# Cryptography using mechanical implementation over Galois fields.

Ashton Snelgrove
*Electrical and Computer Engineering*
*University of Utah*
Salt Lake City, Utah, USA
ashton.snelgrove@utah.edu

Miguel Gomez
*Electrical and Computer Engineering*
*University of Utah*
Salt Lake City, Utah, USA

***Abstract*—In the shadow of World War II, the cryptographic landscape was shaped significantly by mechanical cipher machines such as the Enigma and the Lorenz cipher. These devices were systematically deciphered by the cryptanalysts at Bletchley Park, whose work crucially influenced the outcome of the war and propelled forward the fields of mathematical cryptography and computational science. This paper explores the feasibility of designing cryptographic schemes using only the mechanical and early digital computation technologies available at the time but informed by contemporary cryptographic knowledge. We propose a security scheme with key-distribution, authentication, and a stream cipher, implemented with contemporary mechanical technology. We then evaluate the computational power available during different historical periods and assesses the required key sizes and algorithm complexities that would have been necessary to thwart decryption efforts. Additionally, practical considerations such as cost, durability, and ease of use in field conditions are addressed.**

***Index Terms*—Keywords**

## I. Introduction

Prior to the invention of the digital computer, cryptography was done by hand or by using mechanical devices. World War II saw the height of the mechanical cipher, with famous examples including the Enigma machine and the Lorenz cipher. These ciphers remain of interest in the history of cryptography not necessarily for their own efficacy, but the innovations they spurred in the world of computing. The Bletchley Park code breaking unit was responsible for the cryptanalysis and breaking of the Axis encryption schemes, with a major impact on the outcome of the conflict. They utilized mathematical theory, mechanical computing, and ultimately digital computation.

All of these systems have technical and operational weaknesses exploited by code breakers. The question becomes: using current knowledge of cryptography concepts and ciphers, is it possible to create cipher schemes that would be infeasible to break and implemented using the types of mechanisms available at the time?

This paper explores the feasibility of designing cryptographic schemes using only the mechanical and early digital computation technologies available at the time but informed by current cryptographic knowledge. We aim to determine whether it is theoretically possible to construct cipher schemes that would have been secure against the code-breaking capabilities of the era, even under the assumption of attackers with modern-day cryptographic understanding and significant resources. The requirements for the cryptographic scheme include robust key-distribution, effective authentication, and a cipher that remains secure under Kerckhoffs's principle. Additionally, practical considerations such as cost, durability, and ease of use in field conditions are addressed.

We demonstrate the mathematical basis for operations over non-binary Galois fields and the application to the cryptographic problem. We then propose a stream cipher, key exchange, and authentication scheme over a non-binary field. The mechanical implementation of the scheme is presented and evaluated for feasibility. We attempt to evaluate the computational power available during different historical periods and assesses the required key sizes and algorithm complexities that would have been necessary to thwart decryption efforts. We also present a discussion on the implications of these findings for modern cryptography.

This historical "what-if" analysis not only deepens our understanding of cryptographic progress and limitations but also underscores the evolving challenges in securing communications against increasingly sophisticated threats.

The rest of the paper is organized as follows. Section II recounts the historical context and function of encryption techniques during WWII. Section III sets forth the requirements for an encryption scheme. Section IV explores the theory and implementation of stream ciphers. Section V examines key exchange and authentication. Section VI presents the mechanical implementation. Section VII analyzes the security of the scheme against contemporary and current cryptanalysis. Section VIII concludes the paper.

## II. State of the art cryptography, circa 1938-1946

***TODO: need more time to write up history here with citations.***

### A. The competition

Of the cipher machine in use during this period, the most famous and well studied are those used by the Germany - the

The authors wish to thank Dr. Priyank Kalla for putting up with our ridiculous ideas, and Dr. Sanford Meeks for sharing his knowledge of mechanisms and the generous access to his collection.

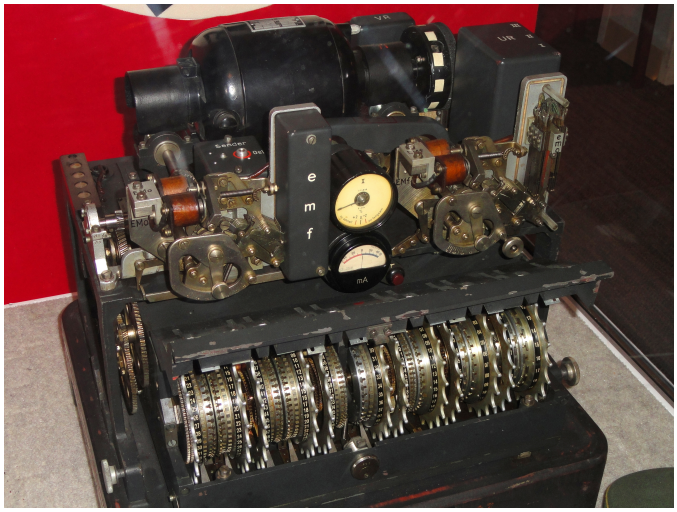

Fig. 1. *Lorenz SZ40 cipher machine, located in the National Cryptologic Museum in Fort Meade, Maryland*

Enigma and Lorenz. Both were broken by the Allied code-breaking units.

Sophisticated rotary machines continued to be used through the 70's, including the NSA designed KL-7 and the Crypto AG HX-63, both considered practically unbreakable[1].

The internals for Enigma and Lorenz machines were inferred by cryptophers. Obfuscation of internals may have been considered part of the security of the cipher.

*1) Enigma:* The Engima is a poly-alphabetic substitution cipher, working with a set of rotating wheels, which change electrical contacts modifying the substitution. The key is the selection of the rotor wheels and seed position. This was the most widely distributed machine. The common machine used by the German army used fewer wheels than those by the Navy, which greatly reduced their efficacy.

*2) Lorenz:* The Lorenz SZ40, SZ42a and SZ42b were rotary stream ciphers, generating a pseudo-random stream of data that is then XOR'd with the plaintext. An example is shown in Figure 1 The machine operated off a set of 12 wheels, all of which had prime number cams. The device was an attachment to a teletype printer, and the data was sent over 5bit telegraph code. [2].

### B. The adversary

The Allied cryptography unit at Bletchley Park brought together a collection of some of the best mathematicians, statisticians, and cryptographers available.

*1) Bombes:* Enigma was broken by a "bombe", which mechanically emulated multiple rotor configurations. This was not an exhaustive search, but one informed by patterns in the data recognized by cryptographers. The contacts of each rotor where able to be determined statistically. Polish cryptographers had already developed the concepts necessary to do the cryptanalysis before the invasion of Poland, when they escaped with their knowledge to the Allies.

*2) Colossus:* Colossus was the first digital computer, developed to attack the Lorenz cipher. Statistical patterns in a message would be recognized by cryptographers, which would inform different possible configurations that were tried via the Colossus. It was not a programmable computer, though it is discussed by B. Wells in [3] that it could theoretically be possible to build a Turing machine. From the abstract:

> A universal Turing machine could have been implemented on a clustering of the ten Colossus machines installed at Bletchley Park, England, by the end of WWII in 1945.

It was designed for the specific operation of searching the Lorenz.

## III. Scheme requirements

A scheme design needs to account for key-distribution, authentication, and a cipher.

We begin with a discussion of the threat model.

1) The attacker is assumed to have knowledge of all internal workings of the scheme except the key, following Kerckhoffs's principle.
2) Attacker has the unlimited resources of a nation, such as the Bletchley Park team or another security agency.
3) The capacity of human, mechanical, or digital computation available is bounded by technological limitations.
4) Attackers could have same level of cryptographic theory as present day.
5) Minimize the damage from breaking one key.
6) Mechanism, physical codebook or key capture is possible, and can be expected to occur if a unit is overrun.
7) All messages are sent over a public channel.

Practical aspects of any implementation.

1) Cipher should be reversible, meaning no separate encryption/decryption hardware.
2) Random numbers must be generated in a true random controlled manner, not relying on operators to pick.
3) The mechanical devices of the scheme should be inexpensive enough to match production numbers of contemporary devices.
4) The devices should be rugged enough for field use.
5) Use of the scheme should be simple enough that a communications officer in the field could use it with minimal training.
6) Data would be transmitted manually with Morse code.

## IV. Stream ciphers over non-binary fields

We can begin by defining the following terms:

| | |
|---|---|
| $\mathbb{F}_q$ | The finite field of $q$ elements |
| $\mathbb{F}_{q^k}$ | The finite field of $q$ elements raised to $k$ |
| $\mathbb{F}_2 \equiv \mathbb{B}$ | The finite field of 2 elements, Boolean space |
| $\mathbb{Z}_n^*$ | The co-prime set of integers less than $n$ |
| $\phi(n)$ | Euler's $n^{\text{th}}$ Totient number |
| $P(x)$ | a primitive polynomial in one variable $x$ |

### A. Stream Cipher

Stream ciphers would have the advantage over block ciphers due to simplicity and the need for fewer elements, notably storage.

The simplest implementation of a stream cipher is to take a random or pseudo-random stream of text and combine it with the cipher text. The most common variant is to take two values and sum them modulo the field. For operations in $\mathbb{B}$ this is done using addition modulo 2, the XOR function.

This has the advantage of simplicity and reversibility - given the keystream and the enciphered text, the same operation is used to decipher the plaintext. A true random keystream, also known as a one-time pad, is unbreakable[4]. Distribution of shared keystreams (the pad) is the biggest limitation of such a scheme, so a pseudo-random number generator is used. The security of a stream cipher thus is founded on the randomness of the keystream.

### B. Linear Finite Shift Registers

A linear finite shift register (LFSR) represents a polynomial as a set of stored coefficients. LFSRs are typically built over the field $\mathbb{F}_{2^k}$, due to the ubiquity of digital boolean hardware. However, non-binary Galois linear feedback shift registers can also be built[5]. Finite fields $\mathbb{F}_{p^k}$ have maximum period generators available as long as $p$ is prime.

As for the choice of $p$, a device could be built to function in the field $\mathbb{F}_{p^k}$, where $p$ is the size of the alphabet to be encrypted. 37 would be a useful choice: 26 letters, 10 numbers, and a period mark give a prime number in a reasonable size that is realizable with a gear. The key length $k$ can be arbitrarily long, and complexity scales quickly. Each increase in key length achieves the same complexity as $\log_2(37) \approx 5.21$ bits.

We will evaluate an example primitive polynomial (1) in $\mathbb{F}_{37^4}$

$$P(x) = x^4 + x^2 + 2 \mod 37 \tag{1}$$

For a starting polynomial $gx^n$ with coefficients $c_0 ... c_3$, we multiply by $x$.

$$\begin{aligned} gx^n &= c_3x^3 + c_2x^2 + c_1x + c_0 \\ gx^{n+1} &= c_3x^4 + c_2x^3 + c_1x^2 + c_0x \\ gx^{n+1} &= c_3(-x^2 - 2) + c_2x^3 + c_1x^2 + c_0x \\ gx^{n+1} &= c_2x^3 + (c_1 - c_3)x^2 + c_0x - 2c_3 \end{aligned} \tag{2}$$

This yields equation (2), which shows the coefficients after one iteration. We can see that each coefficient is a function of the coefficients of the previous iteration.

The more complicated primitive polynomial (6) produces the same behavior. While functional, the choice of primitive polynomial has implications for implementation complexity.

$$\begin{aligned} P(x) &= 33x^4 + 33x^3 + 25x^2 + 21x + 2 \\ x^4 &= -33^{-1}(33x^3 + 25x^2 + 21x + 2) \end{aligned}$$

$$\begin{aligned} gx^{n+1} = (-c_3 + c_2)x^3 + (-3c_3 + c_1)x^2 + \\ (c_0 - 4c_3)x - 18c_3 \end{aligned} \tag{3}$$

Figure 2 shows a block diagram of an implementation of equation (2)

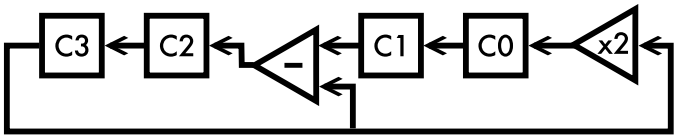


Fig. 2. *Implementation of* $gx^{n+1} = c_2x^3 + (c_1 - c_3)x^2 + c_0x - 2c_3$ *as an LFSR.*

As we are working with a linear system, the required operations are polynomial multiplication, multiplication by a constant, and addition - all done modulo $p$.

The typical LFSR implementation is done over $\mathbb{F}_{2^k}$, where the addition operation modulo 2 is the XOR operation, multiplication is the AND operation, and polynomial multiplication is done via shift registers. Because the only coefficients are 1 and 0, no constant multiplication or negative coefficients are explicitly needed.

### C. Non-linear FSR

With the use of a primitive polynomial, the LFSR demonstrates a maximal non-repeating period $p^k - 1$. This yields a significantly large number of possible values for an appropriately chosen $k$. LFSRs also demonstrate balanced probability distribution, reducing the opportunity to use bias in the stream during cryptanalysis. However, an LFSR on its own as a pseudo-random number generator has some weaknesses. Of particular concern is that if a small but significant number of values from the keystream are sent in the clear the original key can be derived[5]. The general solution to this problem is to introduce non-linearity into the system, forming a non-linear finite shift register (NFLSR).

One possible scheme is to take multiple LFSRs and feed the output into a non-linear combining function. Other options include using one LFSR to control another's behavior - for example the shrinking generator, where the output of one LFSR is used to determine if the output of the other should be used or discarded. Ciphers like MICKEY[6] utilize internal conditions to irregularly clock the registers.

More advanced stream ciphers, such as those from the eSTREAM program [6], utilize sophisticated yet minimalist schemes to improve security. Developing a new cipher is not easy - of the ciphers submitted to eSTREAM, being broken was not atypical and 7 ciphers were ultimately chosen[7]. There are no known methods for determining if an NFLSR has full periodicity. There are desirable properties of non-linear combining functions, notably correlation-immunity[8]. Evaluating current stream ciphers in the eSTREAM program, given our current understanding, we are uncertain of the possible translation into a field with $q > 2$. While the design of a non-binary NLFSR is beyond the scope of this work,

it is worth noting possible non-linear operations available for future investigation.

For this proposed scheme, the use of the LFSR alone is considered to be acceptable, and we further analyze the consequences and possible solutions in Section VII.

## V. Key exchange and authentication

### A. Asymmetric key-exchange

The Diffie-Hellman-Merkle (DFM) key exchange relies on a one-way function - one that is easy to calculate but difficult to find the inverse[9]. In the case of the original DFM scheme the operation is modular exponentiation. Calculation of the inverse requires a solution to the discrete logarithm problem, which is believed to have no polynomial algorithm. There are several simple algorithms to calculate modular exponentiation over $\mathbb{Z}$, but each requires many iterations of large multiplications. Well within the realm of modern digital computers, but would be difficult to do either mechanically or by hand. Modular exponentiation is the calculation of $B = g^b \mod p$. The common algorithm requires $b$ repeated multiplication of $g$ with an intermediate result, modulo $p$. $g$ can be a small integer such as 2 or 3, but in practice $p$ and $b$ are very large.

### B. Modular exponentiation over cyclic groups

However, the modular exponentiation concept can be extended to cyclic groups. Given a generator $g$, the root of a primitive polynomial $P(x)$, the DHM requires the calculation $g^a$.

- Alice and Bob agree on generator $g$, which is the root of a primitive polynomial over $\mathbb{F}_{p^k}$. This is public knowledge.
- Alice and Bob generate random private keys $a$ and $b$, which are integers $1 \leq a \;\; b \leq p^k - 1$.
- Alice calculates $A = g^a$ and Bob calculate $G = g^b$. The two exchange $A$ and $B$ over a public channel.
- Alice then calculates $B^a$ and Bob calculates $A^b$. Both now have shared secret $g^{ab}$.

Exponentiation of an arbitrary unknown polynomial can be accomplished with a simple algorithm. The algorithm is to do repeated squaring and multiplication by root $g$. For some $g^n$, there exists some $k$ such that $n = 2 \cdot k$ or $n = 2 \cdot k + 1$, requiring calculation of either $g^{2k}$ or $g^{2k+1}$

A simple variant would be to generate a binary number sequence (e.g., flip a coin). If true, square and multiply by $g$; if false, just square. Marking down the binary sequence and applying the key to the value received from the other party requires repeating the sequence.

The initial calculation step three, $g^a$, can be accomplished in fewer operations by leveraging the simplification that any number $g^a$ can also be expressed as $g^{n_k p^k} \cdot g^{n_{k-1} p^{k-1}} \cdot \ldots g^{n_0}$. For the field $\mathbb{F}_{p^k}$, a table can be pre-computed, with $k$ columns of $p$ items, containing $g^{0 \cdot p^k}$ to $g^{(p-1) \cdot p^k}$. Given the random $k$ digit key, the calculation can be done in $k$ multiplications.

**Algorithm 1** Exponentiation by Squaring

**Require:** An integer $g \geq 1$ and an element $B$ of a multiplicative group
**Ensure:** Returns $B^g$

```
G ← B
g ← a
while g > 1 do
  if g is odd then
    g ← g − 1
    G ← G · x
  else
    g ← g/2
    G ← G^2
  end if
end while
return G
```

Preventing generating weak random numbers at the user side is one of our requirements. If the user has the LFSR from sectionIV, this can be used as a random number generator. The choice of seed could still create a bias, especially given the soldier's propensity to choose simple numbers and obscenities as nonces.

### C. Authentication

DHM does not provide authentication, which leaves it open to man-in-the-middle attacks. The addition of a shared password to do password-authenticated key exchange would be possible. (***TODO: we didn't have time to do the math on this***). Distribution of shared secrets is discussed later in this section.

### D. Implementation

Multiplication over finite fields can be done with the Mastrovito multiplier architecture.

Mastrovito multiplier over $P(x) = x^4 + x^2 + 2$

$$\begin{aligned}(ax^3 + bx^2 + cx + d)(ix^3 + jx^2 + k + l) \\ = x^3(-ai - bi + di - bj + cj + bk + al) \\ + x^2(-ci - bj + dj - ak + ck + bl) \\ + x(-2bi - 2aj + dk + cl) \\ + (2ai - 2ci - 2bj - 2ak + dl) \quad (4)\end{aligned}$$

Mastrovito squarer over $P(x) = x^4 + x^2 + 2$

$$\begin{aligned}(ax^3 + bx^2 + cx + d)^2 \\ = x^3(-a^2 + 2ad + 2bc) + x^2(-2a^2 - 2ab + 2bd + c^2) \\ + x(-4ab - 2ac - b^2 + 2cd) + (-4ac - 2b^2 + d^2) \quad (5)\end{aligned}$$

The Mastrovito squarer for any primitive polynomial with three terms in $\mathbb{F}_{37^4}$ has 10 multiplications and sum of either 14 or 17 terms, with constant multiplication possible on each term. ***TODO: including a block diagram for conference would be awesome***

| | $x^3$ | $x^2$ | $x^1$ | $x^0$ |
|---|---|---|---|---|
| $a^2$ | -2 | | 1 | 2 |
| $ab$ | -2 | -4 | | |
| $ac$ | | -2 | -4 | |
| $ad$ | | | -2 | -4 |
| $b^2$ | | -1 | -2 | |
| $bc$ | | | -2 | -4 |
| $bd$ | 2 | | | |
| $c^2$ | 1 | 2 | | |
| $cd$ | | 2 | | |
| $d^2$ | | | 1 | |

TABLE I
MASTROVITO SQUARER $g^2$ COEFFICIENTS FOR PRIMITIVE POLYNOMIAL $P(x) = x^4 + x^2 + 2$

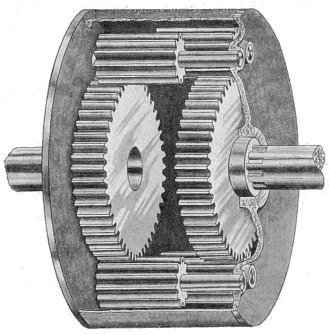

Fig. 3. *Spur-gear differential. The rotation of the two independent axles produces a carrier rotation equal to the sum of the two rotations.*

### *E. Shared secrets*

The application of cryptography to a highly hierarchical organization such as a military chain of command brings some advantages. The tree-like command structure of a military organization can be used to do key authority. Disruption or compromise of a unit's parent is expected to reflect a breakdown in the chain of command or the efficacy of the unit. Unless told otherwise by a higher authority, your command is assumed to be a trusted authority. General communication is expected to be up or down at most one level of the hierarchy, from a unit to it's parent, siblings, or children. Communication with a further ancestor is possible, for example to revoke authority.

Each unit has a paired secret for all its ancestors, descendants, and siblings. Each unit could also have a shared secret with all its siblings and parent to allow broadcast. A parent of a unit can revoke it or any children's authority by using their broadcast token. Knowledge of the shared secret authenticates. Secrecy at each level is expected to be naturally higher up the chain. Secrets are true random, generated by central source.

Distribution of shared secrets remains a problem - one likely best solved by good old fashioned hand delivery.

### *F. Key codebook*

Given the necessity of distribution of shared secrets, the simple option would be to directly physically share keys rather than relying on the rather costly computations required for asymmetric exchange. Historically this is implemented with the distribution of code books. The one-time pad is unbreakable, but suffers from two coupled issues: shared distribution of a secret keystream that must be as long as the message being sent. Leveraging the generative nature of the stream cipher keystream, the keystream length problem can be mitigated, using a large number of easily changed true random stream keys.

Each unit has matching code book for each of its parent, grandparent, children and grandchildren, and siblings. Capture of one unit's codebook fails to reveal any secrets beyond that unit's relations. Secrets are true random, generated by central source. Code books have an randomly generated or sequential index mapped to a stream key. Each book has a sufficiently large number of entries to send enough messages between courier deliveries of new code books. Choice of key from a codebook for a message exchange is arbitrary, index is sent in the clear. Codebook entries should be destroyed once used, for example printed on paper which is burned. Code books would be distributed by secure courier. Authentication of receipt of the codebook by a unit could be done with the costly asymmetric key exchange. Given a sufficiently large codebook, keys could changed frequently.

Experimentally, a large number of keys can be stored compactly. An codebook page was generated for keys of length 11 with 9 digit identifiers. An A4 size page with 1 inch margins and three columns, with 10 point Courier font, is capable of holding 159 keys. At 8 point font increases capacity to 186.

## VI. MECHANICAL IMPLEMENTATION

### *A. LFSR*

A gear mechanism provides all the required operations to implement an LFSR. Gears implement modulo by virtue of being round - the modulus is the number of teeth present on the gear. A differential gear arrangement can be used to sum two rotations into a third rotation (Figure 3). Constant multiplication is done with a transmission ratio between two different sizes of the appropriate value.

The most complicated operation is polynomial multiplication, which requires adding discrete state into the system. This can be accomplished by using a clutch system to engage and disengage gears to shift data between coefficient storage gears. The abstract operation of such a system is shown in Figure 4. The process requires a two-stage operation. The first operation is to transfer the values from the storage gear into a temporary transfer gear. The transfer gear is disconnected from the current gear and connected to the next storage gear, transferring the value. The mechanism is reset, and the procedure is repeated for each multiplication.

The number of operations required depends on the primitive polynomial chosen. For primitive polynomial $x^4+x^2+2$, four registers, one constant multiplication, and one subtraction are required, as seen in Figure 2. Increasing the length of register digits $k$ causes a linear increase in the number of operations required.

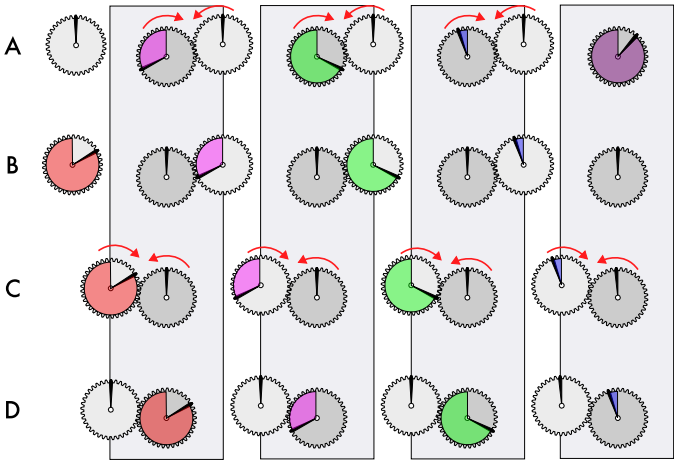


Fig. 4. *Implementation of a mechanical shift register. Four stages of the operating cycle are shown: (A) Storage gears (dark grey) are stable and hold a value indicated by their angular position relative to 12 o'clock (shown is stored 13, 26, 2, 33) (B) Storage gears are rotated until they are at 0, driving transfer gears (light grey) to the stored value. (C) Transfer gears are disengaged from storage and are engaged in the next stage. (D) Transfer gears are rotated until they are at 0, driving storage gears to the value. (stored 30, 13, 26, 2).*

### B. Modular exponentiation

Precise multiplication of two variables is historically a more difficult task for mechanical calculation. The simplest solution would be a Leibniz stepped reckoner wheel or a pinwheel to create a variable rotation (Figure 5) and perform repeated addition.[10]. Because we are operating in a finite field, no carry values are needed, which simplifies the operation. An alternative would include a transmission with a dynamic gear ratio set by one variable. This suffers the scale problem, as 37 different ratio settings are not an ideal solution.

Other functions are possible to implement mechanically. Linkages can be used to trace arbitrary curves.[11][12]. Cams of various profiles can be also used to create functions[13]. Examples are shown in Figure 8.

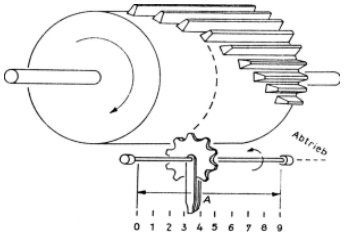


Fig. 5. *The Leibniz stepped reckoner wheel. The position of the reading gear (bottom) along the stepped drum (top) places it in the path of a variable number of teeth. A full rotation of the drum causes the reading gear to rotate a fixed amount.*

For the primitive polynomial equation 1, one iteration requires 10 multiplications, 10 doubling operations, and 14 sums. The calculation requires $log_2(37^4) \approx 20$ stages. At a total of 680 operations, this is feasible (if tedious) to do by hand if a mechanical implementation is impractical (or out of service). The number of operations per squaring step required scales non-linearly with an increase in length $k$.

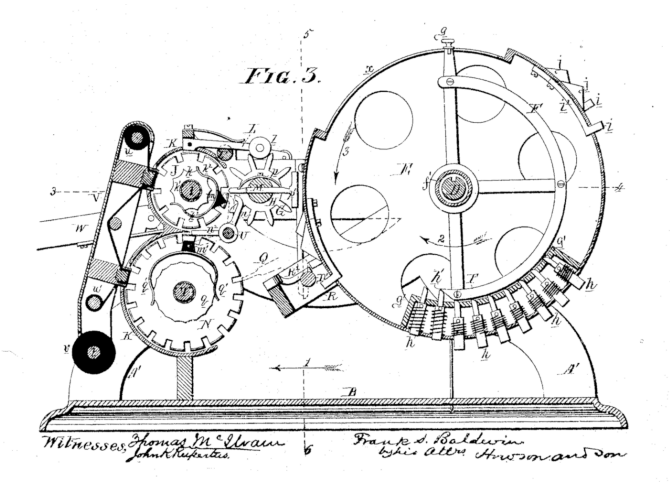


Fig. 6. *The pinwheel was a commonly used element in adding machines. The wheel has a set of spring loaded pins (bottom right of wheel) that can be extended, creating a variable number of teeth. [10]*

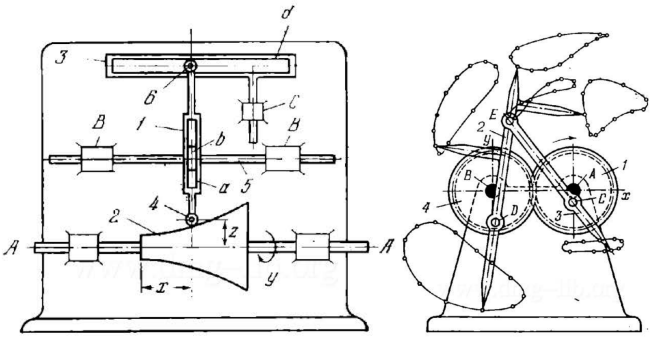

Fig. 7. *Generation of functions mechanically. Mechanism 3205 (left) implements a function of two variables, set by position of the wheel on the multi-dimensional cam. Mechanism 2479 (right) uses a set of linkages and gears to trace complex curves[13]*

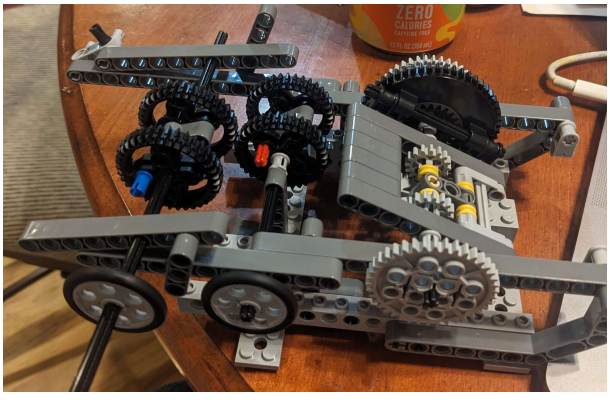

Fig. 8. *Terrible prototype demonstrating a differential (right) and a register (black wheels left).*

## VII. Analysis

Choice for $k$ in $\mathbb{F}_{37^k}$ When considering the finite field $\mathbb{F}_{37^k}$, the choice of $k$ defines the dimensionality of the vector space over the prime field $\mathbb{F}_{37}$. In the context of the Lorenz cipher, if we abstract the cipher's mechanics into a field theory framework, $k$ could hypothetically represent the complexity or length of the sequences manipulated by the cipher's mechanism. However, the original Lorenz cipher does not directly correspond to operations in a field like $\mathbb{F}_{37^k}$, as it primarily operated in $\mathbb{B}$. For a modern cryptographic system inspired by Lorenz that uses $\mathbb{F}_{37^k}$, selecting $k$ would depend on the required security level, where larger $k$ provides higher security but at increased computational cost.

Searching for such a polynomial is an undertaking in and of itself, but thankfully, the Sage language is made for tasks such as this. Similar to Singular, it can generate finite fields and gives a simple interface for doing so. The process needed to search for a irreducible polynomial and a generator is below in appendix A. In a contemporary implementation, enumerating every possible polynomial that fits our criteria is much easier, but thinking in terms of the era, one could come up with a systematic way to enumerate all choices by creating a sieve of sorts. If we assume the sieve has not been formulated, then a random search could be a better way to go if you are attempting to break possible encrypted communications. Given enough resources, one could come up with a list of all polynomials and have computers (human analysts) computing the properties of the polynomial to find those that are generators.

The requirements for the eStream program were 80 or 128-bit keys, usually with an equivalently sized IV. This would suggest a $k$ of 16 and 25 (rounding up). [6]. 1024 bits for DHM key exchange is considered breakable [14].

### A. Finding the Number of Primitive Polynomials

As a bit of an aside to the matter at hand, the question of the number of primitive polynomials and the ratio of these to the total number of polynomials in the space struck interest. Naturally, Euler's Totient function $\phi(n)$ is the go-to way of evaluating this. This function relates the set of integers less than $n$ that are relatively prime to $n$ in $\mathbb{Z}_n^*$. To determine the number of primitive polynomials of degree $k$ over a finite field $\mathbb{F}_q$, we leverage properties of the field's structure and the characteristics of its elements. A polynomial is primitive if and only if it is irreducible and its roots generate the multiplicative group of the extension field $\mathbb{F}_{q^k}$. The roots must, therefore, have a multiplicative order of $q^k - 1$, which is the size of this group minus the zero element. This group is displayed in Figure 9 and demonstrates the cyclic nature of the group by showing each row consists of a set with no repeating values, only permuted values.

### B. The Euler's Totient Function

The Euler's totient function, $\phi(n)$, is useful to our calculations. It counts the number of integers up to $n$ that are co-prime with $n$. In mathematical terms, it is defined as the size of the set of positive integers less than or equal to $n$ that are relatively prime to $n$.

*1) Calculation of Euler's Totient Function:* The calculation of Euler's totient function, $\phi(n)$, for $n = 37^4 - 1 = 1874160$, based on its prime factorization, is critical for determining the number of primitive polynomials over a finite field. The factorization of $n$ is $2^4 \cdot 3^2 \cdot 5 \cdot 19 \cdot 137$, and the totient function is calculated as follows:

$$\begin{aligned}
&\phi(37^4 - 1) = \\
&= \left(37^4 - 1\right)\left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{3}\right)\dots \\
&\left(1 - \frac{1}{5}\right)\left(1 - \frac{1}{19}\right)\left(1 - \frac{1}{137}\right) \\
&= \left(37^4 - 1\right) \cdot \frac{1}{2} \cdot \frac{2}{3} \cdot \frac{4}{5} \cdot \frac{18}{19} \cdot \frac{136}{137} \\
&= 470016.
\end{aligned}$$

This result demonstrates that there are $470016$ integers up to $\left(37^4 - 1\right)$ that are co-prime with $\left(37^4 - 1\right)$. The number of primitive polynomials of degree 4 over $\mathbb{F}_{37}$ can then be calculated as:

$$\begin{aligned}
|S_{\text{Poly}}| &= \frac{\phi(\left(37^4 - 1\right))}{4} \\
&= \frac{470016}{4} \\
&= 117504 \\
&= \left(\frac{117504}{(37^4 - 1)}\right) \cdot 100\% \\
&= 6.27\%.
\end{aligned}$$

This approach allows us to determine how many of the polynomials of a given degree can generate the multiplicative group of the corresponding extension field, which is crucial for their application in our chosen problem.

### C. Probability Calculation for Finding a Primitive Polynomial

Interpreting this as a Bernoulli trial, given the probability $p$ of finding a primitive polynomial on a single trial is $6.27\%$ if sampling the space in a uniformly random fashion, the probability of not finding a primitive polynomial in one trial is $q = 1 - p$. Interpreting the trials as success or failure gives a clear way to arrive at an arbitrarily high likelihood of success. The calculation involves determining the number of trials $n$ required to achieve at least a 99% chance of success at least once. The detailed steps are as follows:

$$
\begin{aligned}
&\text{(Probability of success on a single trial)} \\
&p = 0.0627 \\
&\text{(Probability of failure on a single trial)} \\
&q = 1 - p = 0.9373 \\
&\text{(Desired success probability across trials)} \\
&\text{Desired Probability} = 0.99 \\
&\text{(Probability of failure in all } n \text{ trials)} \\
&q^n = 1 - \text{Desired Probability} \\
&n = \frac{\log(1 - \text{Desired Probability})}{\log(q)} \\
&\text{(Solving for } n \text{ using logarithms)} \\
&n = \frac{\log(1 - 0.99)}{\log(0.9373)} \\
&n \approx 71.12 \\
&\text{(Calculated number of trials needed)}
\end{aligned}
$$

To ensure at least a 99% probability of finding a primitive polynomial at least once, the number of trials required is given by rounding up the calculated value:

$$n = \lceil 71.12 \rceil = 72$$

Thus, approximately 72 trials are needed to reach the desired probability threshold. This will be used in the coming algorithm for finding the primitive poly, which can be found in appendix A. Now that we have a polynomial for our implementation, we can use the following as a target $P(x)$ modulus. Assuming the allocation of near-unlimited resources towards solving this problem before constructing a Colossus-like machine, we can expect the adversary to put at least a hundred analysts to work on this day in and day out, ruling out polynomials as they went along. Of the information available, this seems highly plausible as Bletchley Park employed nearly 10,000 people by the end of the war [15].

$$p(x) = 33x^4 + 33x^3 + 25x^2 + 21x + 2 \tag{6}$$

Using the Sage language, validating this process for any base Field $\mathbb{F}_q$ is possible. The steps needed to verify that a given polynomial would be considered a generator are in appendix A. Since we would end up with $37^k - 1$ printouts, we will skip that part here and encourage the reader to look at the algorithm and the included files for the Sagemath generation code if they want to verify this claim.

### D. The inverse

Using the Cayley table in Fig. 9, we can formulate the inverse of any element by finding the location where any of the rows or columns contain a 1. which in this diagram corresponds to the darkest blue. Using a stepped reckoner system as shown in Fig. 5, we could multiply and divide using long-form operations to obtain our solution, but performing the multiply in the fashion discussed in VI, we can achieve the multiplication of an inverse.

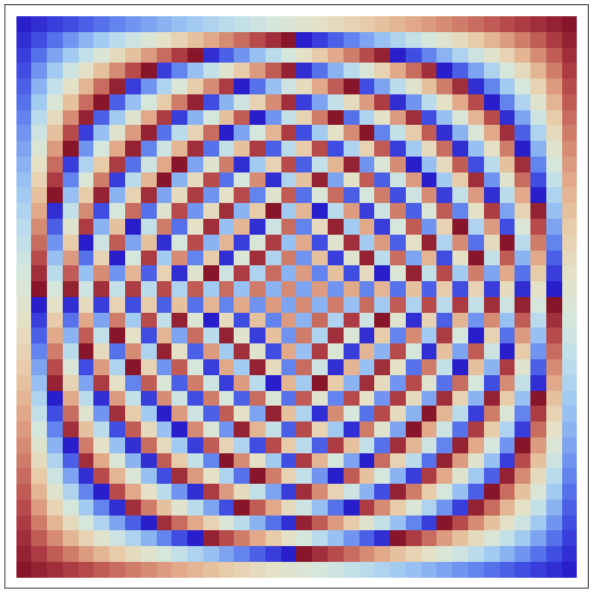

Fig. 9. *The multiplicative Cayley table colored for simplicity*

*1) Scaling Over Compute Capacity Through History to Now:* The original Colossus machines were able to process up to 5,000 characters per second, an impressive feat for the time. Today, modern processors can handle billions of operations per second, illustrating a massive scale-up in computational capacity. For example, using modern GPUs, the entire search space for the Lorenz cipher could potentially be explored in seconds or minutes, a task that took hours or days during World War II.

*2) Examples from Bletchley Park Breaking and Defeat Mechanisms:* At Bletchley Park, the breaking of the Lorenz cipher involved statistical analysis to detect patterns in the ciphertext, exploiting weaknesses in the rotor settings' predictability. Today, such ciphers would be defeated using even more sophisticated statistical and machine learning algorithms, which could identify subtle patterns and correlations far beyond human capability.

### E. Weaknesses

*1) LFSRs on Their Own vs. In More Complicated Combinations:* Linear Feedback Shift Registers (LFSRs) are vulnerable to attacks such as the Berlekamp-Massey algorithm, which can efficiently determine the LFSR's parameters if enough output bits are known. In the Lorenz cipher, the complexity was increased by combining multiple LFSRs (Chi and Psi wheels), which made direct attacks more challenging but not impossible, as demonstrated by the success of Colossus.

*2) Key Distribution Scale:* The practical deployment of any cipher involves securely distributing and managing keys. The Lorenz cipher required the secure initial setting of wheels at both ends of a communication link. In modern settings, key distribution for highly secure systems often utilizes public key infrastructures, which solve many problems inherent in the distribution of symmetric keys over insecure channels.

*3) Possible Traffic Attack:* The repeated use of key settings in the Lorenz cipher could potentially lead to vulnerabilities, especially if the same key settings are used across multiple messages—a form of depth attacks. While no specific instances of such attacks during WWII are documented, today's ciphers mitigate such risks using techniques like nonce and initialization vectors to ensure each message is encrypted uniquely.

### *F. Number Theory and Bounds of Computation on the Colossus Machine*

The original Colossus was designed for specific tasks rather than general-purpose computation, focusing on Boolean logic and counting operations suitable for cryptanalysis. Its computational model did not directly involve number-theoretic operations like modular exponentiation, which are common in modern public-key cryptography.

*1) Exact Number of Operations for Modular Exponentiation:* To calculate the exact number of operations for modular exponentiation, commonly used in public-key cryptography (e.g., RSA, Diffie-Hellman), we often refer to algorithms like square-and-multiply, which are efficient for such purposes. For a base $a$, exponent $b$, and modulus $m$, the number of operations primarily depends on the bit length of $b$.

*2) Infeasibility of Key Exchange by Hand in the Field:* Traditional hand methods for secure communication, such as those used during WWII, are impractical with modern public-key schemes due to the computational complexity and error-prone nature of manual calculations. In $\mathbb{F}_{37^k}$, exchanging keys using something like Diffie-Hellman would involve operations that are infeasible to perform accurately without electronic assistance.

*3) Choice for the Prime Polynomial Generator in $F_{37^k}$:* The choice of a prime polynomial generator in a field like $\mathbb{F}_{37^k}$ depends on the desired properties of the field, such as having no small subfields other than the prime field itself, which helps in achieving good cryptographic properties like maximum period length for LFSRs or robustness against linear algebra attacks. The selection process would typically look for irreducible polynomials that fit these criteria. Primitive polynomial selection has impacts on the number of operations and the complexity of implementation.

For polynomials over $\mathbb{F}_{37^4}$, we evaluated all possible three term primitive polynomials for use in the Mastrovito squarer (see section V). We looked for specific characteristics - the squarer should have few coefficients, the coefficients should be small, and the coefficients should be powers of two. $x^4 + x^2 + 2$ and $x^4 + 36x^2 + 2$ were found to have the best characteristics, all coefficients terms being a 1,2, or 4. The Mastrovito equations differ only in sign, due to 36 being the multiplicative inverse of 1. $x^4 + 22x^3 + 13$ was the worst, with coefficients being large, and often prime numbers like 13 and 17 and having more terms. The general pattern indicates that the primitive polynomial should have small coefficients, particularly avoiding prime numbers.

## VIII. Conclusion

We have a start here, needs more stuff.

TODO list.

1) history citations.
2) detail analysis of lorenz mathematical function
3) key authentication math.
4) working mechanical implementation
5) finish the hello world appendix
6) more analysis of Trivium, Rabbit, MICKEY, and Grain, applications to non-linear

CHES journal takes 20 page papers, due date July. https://ches.iacr.org/2024/cfp-ches2024.pdf

## Appendix
## Silly problem

Listing 1. FindIt Algorithm for GF(37)

```
import galois
alphabet = "0123456789abcdefghijklmnopqrstuvwxyz␣"
encode = { k:i for i,k in enumerate(alphabet) }

def findit_37(s):
    GF = galois.GF(37)
    text = [encode[i] for i in s]
    y = GF(text)
    return galois.berlekamp_massey(y,
           output="galois")
```

Listing 2. FindIt Algorithm for GF(37)

```
findit_37("hello␣world")
>Galois LFSR:
 field: GF(37)
 feedback_poly: 32x^6 + 19x^5 + 28x^4 + x^3 + 8x + 1
 char_poly: x^6 + 8x^5 + x^3 + 28x^2 + 19x + 32
 taps: [ 5 18  9 36  0 29]
 order: 6
 state: [ 2 16 21 22  2 17]
 initial_state: [ 2 16 21 22  2 17]
```

exists taps $g_0...g_{n-1} \in \{0, 1\}$
and initial vector $b_0...b_n \in \{0, ..., k\}$
such that $[a_{n_{t_0}}, ..., a_{n_{t_1}1}]$ = 'hello world!' The polynomial $\mathbb{F}_{k^{12}}$ at time $t$ is $a_{n_t} * x^n ... a_{0_t} * x^0$. Individual coefficient $a_m$ at time $t$ is $a_m = g_m * a_n + a_{m-1}$ The required conditions are: $a_{n_t} = g_{n-1} * a_{n_{t_{-1}}} + a_{n-1_{t-1}}$

This gives a polynomial of length $k$, but this is worst case in application of the Berlekamp-Massey algorithm, which will find the minimum LFSR for a given sequence [16].

## APPENDIX
## ALGORITHMS

**Algorithm 2** Find a Primitive Polynomial

**Require:** Degree $k > 0$
**Ensure:** Returns a primitive polynomial of degree $k$ or indicates failure after N attempts.
  **Import** PolynomialRing, FiniteField from SageMath libraries
  **Define** the field $F_{37}$ and polynomial ring $P \leftarrow x$ in $F_{37}$
  $R \leftarrow x$ in $F_{37}$
  Initialize $attempts \leftarrow 0$
  **while** $attempts < N$ **do**
    $poly \leftarrow$ random element of degree $k$ from $R$
    **Print** "testing poly: $poly$"
    **if** $poly$ is irreducible **then**
      **Print** "—Is irreducible"
      Try to construct the extension field $F_{37^k}$ with $poly$ as $alpha$
      $E \leftarrow F_{37}.extension(poly, 'alpha')$ if possible else
      **Print** "Failed to construct field with polynomial $poly$"
      **if** Extension field $E$ is successfully constructed **then**
        $alpha \leftarrow E.gen()$
        Generate set of elements $elements \leftarrow \{\alpha^i \mid i = 1 \text{ to } 37^k - 1\}$
        **if** $|elements| = 37^k - 1$ **then**
          **Print** "—Contains correct number of generated elements."
          **Print** "Primitive polynomial found: $poly$"
          **return** $poly$
        **else**
          **Print** "Polynomial $poly$ is irreducible but not primitive."
        **end if**
      **end if**
    **end if**
    $attempts \leftarrow attempts + 1$
    **Print** "—Not irreducible."
  **end while**
  **return** **None**
  **Print** "No primitive polynomial found after N attempts."

*Note*: The N mentioned below would need to be greater than 72 given our probability analysis.*

**Algorithm 3** Verify Distinct Elements in a Finite Field Extension

**Require:** *None*
**Ensure:** *Prints whether all elements generated are distinct*
  **Import** PolynomialRing, FiniteField from SageMath libraries
  **Define** the base field $F_{37}$
  **Define** the polynomial ring $P \leftarrow x$ in $F_{37}$
  **Set** primitive polynomial: $p(x) = 33x^4 + 33x^3 + 25x^2 + 21x + 2$
  **Construct** the extension field $F_{37^k}$ using $p(x)$
  **Initialize** generator $\alpha$ of $F_{37^k}$
  **Set** degree $k \leftarrow 37^4$
  **Initialize** list of elements $elements$
  **for** $i = 0$ **to** $d - 1$ **do**
    Compute $\alpha^i$ and append to $elements$
  **end for**
  **Output** the elements to verify they are distinct
  Calculate the number of unique elements in $elements$
  **Print** number of unique and total elements
  **if** number of unique elements == total elements **then**
    **Print** "All elements are distinct."
  **else**
    **Print** "There are repeated elements, indicating a problem."
  **end if**

## REFERENCES


[1] J. D. Paul, "In the twilight of electromechanical encryption, an exceptional machine figured in a major spy scandal," *IEEE Spectrum*, vol. 58, no. 9, pp. 32–52, 2021.
[2] J. Good, D. Michie, and G. Timms, "General report on Tunny with an emphasis on statistical methods," UK Public Record Office HW 25/4 and HW 25/5, 2010.
[3] B. Wells, *Advances in i/o, speedup, and universality on colossus, an unconventional computer*, 1970.
[4] C. E. Shannon, "Communication theory of secrecy systems," *The Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.
[5] R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications*, Rev. ed. Cambridge ; New York: Cambridge University Press, 1994, 416 pp.
[6] M. Robshaw and O. Billet, Eds., *New Stream Cipher Designs: The eSTREAM Finalists*, red. by D. Hutchison, T. Kanade, *et al.*, vol. 4986, Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.
[7] D. J. Bernstein, "Which eSTREAM ciphers have been broken?," 2008.
[8] T. Siegenthaler, "Correlation-immunity of nonlinear combining functions for cryptographic applications (corresp.)," *IEEE Transactions on Information Theory*, vol. 30, no. 5, pp. 776–780, 1984.
[9] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.
[10] "Improvement in calculating-machines," U.S. Patent 159244A, 1875.
[11] A. B. Kempe, "On a general method of describing plane curves of the n th degree by linkwork," *Proceedings of the London Mathematical Society*, vol. s1-7, no. 1, pp. 213–216, 1875.
[12] M. Kapovich and J. J. Millson, "Universality theorems for con guration spaces of planar linkages," 2002.
[13] I. Artobolevsky, *Mechanisms in modern engineering design*, 5 vols. Mir Publishers Moscow, 1976, vol. 1.
[14] D. Adrian, K. Bhargavan, *et al.*, "Imperfect forward secrecy: How diffie-hellman fails in practice," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, Denver Colorado USA: ACM, 2015, pp. 5–17.
[15] D. Turing, *The Codebreakers of Bletchley Park: The Secret Intelligence Station that Helped Defeat the Nazis*. Arcturus Publishing, 2020.
[16] L. Massey and I. {INTRDuCTIN}, "Shift-register synthesis and BCH decoding l," 1969.