The Finite Lab-Transform (FLT) for Invertible Functions in Cryptography

Peter Lablans – LabCipher

Abstract

In cryptography, algorithms frequently leverage number theory, employing operands from $\mathbb{Z}_n$ (e.g., $x \in \mathbb{Z}_7$ with elements {0, 1, 2, 3, 4, 5, 6}). This article introduces a novel transformative approach: the application of n-state inverters (n > 2) to construct novel n-state 2-operand computer operations. The novel transformation is the Finite Lab-Transform (FLT) which preserves meta-properties of transformed functionality. Existing functionality often applies functions described as addition over Finite Field GF(n=2^k) and as applied in encryption such as AES-GCM and ChaCha20 and hashing as SHA-256, which are part of standard TLS 3.1. This functionality is modified by the FLT. Properties like involution, associativity and invertibility are preserved by the FLT. The FLT allows secret customization of existing and novel cryptographic primitives while maintaining proven data-flow. Improvement of security with a factor greater than 10^400 can be achieved.

Key Contributions:

- Introduces a novel framework for n-state operations in cryptography, transcending conventional constraints.
- Demonstrates the construction of n-state operations with properties defined over Finite Fields GF(n) but with unique numerical properties.
- Presents practical examples, showcasing the approach's versatility.
- Discusses potential implications for cryptographic algorithm design and security.

Keywords: cryptography, finite fields, n-state inverters, cryptographic primitives, involutions, Finite Lab-Transform, FLT, encryption, hashing, AES-GCM, ChaCha20, SHA-256

1. The Used Notation Herein

The applied notation herein is derived from the teaching approach of Prof. Dr. Gerrit Blaauw, one of the 3 chief designers of the legendary IBM System/360, as applied in his book "Digital System Implementation," [1]. Therein Blaauw uses APL to describe standard digital components like the AND gate by for instance: $c \leftarrow a \wedge b$ with 'a' and 'b' being binary input operands and 'c' its output. APL allows a symbol like θ to be introduced/defined as an operator representing, for instance, a specific look-up table or customized operation. One may also compute, in for instance Matlab, $c = \theta(a,b)$ wherein θ is a predefined lookup table. While unusual, there is

mathematically nothing wrong with this type of representation, as long as one observes the properties of the operation/table, like associativity or lack thereof.

A straightforward table-based notation is applied herein scn for n-state addition-like operations and mgn for multiplication-like operations. Thus, c=scn(a,b) or c=mgn(a,b) become the preferred notation herein. It allows direct replacement in computer programs by relevant look-up tables.

The following table provides the lookup table for modulo-5 addition:

sc5	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Instead of having to repeat or use the table one may use c=sc5(row,column), wherein one convention is that the first operand indicates the row index and the second one is the column index of a look-up table.

2. The Reversible N-state Inverter

An important concept in non-binary logic operations herein is the reversible n-state inverter. Its notation is as a function y=inv(x) wherein x, y ∈ ℤn. One representation of the n-state inverter is provided by its transformation from input to output states. The input states are all possible input states, represented by an index number starting at 0. So, input= [0 1 2 3 4 … n-1]. And thus input(0)= 0, input(3)=3 and input(n-1)=n-1. This is also explained in an earlier website [13].

Using a 5-state inverter inv5, we may have an inverter transformation input→output as [0 1 2 3 4] → [4 0 3 1 2]. Because the input is always input= [0 1 2 … n-1], with predetermined indices, we may also say: inv5= [4 0 3 1 2]. Because of the above convention one knows that inv5(0)=4, inv5(1)=0, inv5(2)=3, inv5(3)=1 and inv5(4)=2.

Each reversible n-state inverter invn has its reversing n-state inverter rinvn, with as property that invn(rinvn(x)) = x and rinvn(invn(x)) = x.
The reversing 5-state inverter rinv5 of the above inverter inv5 is then: rinv5= [1 3 4 2 0].

The above is explained in commonly used mathematical terms. In programming such as APL and C, it relates to array indexing starting at origin 0. Matlab starts array indices at origin-1 and statements like rinv5(0) = 1 will generate an error message. Using indexing starting at origin-1 is by itself not a problem as one may increase all inverter states with 1, as in Matlab. But it requires careful management of inverter statements.

Figure 1 illustrates an n-state inverter in a diagram.

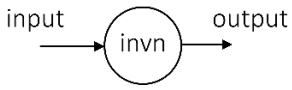

Figure 1

Program-wise, an n-state inverter is (in APL terms) a one-argument or monadic operation. It is represented by a one-dimensional table or array.

There are n^n different n-state inverters, ranging from [0 0 … 0] to [(n-1) (n-1) … (n-1)]. There are factorial of n (n!) different reversible n-state inverters. For n=3 one has as 3! = 6 reversible inverters: [0 1 2]; [0 2 1]; [1 0 2]; [1 2 0]; [2 0 1]; and [2 1 0].

The 3-state inverter [0 1 2] or in general invn = [0 1 2 … n-1] is the identity or identity inverter.

Furthermore inv3 = [1 0 2] has as reversing inverter rinv3 = [1 0 2] and is called self-reversing.

3. The Finite Lab-Transform (FLT)

The Finite Lab-Transform ("FLT") is a multi-argument operation with two or more input operands. The FLT will be used to transform n-state operations that are generally represented as two-operand operations, such as addition modulo-n. However, it is to be understood that the operation may be performed with more than 2 operands, like y= a+ b +c mod-n. The FLT may be applied to multiple arguments or operands, but with at least one output variable. The FLT applies first a reversible n-state inversion invn to all the input arguments. It then performs the multi-operand operation on the n-state inverted input arguments, using the n-state operation. It then generates an output of the n-state operation, which is inverted with the reversing inverter rinvn of invn.

The FLT on a 2-operand operation is shown below in diagram in Figure 2.

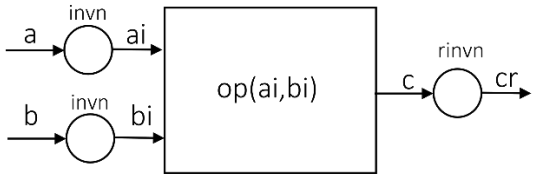

Figure 2

Figure 2 illustrates how the FLT is structured around a 2-operand operation c= op(a, b). Herein the operation op(a, b) may be represented by an n by n lookup table. Basically, it says cr=rinvn(op(invn(a), invn(b))).

One may replace the operation by a condensed operation cr= opf(a,b), wherein the inverters are "moved into" the table of operation op(a,b). Consequently, the FLTed operation may be represented by single lookup table operation cr= opf(a,b). This is illustrated in Figure 3.

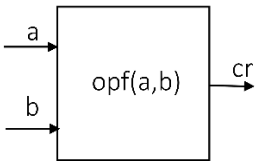

Figure 3

This reduction, which may be achieved by a computer program, performs the steps of Figure 2 but stores the results in a lookup table as provided in Figure 3.

Even for fairly large lookup tables, like a 1024-state 1024 by 1024 table, the execution is extremely fast. For instance, an operation like a multiplication modulo-n, which is associative and invertible, is also associative and invertible around a possibly different neutral element after being FLTed.

Take as example the multiplication modulo-5, named mg5, as provided in the Figure 4.

mg5	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Figure 4

The operation mg5 is FLTed using the inverter inv5 = [1 2 4 0 3]. The corresponding reversing inverter is rinv5 = [3 0 1 4 2]. The resulting FLTed operation is mn5, of which the look-up table is provided in Figure 5.

mn5	0	1	2	3	4
0	0	1	2	3	4
1	1	2	4	3	0
2	2	4	0	3	1
3	3	3	3	3	3
4	4	0	1	3	2

Figure 5

The modification of the zero-element and the one-element in this FLT is provided by rinv5, wherein rinv5(0)=3, which is the new zero-element and rinv5(1)=0 is the new one-element. The definition of a zero-element z in an operation op is op(a,z)=z for all a ∈ $\mathbb{Z}_5$. Normally z=0 in non-FLTed operations such as mg5. However, in mn5 z=3 as can be seen in Figure 5. One can check that mn5(a,3)=3 for all a ∈ $\mathbb{Z}_5$.

The one-element is defined as the element e for which op(a,e)=a for all a ∈ ℤn. This is commonly the element e =1. However, for mn5 due to the FLT the neutral element now is e=0, as in Figure 5.

Preservation of Properties under FLT

An important feature of the FLT is that it preserves properties (called "meta-properties" herein) of an operation that is FLTed. At the same time, the FLT may change one or more numerical outputs of an n-state operation. For simplicity an n-state operation may be defined by an n-by-n table, wherein an n-state output is defined by a row index and a column index of the n-by-n table. The indices may be considered the input operands and the n-state value in the table determined by the indices is its output. The table forms a closed system wherein the indices and the outputs are from the same set of n-state elements.

The FLT modifies the numerical content of the table, while keeping structural properties unchanged.

4. Finite Fields

The general definition of a finite field is often illustrated with a numerical example. A finite field or Galois Field GF(n) is closed, has two operations with an additive and multiplicative inverse, respectively, being associative and the two operations distribute. These are all well-known conditions of a finite field. Textbook examples such as [15] define an addition and a multiplication each with its inverse being defined relative to 0 and 1, respectively. Though that is correct as an example, it is not a necessary condition for a finite field.

This may leave the impression that there is only one finite field GF(n) with n being prime. But that is not correct. In fact, there are very many different finite fields GF(n) with n being prime.

This is illustrated with the FLT in the example in Figures 6 and 7. Figure 6 shows the lookup tables sc5 for addition modulo-5 and mg5 for multiplication modulo-5.

sc5	0	1	2	3	4	mg5	0	1	2	3	4
0	0	1	2	3	4		0	0	0	0	0
1	1	2	3	4	0		0	1	2	3	4
2	2	3	4	0	1		0	2	4	1	3
3	3	4	0	1	2		0	3	1	4	2
4	4	0	1	2	3		0	4	3	2	1

Figure 6

An FLT as illustrated in Figure 2 will be applied to the operations illustrated in Figure 6 with a 5-state inverter inv5=[2 4 1 0 3] and corresponding reversing inverter rinv5=[3 2 0 4 1]. This creates the FLTed 5-state functions sn5 and mn5 which are shown in Figure 7.

sn5	0	1	2	3	4	mn5	0	1	2	3	4
0	1	2	4	0	3		1	4	0	3	2
1	2	4	3	1	0		4	2	1	3	0
2	4	3	0	2	1		0	1	2	3	4
3	0	1	2	3	4		3	3	3	3	3
4	3	0	1	4	2		2	0	4	3	1

Figure 7

One can see in the table sn5 that its neutral element (also called the zero element) is 3, as both row and column at index three provide 5-state identity [0 1 2 3 4]. Or symbolically with c=sn5(a, b) and sn5(a, 3) = a for all a with z=3. The neutral element (or one-element) of mn5 is e=2 as mn5(a, 2) = a. The zero-element z in mn5 is (of course) the same zero-element of sn5 is z=3 and mn5(a,3) = 3 for all a.

One can check that both sn5 and mn5 are commutative and associative, both have inverses and the functions distribute and are closed. Thus, per definition, the set established by laws of composition sn5 and mn5 establish a finite field, wherein the zero-element is 3 and the one-element is 2.

One may perform all finite field operations using the FLTed functions. However, one should carefully manage the correct inverses. Accordingly, the multiplicative inverse in mn5 is determined relative to one-element 2. Assume an element a with ai its multiplicative inverse ai. Then mn5(a,ai)=2. From the table of mn5 one can read that with a=0 and ai=4; a=1 and ai=1; a=3 and ai=2; a=4 and ai=0, wherein ai is the multiplicative inverse of a over mn5. An inverse for a zero-element (a=3) is not defined.

6. Extensions of Operations and the FLT

Many cryptographic algorithms use computer functions that are defined over a finite field, GF(n=2^k). These finite fields are called extension fields and are based (in this case) on a base field GF(2). In some cases such as some cryptographic functions , only additions (and corresponding subtractions) are required. In some cases, only the multiplication.

Often applied functions over an extension field are additions over GF(n=2^k). These are executed on a computer by bitwise XOR of words of k bits and converting the result into its radix-n value. One may generate a related n-by-n lookup table that represents an addition over GF(n=2^k). These functions have corresponding subtractions, wherein the subtraction over GF(n=2^k) is identical to the addition over GF(n=2^k).

The FLT of these functions all work the same way, and as illustrated in Figure 2, with a first reversible n-state inverter for the input operands and a reversing n-state inverter of the first inverter at the output. If so desired, one may move the inverter into a lookup table as illustrated in Figure 3. However, for very large values of n this may not be practical and one has to apply, for instance, rule-based n-state inverters to create rule-based operations, rather than lookup

tables. This may be the case for large to very large numbers as applied in Diffie Hellman key exchange for instance. Matlab and C easily handle 1024-by-1024 1024-state tables.

Virtually all applications, when using operations over GF(2^k) use the bitwise XOR as the addition. And in AES-GCM and ChaCha20 encryption ([10] and [11]) the bitwise XOR is used for 'mixing' a generated keystream with the cleartext data. The 'mixing' function itself provides no cryptographic security therein. By applying the FLT to generate secret n-state mixing functions security is improved. Security comes from the immense number of different modifications that is possible by applying the FLT.

A problem of finding alternatives for functions that define a finite field GF(2^k), is the need for finding and using irreducible polynomials of degree k over base-field GF(2). While there are a number of computer programs to do that, they are often time consuming and may require insights into polynomial arithmetic. As such, it is user-unfriendly. Furthermore, the number of variations is still limited. A further limitation is that all thusly generated extension fields ALL have zero-element 0 and one-element 1. The FLT does not have those limitations and can be easily implemented using a structure as shown in Figure 2.

As an illustration, Figure 8 and Figure 9 show tables that define a classical extension field GF(8) with zero-element 0 and one-element 1 defined by polynomial x^3+x+1 and an FLTed version thereof with 8-state inverter inv8=[5 3 2 0 6 7 4 1].

sc8	0	1	2	3	4	5	6	7		mg8	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7			0	0	0	0	0	0	0	0
1	1	0	3	2	5	4	7	6			0	1	2	3	4	5	6	7
2	2	3	0	1	6	7	4	5			0	2	4	6	3	1	7	5
3	3	2	1	0	7	6	5	4			0	3	6	5	7	4	1	2
4	4	5	6	7	0	1	2	3			0	4	3	7	6	2	5	1
5	5	4	7	6	1	0	3	2			0	5	1	4	2	7	3	6
6	6	7	4	5	2	3	0	1			0	6	7	1	5	3	2	4
7	7	6	5	4	3	2	1	0			0	7	5	2	1	6	4	3

Figure 8

sn8	0	1	2	3	4	5	6	7		mn8	0	1	2	3	4	5	6	7
0	3	4	5	0	1	2	7	6			5	6	7	3	1	4	2	0
1	4	3	7	1	0	6	5	2			6	0	4	3	7	2	5	1
2	5	7	3	2	6	0	4	1			7	4	6	3	5	0	1	2
3	0	1	2	3	4	5	6	7			3	3	3	3	3	3	3	3
4	1	0	6	4	3	7	2	5			1	7	5	3	2	6	0	4
5	2	6	0	5	7	3	1	4			4	2	0	3	6	1	7	5
6	7	5	4	6	2	1	3	0			2	5	1	3	0	7	4	6
7	6	2	1	7	5	4	0	3			0	1	2	3	4	5	6	7

Figure 9

Both sc8 and mg8 in Figure 8 and sn8 and mn8 in Figure 9, establish a finite field GF(8). Both functions sc8 and sn8 are involutions (self-reversing). However, the zero-element in the FLTed

functions is z=3 and the one-element of mn8 is e=7. This modification of zero-element and/or one-element by FLT is believed to be novel.

7. What Function to Use and to FLT

It depends on the type of operation that needs to be performed that dictates what type of function needs to be applied. For instance, an encryption requires a decryption. That means that a function may have to be reversible for all relevant operands. One may use any n-state reversible function, including an addition over GF(n), any modulo-n addition and any reversible n-state function. These functions all may replace the traditional bitwise XOR of words of k-bits as used for instance in the encryption step of AES-GCM and ChaCha20. It requires the creation of a corresponding reversing function.

One tends to see encryption and decryption in terms of addition and subtraction, but that is not required. In fact, the only requirement is that each row (or column) of the encryption lookup table is an n-state reversible inverter. It is not even required that both columns and rows are n-state reversible tables. The sum-space of such operations always has a flat or uniform distribution over all outcomes.

In the application of elliptic curves, one is required to use operations over finite fields. This is because addition of points on elliptic curves requires both addition and multiplication of coordinates.

Diffie Hellman key computation requires multiplication over a finite field. RSA requires both multiplication and subtraction, but the composite p*q is of course not prime.

In AES-GCM and ChaCha20 encryption for keystream generation, as well as hashing such as SHA-256 and SHA-3, operations are one-way and need not to be reversible. Preferably they still should be balanced in output in the sense that no bias towards certain values exists. But the main requirement is that the operations are repeatable and can be done at both ends of a transmission.

8. Uniqueness of FLTs

How unique is an n-state FLT? While there are factorial of n (n!) different n-state reversible inverters, it has not been determined that there are also n! different FLTed functions of a base function. A different FLTed function would be one with at least one output state being different from the base function. Experimentation shows that duplication occurs, but can easily be detected. Experiments were conducted to find an indication of duplication.

9. FLT of Additions over GF(2^k)

A computationally difficult experiments in numbers, is the FLT of additions over GF(n=2^k). The number of reversible n-state inverters are: for n=2^1=2 there are 2 reversible 2-state inverters; for n=2^2=4 there are 24 reversible 4-state inverters; for n=2^3=8 there are 40,320 reversible 8-state inverters; for n=2^4=16 there are over $2*10^{13}$ reversible 16-state inverters; etc. To determine all different 16-state FLTed additions over GF(16) seems infeasible on a single computer.

The addition over GF(n=2^k) is an involution or self-reversing function. Here are some results for the FLTs. For n=4 there are 4 different 4-state involutions. For n=8 there are 240 different involutions. That is fairly low, indicating that n=8 would probably not be a good application for security. For n=16 there is a problem of generating the necessary permutations of the reversible 16-state inverters. One can take several approaches. A first approach is to generate with Matlab a random inverter inv16=randperm(16), create the related FLT of the original addition over GF(16) and continue the process until a duplicate FLT is generated. This was done for n=16 until 100,000 different 16-state involutions were generated.

A second approach for GF(16) was to create two sets of permutations. One for xx1= [1 2 3 4 5 6 7 8] which will generate in Matlab, 40,320 permutations with invt16a=perms(xx1). And one for xx2= [9 10 11 12 13 14 15 16] which will generate in Matlab also 40,320 permutations with invt16b=perms(xx2). By looping through both sets of permutations one can create inv16= [invt16b(i1,:)) invt16a(i2,:)] with i1 and i2 running from i1=1:40,320 and i2=1:40,320. With i2 running from 1 to 4 and i1 from 1 to 40,320 (or about 160,000 different 16-state inverters) there were about 150,000 different involutions over GF(16) generated. (150,000 different involutions over GF(16) were achieved at i2=4 and i1=29040). At that time the Matlab program was stopped manually. However, clearly more 16-state involutions could have been generated.

10. FLT of Multiplications over GF(2^k)

The different FLTs of multiplications over GF(2^k) occur is greater numbers than the additions. For n=4 there are 12 different FLTs and for n=8 there are 6720 different FLTs. Clearly, different FLTs of the multiplication over GF(2^k) have greater numbers than the addition over GF(2^k).

11. FLT of Additions/Multiplications over GF(n) with n Being Prime

The addition over GF(n) with n being prime is a modulo-n addition. To compare differences, the addition modulo-8 was FLTed with 40,320 different 8-state inverters and the resulting FLTs were compared to find the total number of different FLTs of addition modulo-8. This turned out to be 10,080 different FLTed reversible functions. One can repeat the FLT for different values of n and find out the results. For instance, for n=7 there are 840 different FLTs for additions modulo-7 and 2520 different FLTs of the multiplication modulo-7.

12. A Lower Bound, At Least (n-3)! Different FLTs

Based on the above and other related experiments, it is asserted with confidence that for any n there are *at least* (n-3)! different FLTs of functions that are of an addition or multiplication nature. In general, there are probably many more than (n-3)! of those types of FLTed functions. However, (n-3)! would be an acceptable operational minimum bound for making decisions in applying the FLT in cryptographic operations.

13. Computational Examples

MixColumns() in AES

As a computational example will be used the operation MixColumns() of the Advanced Encryption Standard (AES) as described in FIPS-197 [2] in sections 4.2 and 5.1.3. This operation is a 4 by 4 256-state matrix multiplication over GF(256) with a vector or array of 4 256-state elements.

The prescribed array multiplication is provided in Figure 11 below, copied from (5.7) in [2]:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Figure 11

The elements in the array are in hexadecimal notation. Each row is a shifted version of a prior one.

In the decryption in AES the operation is an inverse and is defined as InvMixColumns() as in (5.14) of [2] and it shown in Figure 12.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Figure 12

The first row can be read as [14 11 13 9] in decimal.

One may apply a random 256-state reversible inverter to FLT the addition and multiplication over GF(256). The 256-state inverter is; inv256= [122 98 173 243 … 9 11 225 117], showing the 4 and last 4 elements of the inverter. The reversing 256-state inverter is rinv256= [210 228 28 231 … 139 60 129 165].

Using the same AES array for encryption in MixColumns() but with the FLTed addition and multiplication, generated with the inverter inv256, the inverse matrix for InvMixColumns() is shown in Figure 13 as follows:

41	202	254	163
163	41	202	254
254	163	41	202
202	254	163	41

Figure 13

Figure 13 shows the result in decimal form. One should be aware that the matrix multiplication (which is a combination of multiplication and addition) under the FLT gives a different outcome for MixColumns(). The generation of the inverse matrix is like the normal inverse matrix computations, including a determinant, but wherein all the operations (+, - and *) are now FLTed operations. This may create a problem, because computer operations like determinant and matrix inverse are often internal operations in computer languages. The above InvMixColumns() matrix has been computed using standard functions from FIPS-197 [2] and FLTed appropriately. These operations are easy to program when using a small (4 by 4) matrix. The subroutine for FLTed inverse matrix computation and FLTed vector matrix multiplication has been reprogrammed for medium large matrices like 100 by 100 element arrays, and work well and fast in Matlab.

As a further example, a vector [37 154 12 233] is first "mixed" with the standard operation and generates as output [26 247 135 48]. The FLTed operation in MixColumns() generates [177 229 162 206].

The result is entirely different and thus the FLT, even when applied only in one round, will contribute to an entirely different ciphertext in AES.

RSA Example

The RSA key-exchange algorithm currently is recommended for keys with a size of at least 2048 bits or equivalent over 600 digits decimal. That is not very instructive as an example. Therefore, one is referred to the explanation and toy example of RSA on Wikipedia [3]. The example finds an encryption exponent e=17 and decryption exponent d=413 for a multiplication modulo-3233.

For the FLT, the same exponents e and d are used. The multiplication is mg3233 and is modulo-3233. The multiplication modulo-3233 is FLTed in Matlab, by using inv3233=randperm(3233). The FLT with inv3233= [1717 352 2112 2617 … 102 1758 1618 457 2570] and is used in FLT to create mn3233. The FLTed RSA uses the same exponents, but replace mg3233 with mn3233.

In the Wikipedia example m=65. Using the FLT and mn3233, the ciphertext is then c=421, instead of the original 2790. Using mn3233 with d=413 recovers the cleartext.

Involution Encryption Example, 8-state

An addition over GF(2^k) is formed by XORing 2 words of k bits and replacing the binary word by its decimal value. Every addition over GF(2^k) is an involution, which means that the functions is its own inverse. For instance, if we name the addition over GF(2^k) as scn. Then with

operands a and b: c=scn(a,b). And a=scn(c,b) and a=scn(b,c), etc. The FLT of scn is also an involution.

Using an 8-state message mes=[2 2 5 3 4 3 4 2 0 1 4 2 4 7 5 6] and a key=[1 7 0 1 4 6 0 6 2 4 3 6 1 6 2 7] using the involution sc8 one gets as ciphertext ciph1=[3 5 5 2 0 5 4 4 2 5 7 4 5 1 7 1] and with sn8 ciphertext ciph2=[7 1 2 1 3 6 1 4 5 0 4 4 0 0 0 0].

14. Other Examples

One can create other examples to demonstrate the effect of the FLT, including in Elliptic Curve Cryptography (ECC) and hashing for instance. In SHA-256 as defined in FIPS 180-4 [4] a function $Ch(x,y,z)=(x \wedge y) \oplus (\neg x \wedge z)$ is defined as (4.2) of bitwise execution of words of 32-bits.

One may represent the individual bitwise operations into n-state operations similar to as was done with the XOR operation. One may then FLT the thus created n-state operations.

The author has developed a wide range of applications that have been disclosed in US Patents. [5], [6], [7], [8] and [9]. One application is in the creation of novel n-state Feedback Registers (FSRs).

FLTed modified n-state operations, not being invertible operations, as illustrated above in case of SHA-256 may also be applied. In the example of Ch(x,y,z), the output of the function is balanced, preventing a detectable bias, which otherwise may provide an opening for cryptanalysis. AES in AES-GCM is used in a one-way mode to generate a keystream, but not a reversible keystream. That is: both encryption and decryption require the same keystream. This allows processes in AES that must be reversible in generating a cipher and an inverted cipher, may apply non-invertible processes in AES-GCM mode, where only repeatability (and non-leakage of critical information) is required. The same applies to ChaCha20 [11] wherein for instance bitwise XORing and addition modulo-2^32 are used for generating a keystream, used in encryption and decryption. One modification may be achieved with the FLT or at least with a shared n-state inverter between encrypting and decrypting computer devices.

15. Software Examples

In order the reader to facilitate own experimentation, several computer programs in Matlab and C have been published and may be freely downloaded from https://lcip.in/ [12] for educational and trial use only. Please read the related license agreement when downloading these programs.

16. The FLT and Quantum Computing

A looming threat to current cryptography is quantum computing. Quantum computing is expected to be effective in discrete logarithm and factorizing based attacks. And applied to Public Key Infrastructure (“PKI”) keywords. This means that an attacker will be able to determine critical parameters in key exchange (PKI), and thus render the related cryptographic primitive ineffective. A substantial segment of cybersecurity experts believe that this threat is

real and will be operational within the foreseeable future. These experts also believe that Harvest Now, Decrypt Later ("HNDL") attacks are now actively applied.

The FLT is, looked upon from the outside, a random change in functionality and is from that perspective not open to discrete logarithm or factorization attacks. Most likely only brute force attacks on the FLT are currently possible. It is believed that this makes the HNDL attack useless for the foreseeable future.

While often QC-attacks seem to cause alarm, novel attacks and growing computer power also affect security of current cryptography. This is exemplified by the recommendation to increase the size of PKI parameters. For instance 512-bit key for RSA has been deemed insecure and a 2048-bit key is recommended. For extreme secure requirements for the next 10 years, a 4096-bit RSA key is recommended.

Furthermore, the FLT can be applied to some Post Quantum (PQ) methods as proposed in the National Institute of Standards and Technology PQ program. Some of the FLT applications are disclosed in a US Patent Application [14].

17. Conclusions

Security of cryptographic machines is usually obtained from two aspects:
1) a set of well described operations that in combination create an output that is intractable to be inverted to an input operand; and
2) at least one operand that is so large that brute force attacks in the context of the combined operations are infeasible.

The herein described FLT is different in that it modifies the computer functions in the operations. The intractability of the number of modified functions is greater (>10^400 for n=256) than security provided by the size of a 256-bit keyword for instance (2^256), making it unlikely that even all available computing power in the world will successfully attack by brute force FLT encrypted data during the lifetime of the universe. In that context, it is possible to restore with the FLT security of already broken cryptographic primitives like DES [16]. It requires that these primitives have sufficiently large internal word sizes (like 48 or 56 bits) to tap into the enormous space of FLT modifications..

The FLT can be applied to a wide field of cryptographic methods, including encryption and hashing. Its use has been demonstrated in applications such as AES, AES-GCM, ChaCha20, RSA and SHA-256 as illustrative examples. The FLT has been described and demonstrated in terms of n-state operations and/or lookup tables. Rule-based FLTs are also possible.

It was suggested that a factor (n-3)! in increased security against brute force attacks may be achieved. Also, ease of use of FLT may offer a significant opportunity to customize security or add a private level of security to Cloud stored data.

The FLT overall is a novel development that suggests a significant increase in security of cryptographic methods. In that sense it may be worthwhile to further investigate its potential.

18. License to Use the FLT in Patented Applications

Certain aspects of the FLT are claimed in issued USPTO Patents ([6], [7], [8] and [9]). The patents are assigned to LCIP jv. LCIP jv provides explicit license to use the claimed invention for trial, research and educational purposes only. All downloadable programs have a license statement. Use of these programs is at your own risk and is only allowed with the complete license statement being copied in the applied source code. Furthermore, any use of results from these programs in publications must be accompanied by the statement that Peter Lablans is the inventor of the FLT. Other usage of the programs, or application of the claimed FLT is expressly precluded from the above license. This specifically pertains to operational data encryption and hashing in operational storage and/or exchange of data. Permission and license for operational use can only be obtained by written permission provided by Peter Lablans. Such license may require a reasonable royalty or license fee. Please contact Peter Lablans at info at labcipher dot com for further information.

References

[1] Gerrit A. Blaauw, Digital System Implementation, 1976, Prentice-Hall, Inc., Englewood Cliffs, NJ

[2] National Institute of Standards and Technology (NIST). (2001, November). Advanced Encryption Standard (AES) (Federal Information Processing Standard (FIPS) 197). https://csrc.nist.gov/pubs/fips/197/ipd

[3] Rivest, R., Shamir, A., & Adleman, L. (1978). RSA cryptosystem. Retrieved from https://en.wikipedia.org/wiki/RSA_(cryptosystem

[4] National Institute of Standards and Technology (NIST). (2015, August). Secure Hash Standard (SHS) (Federal Information Processing Standard (FIPS) 180-4). https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.180-4.pdf .

[5] US Patent Ser. No. 11,336,425B1 to Peter Lablans, entitled Cryptographic machines characterized by a Finite Lab-Transform (FLT), issued on May 27, 2022

[6] US Patent Ser. No. 11,093,213B1 to Peter Lablans, entitled Cryptographic computer machines with novel switching devices, issued on Aug. 17, 2021

[7] US Patent Ser. No. US 10,650,373B2 to Peter Lablans, entitled Method and apparatus for validating a transaction between a plurality of machines, issued on May 12, 2020

[8] US Patent Ser. No. US 10,515,567B2 to Peter Lablans, entitled Cryptographic machines with N-state lab-transformed switching devices, issued on Dec. 24, 2019

[9] US Patent Ser. No. US 10,375,252 to Peter Lablans, entitled Method and apparatus for wirelessly activating a remote mechanism, issued on Aug. 6, 2019

[10] National Institute of Standards and Technology (NIST). (2001, November). Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC for Advanced Encryption Standard (AES) (Special Publication 800-38D). https://csrc.nist.gov/pubs/sp/800/38/d/final

[11] ChaCha20 and Poly1305 for IETF Protocols, Request for Comments: 7539, May 2015, https://datatracker.ietf.org/doc/html/rfc7539

[12] Increasing Security in Cryptography FLTed Encryption and Hashing, website with example Matlab and C code, https://lcip.in/

[13] The Finite Lab-Transform (FLT), a website https://www.labtransform.com/

[14] US Patent Application Ser. No. 17/402968 to Peter Lablans, entitled Cryptographic Computer Machines With Novel Switching Devices, filed on Aug. 16, 2021

[15] Iain T. Adamson, Introduction to Field Theory, second edition, Dover Publications 2007

[16] National Institute of Standards and Technology (NIST). The Digital Encryption Standard (DES). FIPS-PUB 46-3. Withdrawn in 2005. https://csrc.nist.gov/files/pubs/fips/46-3/final/docs/fips46-3.pdf

Biography:

Peter Lablans received a B.Sc. and M.Sc. (Ir.) degree in electrical engineering from the Technische Hogeschool Twente in Enschede, the Netherlands. He did his B.Sc. thesis on a RISC machine in the Digital Technique Group of Prof. Dr. Gerrit Blaauw and his M.Sc. thesis under Dr. Paul Lambeck on ferro-electric memory material. Peter Lablans was the project manager for the first public digital optical fiber project in the Netherlands, taught advanced digital signal processing, was an assistant science attaché, held several executive positions in nosiness development at software companies and was a patent engineer specialized in image and signal processing patent prosecution on hundreds of issued patents. Mr. Lablans currently is a prolific inventor and is the named inventor of over 50 US Patents. He resides in New Jersey.