[Albert Carlson, Benjamin Williams et al. (May 28, 2025). Basics of Polymorphism in Encryption. 2025 IEEE World AI IoT Congress (AIIoT), IEEE Xplore. Source: https://ieeexplore.ieee.org/document/11105375]

Basics of Polymorphism in Encryption

Albert Carlson* **Benjamin Williams**†, and **Mandeep Singh**‡,

* Computer Science Department, National University, La Jolla, CA, USA.
†Computer Science Department, University of Idaho, Moscow, ID, USA.
*Email: *acarlson2@nu.edu, †will9847@vandals.uidaho.edu, ‡*

***Abstract*—Polymorphic encryption algorithms have been known and received limited use since the 1880s. Although used by the USSR for a period ranging from the late 1930s to the mid-1980s, polymorphic, or "mutating" ciphers are relatively unknown outside of those who have previously used this type of encryption. Often confused with homomorphic encryption, polymorphic encryption is easily identified and has recently been shown to be very effective in securing messages. This paper presents an introduction into polymorphic ciphers and how to recognize them, as well as what is needed to implement a polymorphic encryption engine and polymorphic RNG.**

Keywords: Polymorphic encryption, polymorphic RNGs, polymorphism, homomorphic encryption, Geffe generators key progressions, encryption, and RNGs.

I. Introduction

A. History

Since the advent of hiding information through encryption more than 3000 years ago [1], encryption has been in an arms race with those who attempt to break encryptions, reveal the message, and make use of that message. Advances in obscuring are met with new decryption techniques, which then require new ciphers, followed by the requirement for new decryption methods. That cycle is unlikely to change in the future, given the introduction of quantum computers and the advent of the post-quantum environment (PQE). Quantum computers (QCs) that can break Public Key (asynchronous) Encryptions (PKE) are just the latest development in that arms race on the side of breaking the newest encryptions to combat PKE. However, QCs cannot change the rules of mathematics, and one of the older ciphers that was previously impractical has been shown to be a possible solution. That cipher family is the polymorphic or morphing cipher, also known as the one-time pad (OTP).

Recent research [2] into polymorphic ciphers has shown that it is possible to reduce the costs and complexity of that class of ciphers. Polymorphic ciphers are more correctly categorized as applications of a polymorphic engine (PME) [3] that create polymorphic cipher text (CT) streams. Having seen very limited use in the last 150 years the reintroduction of this type of cipher requires educating cryptographers and cybersecurity users on the characteristics and use of PMEs and polymorphic ciphers. That education begins with and understanding of polymorphism and how it is applied to information theory (IT) [4].

B. Paper Outline

The paper is organized into four sections. The first section frames the subject of the paper. Section II, the Background, gives needed information for the reader to understand the concepts presented. Definitions of homomorphic ciphers and polymorphic ciphers are of special importance and interest. The concepts of sharding, Geffe generators and polymorphic RNGs are also discussed. Section III, Analysis, addresses how polymorphism increases entropy, how keys relate to time and key applications, as well as polymorphism can be extended to random number generators. In Section IV, Conclusions and Future Work, summarizes the paper and suggests how polymorphism can be extended to make ciphers more secure.

II. Background

A. Homomorphic Encryption

Homomorphic encryption [5] is a type of encryption that allows math functions to be performed on the encrypted content of a message without having

to decrypt the message, do the math, and then re-encrypt the results. This encryption is possible since the homomorphic cipher is a linear cipher [6] with the property that

$$E(a) \circ E(b) = E(a \circ b) \tag{1}$$

where $\circ$ represents some mathematical function. Homomorphic ciphers have the advantage of being available to all users without having to expose data when working with the information.

At their core, polymorphic and homomorphic ciphers are fundamentally different from each other, although a polymorphic cipher can employ homomorphic ciphers, but homomorphic ciphers cannot be polymorphic. The confusion between the two is the result of lack of familiarity with both homomorphic and polymorphic ciphers and the use of the term "morphic" in both names, meaning "having the form" [7].

B. Polymorphic Encryption

Polymorphism is defined as the ability of something to exist in several different forms [8]. Biological examples are seen in a species having different markings and coloration [9], but the same effect also occurs in technology. Specifically, polymorphism also occurs in computer science. The most familiar use of polymorphism occurs with computer viruses [10]. Many virus scanners work on the principle of signature detection for the virus code. The instructions that comprise a virus have instructions that, when taken together, create a unique signature that identifies the virus inside any program it has infected. To escape detection some viruses add an extra block that decrypts an encrypted virus in the program when the infected program is run. A scanner sees the encrypted signature in the program and it does not match the known instruction signature of the virus so the infection is not reported.

Computer science has traditionally defined two basic types of polymorphism [11]:

- ad hoc polymorphism - This is the use of symbols that are valid, but have several different uses, depending on the context in which the symbol is found. For example, the '+' symbol denotes addition when used with numbers or equations, but also denotes concatenation of strings (or other data structures). There is no way to determine the meaning of the symbol without knowing the types of inputs associated with the function represented by the symbol.
- parametric polymorphism - An instantiation of a function that works no matter what type of inputs are give to it. Parametric polymorphism allows for the same results given a generic call to a function. The body of the function can change in form, but the result of the function is identical in spite of the differences in the method of achieving the computational result

A second use of polymorphism is to disguise code to keep it safe or to differentiate it from a competitor's protected code. The many ways to code the same solution allow for a transfinite ($\aleph_0$) number of ways to represent the same function. Some of the solutions are very close to each other, differing only in variable names, while others implement the same actions with varying levels of efficiency and compactness. This type of polymorphism is static and does not change over use.

Encryption of messages also uses polymorphism. In this case, polymorphism refers to what is known as a "morphing" cipher - a cipher that changes its cipher and key pair at frequent, but irregular, intervals. This type of cipher was first pioneered in 1882 by Frank Miller [12], but the application was not widely accepted at the time. The technique was again revived during World War I when the same method was patented by Vernam in 1917 [13] and became known as the OTP. The OTP follows the principles set out by both Kerckhoffs [14] and Shannon [15]. Shannon proved that the OTP is the only "mathematically secure" cipher type. However, the cost of implementing the OTP for a single character proved to be so high that the OTP was impractical [16]. Later, this form of cipher was shown to be extensible to larger blocks, reducing the cost and allowing for practical implementations [2].

In this paper, the focus will be on the definition of and characteristics of polymorphism related to encryption. The polymorphism used in encryption refers to how the blocks are encrypted using many forms of obscuring - using the same function with different inputs for the same obscuring function. It is meant to leverage entropy to change the unicity distance [15] so that the message cannot be reliably

and verifiably decrypted by an attacker.

Components of polymorphic encryption (see Figure 1) are

1) A library of encryption algorithms - Morphing between different ciphers requires having multiple encryption algorithms on hand for use. These algorithms should be evaluated to be "hard" and be as strong as possible. A large library is also desirable so that attackers have to attempt decryption from a larger choice of algorithms. Libraries also allow for an easy way to customize algorithms by adding newer, stronger algorithms as they become available and remove those ciphers that are broken.
2) An RNG Library - Polymorphic RNGs (polyRNGs) require a library of RNGs to allow for the implementation of long-cycle RNGs that approximate true RNGs (TRNGs). Attacking OTPs, the prototypical polymorphic cipher, can be facilitated by attacking the underlying RNG [16]. Therefore, the highest quality RNG is needed.
3) A TTL counter/timer - Frequent, but irregular, changes for encryption algorithms and RNGs require being able to count the number of accesses to the associated algorithms. The counter tells how long the selection of library component is valid to use, known as the Time to Live (TTL) counter. When the counter reaches its terminal count for an item, a new library component is selected and used in the encryption process.
4) The function block - The specific implementation of the polymorphic function is the heart of the circuit or software. In this case, the polymorphic system is being applied to encryption. All encryption algorithms require a secret key [14]. This key is a randomization seed that selects the mapping of plain text data to cipher text data. The key generator takes the input and produces a key that is based on that input seed. There are methods for decoupling the key from the RNG output by using constructs such as physically unclonable functions (PUFs) [17]. This block provides that functional output.

Polymorphism has several characteristics that are

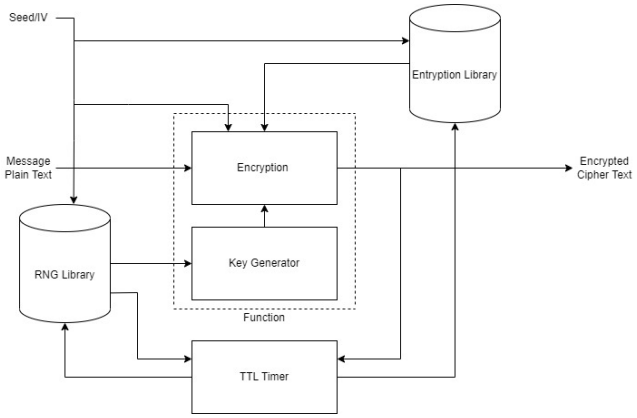

Fig. 1. Polymorphic Encryption Block Diagram

shared by the applications that employ it. The first is a morphing action. One, or more, settings is changed at frequent, but irregular intervals. Changing the settings is done to add entropy into the system and prevent information from accumulating to the point where an attacker can discover the value used for that setting. Shannon called this value the "unicity distance" (n) [15] for the information. If the setting is changed (morphed) before that number of characters is reached, then the attacker cannot unambiguously get the correct value for the setting. This number resets with each change of value, and remains constant for a period

$$TTL < n_{setting} \tag{2}$$

While Shannon applied this measure as a global measure, Carlson, et al [18] showed that it applies to any sub-message and termed it "local" entropy (n_l). When a polymorphic function has more than one setting that should be changed to increase entropy, each setting should have its own TTL and be changed independently of the remaining settings. Each setting is a secret key for the function. If the keys are independent of each other, then each must be solved by an attacker to arrive at the correct answer. Taken together, the independent keys can be concatenated and treated as a composite key. Such a key can be constructed as

$$K = +_{i=1}^{x} k_i \tag{3}$$

for each setting key, where + represents concatenation and k_i are the different setting keys. Every time one, or more, settings change, the composite

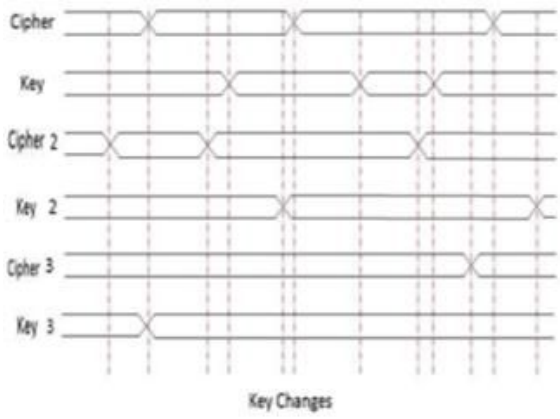

Fig. 2. Time Domain Multiplexing

key changes. Due to each setting having its own TTL and value, the composite key can change very rapidly, as seen in Figure 2. TTLs that are relatively long typically align in such a way that the key changes much more rapidly and decreases the amount of time a composite key is valid. There is no reason why multiple settings cannot change simultaneously, but if settings are controlled by different keys this is less likely.

Consider a pane of glass that represents a message to be encoded. The best way to set the TTLs for settings is to select each randomly. This can be visualized by taking a rock and throwing it through the pane of glass. Each resulting piece of glass would receive its own unique composite key. Each change in settings/key defines the piece of glass and is called a "shard." Sharding is characteristic of a polymorphic encryption function. It is governed by a second pseudorandom function known as the "time to live," or TTL. The TTL specifies how long the shard will be active, or valid. This results in a TDM sharding assignment that contains less than the required number of symbols needed for decryption.

Sharding allows the message to be broken into a group of blocks to be encrypted. The number of blocks (s_b) should be

$$1 \leq s_b \leq |M| \tag{4}$$

where $|M|$ is the size of the message (in characters). However, the most efficient number of shards takes place when the unicity distance for the shard (n_s) is related to the number of shards

$$|s_i| < n_s \tag{5}$$

where n_s is the local unicity distance for that shard. Ideally, the shard size for each shard will vary so that an additional degree of freedom and increased entropy is added to the decryption problem. Each shard will also employ a different block size for encryption. This constraint for the block size ($|b|$) inside the shard is

$$|b| = \{|s_i| \ \% \ |b| = 0\} \tag{6}$$

Solving for the block size requires that the attacker be able to correctly find the size of the shard and then select between the possible block sizes for that shard size. Security is enhanced if the shard size is not prime and has many possible factors. Shard sizes should be selected using the RNG for the polymorphic function using the constraints given for block sizes. The probability of the attacker guessing correctly is very small and a brute-force approach to getting the right shard boundaries grows exponentially with the size of the original message.

C. Polymorphic RNGs

Random number generators (RNGs) can also be designed to be polymorphic. Traditional RNGs are functions that produce a sequence of numbers that appear to be unrelated to each other, but are actually calculated from the preceding number in the sequence. If the RNG is a function represented by $f(x)$ then the sequence of numbers is related to each other as

$$f(x_{i+1}) = f(f(x_i)) \tag{7}$$

This results in a repeating cycle of numbers with a period of λ. By specifying a number in the cycle, called the "seed" any number of authorized users can synchronize themselves and follow the values of the RNG in order. If the value for $\lambda \approx |2^b|$, where b is the number of bits in the random number, the RNG is said to be "maximal" [19]. There are a large number of RNGs comprised of different mathematical functions [20] with varying quality.

Even though λ may be very large, RNGs suffer from several disadvantages. First, RNGs cannot be

truly random. Any RNG implemented on a deterministic computer cannot be random [21]. TRNGs are the most desirable RNGs, but TRNGs must use natural processes that must be observed by all of the authorized users or have the sequence delivered by couriers to maintain that security.

The second issue is that the RNGs can be attacked. Schneier [22] pointed out that knowing the RNG and its state can lead to successfully breaking encryption and is a security issue. This approach was used in the Venona attack [16] very effectively against the Soviet Union for several decades. The RNG attack is the only known vulnerability against the One Time Pad [23]. Similar to the unicity distance in encryption [15], there is also a unicity distance for RNGs, denoted as ρ. For most ciphers, ρ is relatively low - as low as 624 access for the Mersenne Twister [20], [24], and in the single digits for some RNGs.

RNGs can be made much stronger by applying the principles of polymorphism. One such polyRNG is the Geffe generator [25] (see Figure 3). Multiple RNGs are used in parallel to feed a multiplexer. Using a multiplexer allows choosing between the different RNGs without having to constantly set up and synchronize the RNG when changes occur. The number of RNGs is limited only by the ability of the multiplexer (MUX). The particular stream routed to the output is controlled by an RNG connected to the selector inputs for the MUX.

The RNGs selected for use, the number of RNGs used by the MUX, the initial seed for the RNGs, and the time that each RNGS/seed is changed is determined by the TTL unit and come from the library of available RNGs agreed upon by the authorized users. This library can be updated as needed and desired.

The requirements for the polymorphism in a polyRNG are similar to that of a polymorphic encryption. A library of different RNGs is needed, along with a TTL unit for selection of the RNGs, and a multiplexer is also required. Using the architecture shown in Figure 3, a very long, if not arbitrarily long λ RNG can be constructed. The RNG source can be selected to supply a single number before being reassigned, or can be used as a shard of data from a single RNG before proceeding to the next source. The RNGs feeding the MUX can

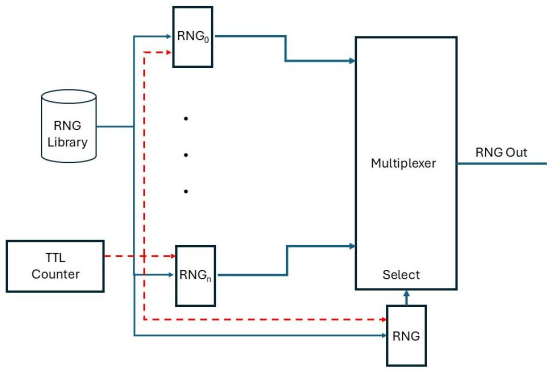

Fig. 3. Geffe Generator Style Polymorphic RNG

be incremented separately, as arbitrary groups, or at the same time, and groupings can be changed on the fly. Replacing RNGs can also be forced when their ρ value is reached. Further, the selected RNG may be changed as well to reduce the chance of an attacker patterning the polyRNG.

PolyRNGs have an interesting feature that is not found in RNGs. Because RNGs cycle regularly and are deterministic, numbers in the "random" output repeat only when the cycle is complete. Seeing a number previously observed in the RNG sequence indicates that the cycle is repeating. However, numbers can repeat in the polyRNG without the polyRNG cycling due to the different RNGs in parallel feeding the MUX.

III. Analysis

Polymorphic systems, such as encryption, have been shown to be susceptible to attacks on the RNG block. This type of attack centers around the TTL block which tracks the need to change the cipher/key pair so that an attacker can either stop changes and allow enough information to accumulate for analysis or so that the attacker can track and predict the output of the RNG.

Kelsey, et al [26] suggested that an attack against the RNG could be effective in breaking the security of a cipher. As an example of such a vulnerability, consider the OTP. Knowing that the RNG cycles, if the user does not use a true RNG, then it is possible to break the OTP [16]. While the attack that was used by the US government has not been revealed, Carlson, et al [27] showed a possible version of the attack that can work.

Frustrating the Venona attack requires that the underlying RNG be truly random (TRNG). However, computing machines are not capable of producing a TRNG stream [21]. No deterministic machine can produce true randomness. Consider the observation of Knuth in "The Art of Computer Science," volume 2, regarding randomness [28]. Knuth stated that it is impossible to know if some random number sequence is truly random. For a string of size $|S|$ it is only possible to show that the sequence seq of size

$$|seq| = \frac{|S-1|}{2} \tag{8}$$

is random. Equivalently, this suggests that if a PRNG has a cycle (λ) whose size is much larger than the required number of accesses by the program

$$\lambda >> |M| \tag{9}$$

where $|M|$ is the size of the message, that the RNG output stream may be seen as being random [29]. Therefore, with the correct, long PRNG polymorphism can be safely maintained.

A second vulnerability for polymorphic systems is the loss of synchronization between the message and the underlying RNG that provides the TTLs and cipher/key pairs. If a portion of the message is lost but the corresponding TTL does not reflect that loss of synchronization then the mismatched portions of the message will similarly be incorrectly decrypted and may also be lost. Any other reason for loss of synchronization, such as corruption of the RNG stream/units, between the TTL and CT will result in a similar outcome.

One of the main advantages of polymorphism is that the increased security does not come at the cost of increased overhead and latency in most computers. Each shard is independent of all of the other shards and is said to be an "orthogonal" problem. Consider the AES cipher with the CBC mode. This cipher is thought to be one of the most secure cipher algorithms in use. The encryption requires an input of the cipher text (CT) of the preceding block to calculate the CT of the next block. Encryption is linear and must be done one block of plain text (PT) at a time. In polymorphic encryption, shards use the same cipher and key, different from the preceding or following shards. Therefore the shard has an initialization vector (IV) and can be calculated as if the shard were its own message (a sub-message in the original message). Therefore, shards can be processed by different threads [30] or using different cores/GPUs [31], [32] on a computer. Each shard is assigned its own thread/core/GPU which does the encryption (or decryption) for that shard. As shards are completed new shards are assigned to the open resource until the full message is completed. There is some overhead in splitting the shards, assigning them to the open resource, and then reassembling the message. Overhead and latency are subsumed by the parallel treatment of the data. The speed up (t_r) is approximately

$$t_r \approx \frac{1}{2^{(n-1)}} \tag{10}$$

due to latency and overhead. The variable n is the number of threads or cores used in the machine assigned to the task. This means that it is possible to process the polymorphic encryption faster than the less complex version of the algorithm.

While faster, polymorphic encryptions are more complex than single cipher/key pair ciphers. Using polymorphism requires more effort and time to design, implement, and develop. Once proper development and testing take place, these disadvantages become moot.

Polymorphic encryption vastly increases the amount of entropy in the encryption. By frequently and irregularly changing the cipher/key pairs by using a random selection of both cipher algorithms and keys the key space is greatly enhanced and the probability of breaking even one shard is hugely reduced. Each shard must be correctly identified and then treated as a set of orthogonal problems in the encryption of the message.

Using a library of encryption algorithms also increases the entropy of the problem. The practice also allows for customizing the ciphers employed in the process and means that algorithms with significant vulnerabilities can be removed as the vulnerabilities are identified. Newer, stronger algorithms can also be added to the library when developed. Customized libraries composed of strong ciphers help manage security. Weaker ciphers are used in a way that increases their strength by using them for a period that is much shorter than their unicity distance.

IV. Conclusion

Cryptographers are looking for new methods to increase the security of their communications. Among those methods is the concept of polymorphism in both encryption and in RNGs. Polymorphism is used in security as well as in the representation of data, implementation of functions in programming, and as a way to select the proper portion or implementation of a program at run time. This paper focuses on the use of polymorphism in encryption and security.

There is some confusion by users between polymorphic and homomorphic methods. Polymorphism uses the concept of sharding to increase the entropy in the encryption and reduce the patterns in the cipher text. Homomorphism is a completely different approach to encryption. That approach is a linear cipher that allows for changes to the plain text without having to decrypt the plain text to make the change. Since data can be altered and adjusted without having to reveal previously encrypted data, eliminating the need for revealing the data increases the security of the information being protected. It is not, however, closely related to polymorphic encryption and is not relevant to the discussion.

Polymorphic encryption and polyRNGs are definitely related. PolyRNGs are a form of encryption, or "obscuring." Both have the same basic blocks used to build the functions. These blocks include: libraries of various functions (encryption algorithms or RNGs), a TTL block to count the accesses and assign TTL timing to ensure that the blocks the service to not exceed their unicity distance, and some method for choosing between outputs (a multiplexer). There are many possible architectures combining these blocks, especially if combining both functions for a cryptographic system.

Polymorphic systems are more complex than simple cryptographic systems and RNGs. This implies an increased cost and expenditure of resources. However, the overhead and latency of the increased computation can be subsumed due to the orthogonal nature of polymorphism. Polymorphic systems are actually faster than linear systems because of that orthogonality. They will still require additional memory, threads, and/or cores to achieve that additional speed. Polymorphic systems increase the entropy and are much more secure than the single encryption and RNG components. Both increase the security of information protected using this methodology.

Polymorphic encryption and polyRNGs are relatively new and not yet well studied. Future work requires a continued study of RNGs and the effects of combining those RNGs. All of the RNGs need to be characterized for their λ values, which are also dependent on initial seeds. Lists of the seeds and their λ values are needed. More attention should also be given to the n distances to properly assign the TTLs for various RNG algorithms. Because polyRNGs are dependent on their underlying constituent RNGs, research into strong RNGs is always important.

Polymorphic encryptions also depend on their underlying encryption algorithms. Therefore, research into strong encryption algorithms continues to be important in the future. In the future, polymorphic encryption needs to be extended to varying block sizes in the shards and requires additional investigation. Of particular interest is the subject of local entropy and unicity distance, as well as better characterization of the statistics of natural languages, especially with respect to metacharacters and metalanguages. Patterns of metacharacters (blocks) become very important due to the security associated with metacharacter size.

References

[1] David Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, 1996.

[2] Albert Carlson, Indira K. Dutta, Bhaskar Ghosh, and Michael Totaro. Modeling polymorphic ciphers. *Sixth International Conference on Fog and Mobile Edge Computing (FMEC)*.

[3] Albert Carlson and Robert Le Blanc. Polymorphic encryption engine, 2015.

[4] Thomas Cover and Joy Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc, New York, 2nd edition, 2005.

[5] Ansumana Jadama, Arbaaz Mohammed, and Farrukh Rashid. Fully homomorphic encryption, 06 2024.

[6] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley and Sons Inc., New York, 2nd edition, 1996.

[7] Merriam-Webster. *The Merriam-Webster Dictionary*. 2016.

[8] Rahul Awati. What is polymorphism? — definition from techtarget, https://www.techtarget.com/whatis/definition/polymorphism, 2024.

[9] Olof Leimar. The evolution of phenotypic polymorphism: Randomized strategies versus evolutionary branching. *The American Naturalist*, 165(6):669 – 681, 2005.

[10] Vinh Nguyen. A study of polymorphic virus detection. Technical report, 11 2018.

[11] Christopher Strachey. Fundamental concepts in programming languages. *Higher-Order and Symbolic Computation*, 13:11 Â 49, 1967.

[12] Steven M. Bellovin. Frank miller: Inventor of the one-time pad. *Cryptologia*, 35(3):203 – 222, 2011.

[13] Gilbert Vernam. Secret signal system, patent no. 1,310,719, 22 july, 1917.

[14] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5 – 83, 161 – 191, 1883.

[15] Claude Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:656 – 715, 1949.

[16] John Earl Haynes and Harvey Klehr. *Venona: Decoding Soviet Espionage in the United States (Yale Nota Bene)*. Yale University Press, 1999.

[17] Basal Halak. *Physically Unclonable Functions, From Basic Principles to Advanced Hardware Security Applications*. Springer, Cham, Switzerland.

[18] Albert H. Carlson, Sai Ranganath Mikkilineni, Michael Totaro, Robert Hiromoto, and Richard B. Wells. An introduction to local entropy and local unicity. *International Symposium on Networks, Computers, and Communictions, ISNCC 2022*.

[19] P. L'Ecuyer. Random numbers for simulation. *Communications of the ACM*, pages 85 – 98, 1990.

[20] Melissa E. O'Neill. Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation. Technical Report HMC-CS-2014-0905, Harvey Mudd College, Claremont, CA, 9 2014.

[21] Rod Downey and Denis R. Hirschfeldt. Algorithmic randomness. *Communications of the ACM*, 62(5):70 – 80, 2019.

[22] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Cryptanalytic attacks on pseudorandom number generators. *Fast Software Encryption, Fifth International Workshop Proceedings*, pages 168 – 188, 1998.

[23] Uli Maurer. A universal test for random bit generators. *Journal of Cryptography*, 5(2):89–105, 1992.

[24] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimendsionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modelling and Comutpute Simulation*, 8(1):3 – 30, 1998.

[25] F. Handayani and N. P. R. Adiati. Analysis of geffe generator lfsr properties on theapplication of algebraic attack. *AIP Conference Proceedings*, 2019.

[26] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Cryptanalytic attacks on pseudorandom number generators. In *Fast Software Encryption, Fifth International Workshop Proceedings (March 1998)*, pages 168 – 188. Springer-Verlag.

[27] Albert H. Carlson, Sai Ranganath Mikkilineni, Michael Totaro, and Christopher Briscoe. A venona style attack to determine block size, language, and attacking ciphers. *International Symposium on Networks, Computers, and Communictions, ISNCC 2022*, 2022.

[28] Donald E. Knuth. *The Art of Computer Programming*. Addison-Wesley, Massachussets, 1997.

[29] Haytham Idriss, Pablo Rojas, Sara Alahmadi andTarek Idriss, Albert Carlson, and Magdy Bayoumi. Shadow pufs: Generating temporal pufs withproperties isomorphic to delay-based apufs. *IEEE International Symposium on Circuits and Systems (ISCAS)*.

[30] Jerome Howard Saltzer. *Traffic Control in a Multiplexed Computer System*. PhD thesis, Massachusetts Institute of Technology, 1966.

[31] Bryan Schauer. Multicore processors – a necessity, https://web.archive.org/web/20111125035151/http://www.csa.com/discoveryguides/multicore/review.pdf, 2009.

[32] Marko Misic, ore urević, and Milo Tomasevic. Evolution and trends in gpu computing. pages 289–294, 01 2012.