n-1)} = \beta^{2^{(d,n)}-1} = 1$ and we get $d \leq (d,n)$ (since the degree of minimal poly of any root of $x^{2^{(d,n)}} - x$ should be no less than (d,n). Then naturally we have $d = (d,n)$ and $d|n$. Then we simply build a bijection from the equivalent class of elements in $\mathbb{F}_{2^n}$ (with respect to the same minimal poly) to the corresponding minimal irreducible polys whose degrees are the factors of n. Since all the irreducible polys whose degrees are factors of n all have their roots in $\mathbb{F}_{2^n}$ and all the elements in this field should all be included in an equivalent class whose corresponding irreducible poly has an n-factor degree, the bijection property could be easily verified. Therefore after combining all the irreducible polys and the comparing it with $x^{2^n} - x$, we get the conclusion that the poly $x^{2^n} - x$ is the product of all the irreducible polys whose degrees are the factors of n.

Then since we have proved $x^{2^n} - x$ has included all the irreducibles with degree n and the factors of n, we only need to rule out the roots of the irreducibles whose degree is the factors of n and smaller than n. Suppose that the root of the primitive poly (which is also a primitive in $\mathbb{F}_{2^n}$) is α, and the root of any irreducible poly is α_i. We have $\forall p|n, p < n$, $(\alpha^i)^{2^p} \neq \alpha^i$ and thus $i \cdot (2^p - 1) \not\equiv 0 \pmod{2^n - 1}$. Therefore if we only enumerate i to avoid such condition, thus we have prove the correctness of our algorithm.

□

Problem 3

The paper we find for current progress of the field of stream cipher, especially linear feedback shift registers (LFSRs) is called "design and implementation of a key generator-based stream cipher for securing text data" [1]. This paper focus on a more efficient manner to encrypt and it consider how to face the man-in-the-middle (MITM) attack. However, the tranditional protocols which resist this always consider vulnerable for some typical attacks, thus the paper proposed a combination of some famous processes and form a good one.

Note that MITM attacks always show up during the asymmetric cryptography, so a nature idea is to use stream cipher which use keys twice. This property also bring some drawbacks, and the one of the worst is how to distribute keys. A famous protocol to avoid this difficulty is to employ LFSRs key generator, but obviously this is so weak in secure that due to the linearity there is a well-known algorithm, Berlekamp-Massey algorithm, which can successfully attack and gain information from this cipher. Nevertheless, we still want to implement this kind of encryption according to the attention on the efficiency. Therefore, the proposal from this paper is to combine three types of shift registers algorithms to hide the vulnerabilities.

Now let's turn to details of the design. First and foremost, we need to introduce these three different protocols. The most basic one is the LFSRs protocol. It starts with an initial vector $\{s_0, \cdots, s_{n-1}\}$ over the

Problem 3 continued on next page...

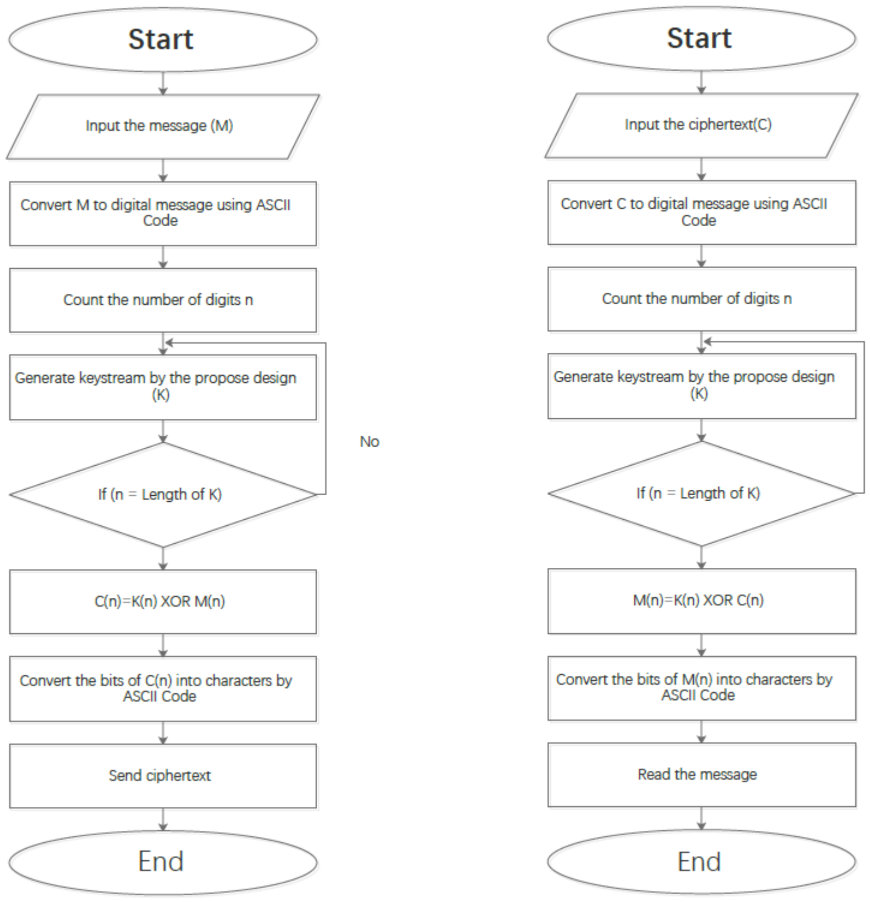

Start
Input the message (M)
Convert M to digital message using ASCII Code
Count the number of digits n
Generate keystream by the propose design (K)
If (n = Length of K)
No
C(n)=K(n) XOR M(n)
Convert the bits of C(n) into characters by ASCII Code
Send ciphertext
End
Start
Input the ciphertext(C)
Convert C to digital message using ASCII Code
Count the number of digits n
Generate keystream by the propose design (K)
If (n = Length of K)
No
M(n)=K(n) XOR C(n)
Convert the bits of M(n) into characters by ASCII Code
Read the message
End

Figure 1: Flow charts for encryption/decryption

field $\mathbb{F}_2$ and in the whole system we only care about their relative positions instead of the indices. Choose some active positions (which we call by *tap*) and we can XOR them to calculate the next s. As descibed above, the process can also be characterized by a polynomial over field $\mathbb{F}_2$ and we only regard x there as a shift-operator.

The next algorithm we introduce is non-linear FSRs (NLFSRs) [2], and we find the rough idea is similar except that NLFSRs will implement some non-linear functions when use tap to calculate the next s. This kind of non-linearity just make our cipher secure under some classical attack target linear functions. The other variation of LFSRs shown in this proposal is called the feedback with carry shift registers (FCSRs). This protocol is like LFSRs except it contains an additional memory and use a function of summation in integer space. With the taps and summation, the result is still an integer, denoted by σ. Then to update the new s and the carry bit, it calculates the parity σ mod 2 and $\lfloor \sigma/2 \rfloor$. More general, it can be transformed to be σ mod N and $\lfloor \sigma/N \rfloor$ respectively for a fixed integer N. FCSRs still keep the advantages from LFSRs like it can be implemented with high speed and relaxed conditions on hardwares. Moreover, this kind of protocol does lack of theoretic encryptioon analysis [3], which means it is impossible to attack this in a general case.

With this background knowledge about the group of protocols, it's easier to explain how we can make them together and how this combination owns some qualities against specific attacks. Thus we now specify how to construct the new protocol using these three traditional ciphers. Since we still care about the MITM attacks, we use the stream cipher and the only thing we need to modify is the psuedo random generator, and then we XOR these psuedo random number with the message in bit string manner to get the ciphertext. We can view the encryption and decryption stages by following flow charts. The generator is combined

Problem 3 continued on next page. . .

from those three famous generators. The critical idea is to use a combination function to deal with the simultaneous outputs from those independent parts. Just viewd like the following paradigm, the generator contains four LFSRs generators, one NLFSRs and one FCSRs generator. These six pieces of subprogram use independent keys (initial vectors for calculate and shift) chosen randomly. Then every generator outputs one bit (denoted by y_i respectively) and the combination function f then calculate the true result by

$$f(y_1, y_2, y_3, y_4, y_5) = y_2y_4y_5 + y_1y_2 + y_1y_5 + y_2y_5 + y_3y_5 + y_3.$$

Note some wired things happened here that we have used only the first five outputs and there are two y_5s in the generator. This is indeed another fancy design that it employs a comparator and a switch to choose different Generator Five (NLFSRs or FCSRs). This switch module calculate both these two generators for 100 bits and if the last bit is 1 it then uses the greater (for the last 100 bits) one for its next 100-bit output. The utility of this comparator is to prevent detection of the initial vectors.

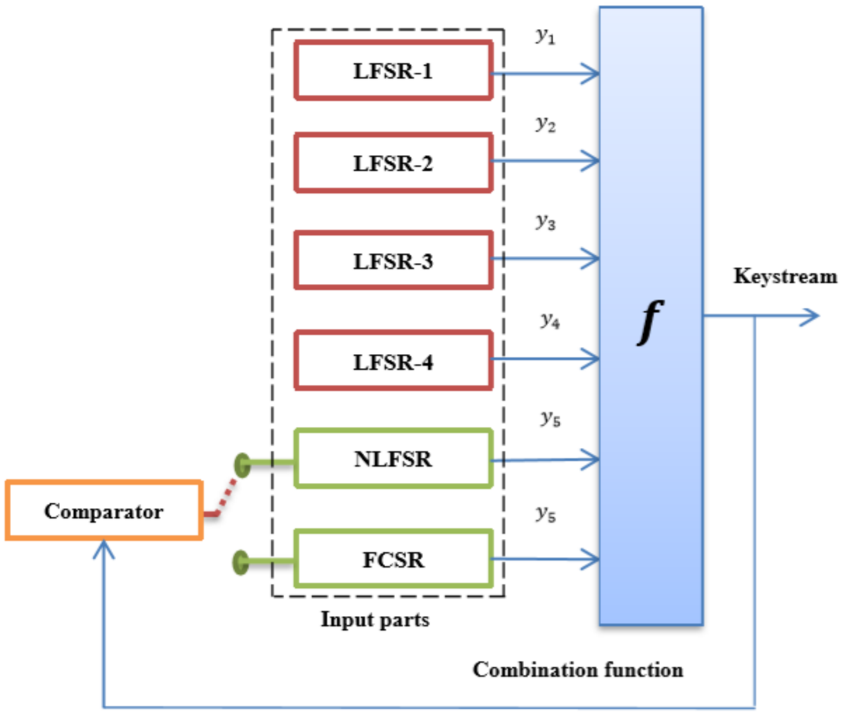
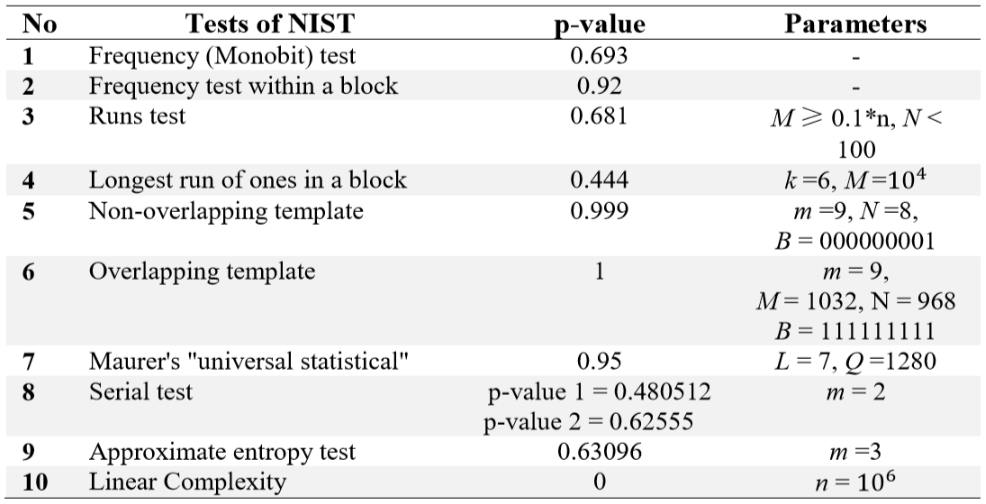


Figure 2: Structure of proposed generator

So far, we have shown the general structure of the proposed pseudo random number generator, so a natural question is how to estimate the performance of this generator. The robustness of the proposed generator was evaluated by measuring the randomness key by using the National Institute of Standards and Technology (NIST). In this system, the author used altogether eleven measurements to estimate the statistical properties. More specifically, it choose the threshold of p-value to be 0.01 to distinguish the statistical significance, but in fact you can view that these values are far more greater than this threshold except one. In the following, we will introduce some of these tests in [4].

The first test (frequency test) is to measure the proportion of 1s and 0s from the output. It can be seen that for a true random generator, this will succeed with high probability. The third one records the runs number of the output bit string, and calculates the differences between both observed numbers of 0s series and 1s series and the expected values. In addition, the fourth test records the longest runs both 0s and 1s and compares these with expected values.

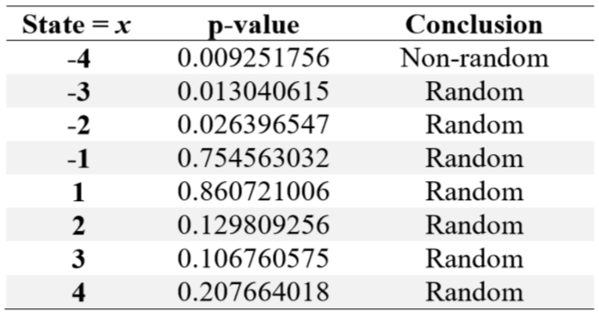
Problem 3 continued on next page. . .



No	Tests of NIST	p-value	Parameters
1	Frequency (Monobit) test	0.693	-
2	Frequency test within a block	0.92	-
3	Runs test	0.681	$M \geqslant 0.1$*n, $N <$ 100
4	Longest run of ones in a block	0.444	k =6, M =10^4
5	Non-overlapping template	0.999	m =9, N =8, B = 000000001
6	Overlapping template	1	m = 9, M = 1032, N = 968 B = 111111111
7	Maurer's "universal statistical"	0.95	L = 7, Q =1280
8	Serial test	p-value 1 = 0.480512 p-value 2 = 0.62555	m = 2
9	Approximate entropy test	0.63096	m =3
10	Linear Complexity	0	n = 10^6

Figure 3: P-value results for ten NIST tests

Another worth a chart statistical test is called random excursion test. This fancy test needs to transform the bit string to a sign string $\{x_i'\}_i$ (namely $x_i \to x_i' = (-1)^{x_i}$). Then it calculates the successive partial sums, that is $s_i = \sum_{j=1}^{i} x_j'$ and you can see this forms a new string $S = 0s_1 \cdots s_n 0$ over $\mathbb{Z}$. Then the test only cares frequence of s to be in the regimes from -4 to -1 and from 1 to 4. It will further records the number of 0s in S except the first digit. With this knowledge the test program can calculate the p-value and determine whether some issues are statistically significant or not.



State = x	p-value	Conclusion
-4	0.009251756	Non-random
-3	0.013040615	Random
-2	0.026396547	Random
-1	0.754563032	Random
1	0.860721006	Random
2	0.129809256	Random
3	0.106760575	Random
4	0.207664018	Random

Figure 4: P-value for random excursions test

With all the descriptions and results of statistical tests, we can get a fully understanding about this fancy protocol using stream cipher. This cipher does better than some widely used ones over these tests. It also resists the MITM attacks and proposed a generator which largely solved the using key twice issue. Moreover, thanks to the properties of the basic LFSRs generator, this protocol can be implemented with a relaxed condition of hardware and software. In the end of the paper [1], the author construct an APP on mobile phone to use this proposal to encrypt message, which shows its practicability.

Appendix

In the theorem 2, we have supposed that the definition of the primitive polynomials in the LFSR(defined by the period of the generated string) is the same as the classical definition of the primitive polys in abstract algebra (there exists a root whose order no less than $2^n - 1$). In this appendix we prove this.

Appendix continued on next page. . .

Proof. Since in Theorem 2, we have shown that every root of an irreducible $f(x) = x^n + \sum_{i=0}^{n-1} p_i x^i$ is in $\mathbb{L}_{2^n}$, it's straightforward to know a reducible $f(x)$ cannot be a second kind of primitive polynomial. Neither is the first kind follow the result of problem 1. Therefore, given the polynomial is reducible, these two definitions remain the same.

For the other case that given the polynomial $f(x)$ is irreducible. Note in the above we declared the splitting field of $f(x)$ is equal to $\mathbb{F}_{2^n}$ and it's easy to see $f(x)$ owns all distinct roots. Therefore, $\alpha_1, \cdots, \alpha_n \in \mathbb{F}_{2^n}$ satisfy $\alpha_i^{2^n-1} = 1 \forall\, i \in [n]$.

Consider the following matrix:

$$D = \begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \cdots & \alpha_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \cdots & \alpha_n^{n-1} \end{bmatrix}$$

It's obvious that this matrix is full-ranked. Thus every a_i with notations similar with those in problem 1 can be represented by a linear combination of $\{\alpha_j^i\}_{j=1}^n$, so the order of α (namely $2^n - 1$) is one kind of period of a, denoted by T. However, it may not be the minimal one, which is denoted by T_A. So the it suffices to prove the following claim.

Claim 3. $T = T_A$.

Proof. Suppose $T_A < T$, thus for all j, $a_{j+T_A} - a_j = 0$, which can be written as

$$a_{j+T_A} - a_j = \sum_{i=1}^{n} k_i(\alpha_i^{j+T_A} - \alpha_i^j) = \sum_{i=1}^{n} k_i(\alpha_i^{T_A} - 1)\alpha_i^j = 0.$$

Since we have shown D is full-ranked, the coefficients are all 0. And this is a contradiction of definition of T, so we are done. □

□

References

[1] BASHEER H ALI, MOHAMMED J ZAITE, and ABDULLAH S AL-HASHIMI. Design and implementation of a key generator-based stream cipher for securing text data. *Journal of Engineering Science and Technology*, 14(6):3372–3386, 2019.

[2] Elena Dubrova. A list of maximum-period nlfsrs, 2012.

[3] Mark Goresky and Andrew Klapper. *Algebraic shift register sequences.* Cambridge University Press, 2012.

[4] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, and Elaine Barker. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, Booz-Allen and Hamilton Inc Mclean Va, 2001.