[Carlson, Albert H., Mandeep Singh. (Jan. 8-10, 2024). Exploring Polymorphic Algorithms and Their Use in Cryptography. IEEE, Conference: 2024 IEEE 14th Annual Computing and Communication Workshop and Conference (CCWC)At: Las Vegas, NV. Source: https://www.researchgate.net/publication/377298199_Exploring_Polymorphic_Algorithms_and_Their_Use_in_Cryptography]

Exploring Polymorphic Algorithms and Their Use in Cryptography

Mandeep Singh[*], **Albert Carlson**[†].

[*]Computer Science & Engineering Department, Santa Clara University, Santa Clara, CA, USA
[†]Chair for Entropy and Encryption, Quantum Security Alliance &
Computer Science Department, Austin Community College, Austin, TX, USA
Email: [*]msingh5@scu.edu, [†]albert.carlson@austincc.edu

***Abstract*—With the ever increasing demand of higher security standards for data encryption, algorithms need to be scalable and efficient while maintaining a high key space to prevent brute force attacks. Although the current encryption standards are decent for the current era of computing power, many will be rendered useless as quantum computing becomes more relevant. Even now, the current encryption standard of AES-256 with CBC is vulnerable to attacks such as the key recovery attack. This paper covers the basic foundations of polymorphic encryptions, the mathematical principles behind them, and a novel post-quantum polymorphic encryption algorithm.**

Keywords: Data Security, Quantum Computing, Cryptography, Encryption, Polymorphic Algorithms, Information Theory

I. Introduction

Polymorphic encryption is the method of obscuring data by breaking a message into sub-messages that are smaller than the unicity distance using "randomly" selected cipher/key pairs that morph from sub-message to sub-message [1], [2]. The most familiar polymorphic cipher is the one time pad also known as the Vernam cipher [3]. Polymorphic ciphers

A. Current Encryption Standards

Currently, in the security field, the Advanced Encryption Standard (AES) [4] with modes, such as CBC, is considered to be the gold standard for encryption for both government agencies and private entities. However, with the advent of quantum computing technology, these cryptographic encryption standards are threatened by the increased computing power of quantum computers. There are also proven side-channel attacks that degrade the reliability of the current encryption standards [5]. These trends most directly affect public key encryption (PKE) [4], [6] systems such RSA, ECDSA, ECDH, and DSA. The current symmetric key cryptosystems, such as AES-256, are considered relatively quantum proof because of the analysis of Grover's algorithm [7]. Grover's algorithm, a random walk searching algorithm, proves that the strength of symmetric key algorithms reduces the key space of those ciphers by a factor of $\sqrt{2}$ on a quantum computer. Keeping the same degree of security for those symmetric key ciphers in the post-quantum environment (PQE) requires that the key space be increased by a factor of 2 to compensate for the efficiency of the search algorithm - something that is easily accomplished with scalable cryptographic algorithms. Either the key space is directly increased or using combinations of ciphers. However, theoretical proofs do not necessarily port to the real world seamlessly. Tests were run by Fujitsu [8] whose analysis shows that a quantum computer is able to break AES-256 in under 4 seconds and the military grade AES-2048 within 104 days. With the increasing availability and use of quantum computers, all of the current encryption algorithms will not be as secure an option as they were prior to the PQE, necessitating an advancement in encryption technology and algorithms. Polymorphic encryptions are presented in this paper as an example of that solution [1].

The paper is organized into five sections, including references. The first section introduces the

979-8-3503-6013-4/24/$31.00 ©2024 IEEE

subject and is followed by Section II on the basics of polymorphic encryption. This particular type of encryption is relatively new and is not generally well understood. Section III covers implementing polymorphic encryption and the required components for the technology. Section IV contains conclusions related to the work and future work anticipated in the field. The final section is composed of the references used in the work.

II. Principles of Polymorphic Encryptions

A. What is a Polymorphic Encryption

A polymorphic encryption is an encryption whose encryption/decryption key pair is changed frequently and irregularly during use. For example, assume that there is an output key with the value 5 and the encryption algorithm splits the input and adds these values up in order for the output to be equivalent to the input. The value of 5 can be derived many different ways - 1 + 4, 2 + 3, 1 + 1 + 3, 2 + 2 + 1, etc. There are effectively almost a transfinite number ($\aleph_0$) of ways to create the output value from the inputs. Even though the inputs are different, they arrive at the same end value, resulting in having a changing key pair while maintaining the same output. The exact number of such pairings is $\aleph_0 \approx \infty$. Why is this so important and is different from current encryption methods? It all comes down to Shannon's Laws [9] and the concept of entropy (the "uncertainty" in information). The entropy for any given message is:

$$H(x) = -\sum_{i=j}^{n} pr(x_i) lg(pr(x_i)) \tag{1}$$

where $\sum$ is the sum over the possible values, pr is the probability, and lg is the base 2 logarithm. This gives the entropy for any part of a given message or file which is known as "local entropy" [10], or $H_l(x)$. The local entropy then gives $R_{\lambda,l}$, the local redundancy, for that sub-message in the message or file. The local redundancy in the sub-message is given by

$$R_{\lambda,l} = 1 - \frac{H(x)}{H_{max}(x)} \tag{2}$$

which leads to the definition of the local unicity distance ($n_l(x)$) - the amount of information or data that is required to unambiguously determine the original plain text message [9], [10] given by

$$n_l(x) = \frac{log(|K|)}{R_{\lambda,l}(x) log(|A|)} \tag{3}$$

The conclusion is that some areas of a message are easier to break than others in the message. Therefore, an attacker can locate and attack the vulnerable areas of a message. If enough data is collected, the encryption can be broken. An example of this is the Soviet Union reusing "one time pads" (OTPs), which led to intelligence agencies being able to break their coded messages [11], [12]. However, if all portions of the message do not have enough data to break, the message cannot be broken. This is why some messages can be broken and others cannot. In order to prevent enough data from accumulating so no part of the message can be broken, the message is split into "shards" [13], [2], or submessages contained in the message stream.

B. Shards and Isomorphism

If $|shard| < n_l$, such that no shard achieves unicity distance, then no portion of the message has enough data to be broken. This means that each shard needs its own cipher and key pair and these pairs must not be predictable. This method makes it completely infeasible to break a message while being cheaper than a OTP [14]. The "key space" is the number of possible keys that can be used by a cipher. To "brute force" [4] the decryption of a message, the worst case would be to try every single key in the key space. Therefore the number of keys that must be tried to successfully brute force a decryption on the average is [15], [16] is:

$$t_k \approx \frac{|K_c|}{2} \tag{4}$$

However, from Shannon's Theory, this number of attempts has a limit, or the maximum that has to be tried on average. For many messages, more than one key will result in encryption or decryption [13]. Equivalent keys normally occur when not all of the alphabet of the language appear in the message. These keys are known as isomorphic keys [17] since they are equivalent. For any two equivalent keys i, j where $i \neq j$:

$$E_{k_i}(M) = E_{k_j}(M) \tag{5}$$

Creating the sets of isomorphic keys, only one key from each isomorphic set $K_{iso,i}$ must be tried in order to eliminate all of the keys from consideration or to accept the isomorphic key as the answer for decryption. Therefore, the true key space is

$$|K_e| = |K| - \sum_{i=1}^{n} |K_{iso,i}| \tag{6}$$

or, the number of sets of equivalent keys. The key representing the set is known as the "systematic isomorph" for the set [13]. This shows that key spaces are often smaller than the maximum and the cipher strength must be based on the message as well as the cipher and key. However, the problem of smaller than expected key spaces can be compensated for by using a polymorphic cipher. A polymorphic cipher engine has a series of ciphers from which one is chosen at random for each shard so, for n shards, the key space is

$$|K_p| = \prod_{i=1}^{n} K_{c_i} \tag{7}$$

This makes a polymorphic cipher much more secure since the key space increases multiplicatively with each component based on factorial key space. The vast number of new keys is why correctly implemented polymorphic encryptions are quantum proof. As computing power gets stronger, it is easier to check all the keys in a key space using a brute force attack, making ciphers lose their effectiveness. This is why the goal of cryptography researchers is to increase the key space over time. For a polymorphic encryption, a new encryption or larger key size is not needed to combat this issue because the shards do not have the required amount of information needed for statistical analysis. Polymorphic ciphers just need to use smaller shards, thus further increasing the key space without much overhead. The smallest shard is a single character. Faster machines make it possible to create those smaller shards during the encryption process without increasing processing time due to the ability to spread the overhead using parallel processing, threads, or graphics processing units (GPUs). Polymorphic ciphers become stronger with increased computing capabilities and keep pace with better computers.

C. Isomorphic Reduction

All ciphers are ultimately substitution (S) ciphers [18]. Consider a block of characters as comprising a single metacharacter (a character composed of characters, or a block of characters) [13] in a metalanguage. Then, an attacker can treat any cipher applied to this block as an S cipher. Every cipher can be replaced by the mapping function ($\mapsto$), the definition of an S cipher. For example, a P cipher across byte boundaries is a S cipher with constraints [19]. After mapping a cipher into an S cipher, multiple S ciphers can also be combined using "idempotence." Idempotence [20] is a property that collapses many ciphers into an equivalent cipher. The action of applying idempotence collapses more than one S cipher into a single, equivalent S cipher with a new key. The application of the property of idempotence can be shown mathematically as

$$S_{k_1}(S_{k_2}(M)) = S_{k_e}(M) \tag{8}$$

where S is a substitution cipher and k is a key. How does this work for ciphers that reduce to the S cipher? The reduction proves Feistels' assertion that all ciphers are S ciphers [21] Assume a product cipher is made up of a P, an S, an XOR, and another S. Since each maps to an S cipher, this can be reduced to:

$$S_e = S_1(S_2(S_3(S_4(M)))) \tag{9}$$

Applying isomorphic cipher reduction, it can be easily proven that a round in the AES cipher [4] is actually only a single S cipher. Since rounds collapse due to idempotence, the full round structure of the entire AES encryption collapses to a large S cipher. Note that the AES cipher is of the type PSP (permutation, substitution, permutation). So how does a PSP cipher compare to a large S cipher? The standard comparison is to compare the unicity distance (security) of the two types of ciphers [22]. The proof is as follows. Let Sec be the ratio of security between the two ciphers, n_i is the unicity distance of the cipher i, $|A|$ is the size of the alphabet, b is the number of bits in the key, and R_λ is the redundancy of the language.

$$Sec = \frac{|n_{PSP}|}{|n_s|} = \frac{\frac{log(b!|A|!b!)}{R_\lambda log|A|}}{\frac{log(A!)}{R_\lambda log|A|}} \tag{10}$$

$$= \frac{log(b!A!b!)}{log(A!)} = \frac{log(b!) + log(|A|!) + log(b!)}{log(|A|!)}$$

$$= 1 + 2\frac{log(b!)}{log(|A|!)}$$

Now let $\epsilon = \frac{log(b!)}{log(|A|!)}$ and $\epsilon < 1 \; \forall \; |A| < 2$. For English $|A| = 26$. Taking a one byte cipher, then $\epsilon \approx .37$. A six byte cipher would have $\epsilon \approx .00004$. We see that as the block size increases, the difference in security decreases. However, with isomorphic reduction, $\epsilon = 1$ and reducing to the S cipher $\epsilon = 0$. Therefore $Sec = 1$. Any single cipher will reduce in this manner, making polymorphic ciphers the only cipher with increasing security. There is no increased security with the increase in complexity of PSP (and related) cipher types. This leads to the conclusion that the increased time and resources of using these complicated ciphers are not worth the effort.

III. Polymorphic Encryption Conceptually Implemented

A polymorphic encryption sysem is composed of a collection of different and important components: a large signature from each system, a Trusted Third Party/Certificate Authority (TTP/CA) for signature storage and transfer, a library of cryptographic ciphers (i.e. AES-256), a pseudorandom number generator (PRNG) (preferably polymorphic) for wave randomization in order to generate keys, and a shard reduction and randomization algorithm. Optional components to increase the complexity are video watermarking, audio watermarking, text watermarking, and compression as vehicles for the data. This paper will not cover watermarking or shard randomization algorithms as their complexity are out of the scope for this paper. However, the other methods needed for the algorithm are described here.

A. Signatures

Every computer has a unique state that consists of hardware configurations, memory and bus configurations, MAC and IP addresses, serial numbers (chip and system), software license numbers (one for each program), OS serial number, BIOS data, and so on. When appended, they comprise a unique identifier for the system, known as a "node signature." Some contents may be similar between systems, but taken as a whole they become a "fingerprint" that can identify the system. Fingerprints are just one type of signature. Many others exist.

These identifiers also provide information that can be used to generate unique random numbers for the keys. Each of these values can be read from the system and reassembled when needed. Consider a "signature" that is formed when this data is concatenated into one string. Imagine this string is arranged as a rectangle. The data can either in any sized chunk. Since reading the string in order can potentially lead to an attacker reassembling it, it is therefore necessary to reorder the signature into a different string through some kind of transformation. Reordering the constituent components completely changes the signature. Such an action can be accomplished by creating a variable to change the assembly order of the signature or by creating a new structure and generating the key from this unrelated collection of data. But this is not enough, we want to do more so an attacker cannot follow. If we use only a part of the signature, it makes it hard for the attacker to try various combinations of this large signature. So we use a Random Number Generator (RNG) to choose the number of bytes/bits we want to keep while making sure there is enough information to keep the signature long enough to be useful in generating keys.

One method that will accomplish this goal is to use a combination of data reassembled from the signatures of both the receiving and sending nodes. The resulting signature data is assembled into a rectangle that starts at a random position in the structure and skips around the rectangle according to some sequence. That sequence is made by picking two numbers, an offset c and a constant k. Once the offset is selected, the key is constructed by adding that bit, or byte, or bytes to the key. Next, consider some function whose form is given by

$$f(x) = (cx + k) \mod p \tag{11}$$

where k is a constant and p is the length of the signature. This allows skipping over the rectangle and picking numbers (pseudo)randomly. Naturally, there are many different functions and Psuedorandom Number Generators (PRNGs) to create the movement in the signature matrix. These can be selected using seeds sent during the handshake or derived

in some other manner, such as a Diffie-Hellman exchange [23]. The same action may be taken by using physically unclonable functions (PUFs) and shadow PUF structures. In this way, those numbers decouple themselves from the signature itself.

B. Physically Unclonable Functions

Physically unclonable functions (PUFs) are a collection of information that are dependent on the construction or manufacturing of a unit that can be used as an identifying signature of the unit [24], [25]. These differences between units typically come from random variations that arise during the manufacturing process, such as the amount of copper deposited on PC board traces, the concentration of chemicals in an integrated circuit junction, or the serial number of a constituent unit in a computer. One of the more common PUF values is characterizing the power-on state of memory bits in a computer. Some bits will always assume the value of binary 1, some of binary 0, others will mostly be binary 0 (known as "X"), and finally, the remainder will power on to mostly binary 1 (known as "Y") [26]. If the PUF value of interest is recorded, then the PUF value can be used to uniquely identify the unit as long as the unit is not physically altered.

Because the PUF signature is unique, it can be used to generate keys and seeds for PRNGs. Further, if combined with another PUF, the resulting PUF (say PUF') can be used in the same manner with information from both parties of the exchange. This approach is used to ensure that both encrypting parties have to provide information unique to their computing machines to share messages. The procedure for such a sharing of information is easy to explain. Consider two computers denoted by "S" (the sender) and "R" (the receiver). Each party selects half of their PUF signature and sends it to the other party. This is normally done via digital certificate using a TTP/CA. Only the part selected by the party for their half of the PUF' is transferred. Each party then locally generates the PUF' by assembling the data of their half of the PUF with the received half in an agreed upon pattern. The PUF' is then used as either a seed or to generate keys and is often assembled in the form of a table, known as a PUF table.

C. Shadow PUF Tables

A major problem with a PUF table is that if an attacker can monitor the requests for data from a PUF table, they can potentially reconstruct the PUF' table and, using the patterns, then figure out how the parties are constructing keys. In order to keep the attacker from discovering the contents of the PUF' table, the PUF' table must morph as it is used. Using the principles of Shannon Theory, specifically entropy and unicity distance [9], and those of polymorphic ciphers [1], it becomes necessary to replace the PUF' table at frequent and irregular intervals. These intervals are determined by the information that accumulates with each use of the PUF table [27]. This time is often referred to as the time to live (TTL) [13] and is related to the local entropy and unicity distance [10] of the PUF' table. The new table is known as a "shadow PUF" [28] table. Idriss, et al showed that it is possible to create any number of shadow PUF tables from the PUF' table that have the same statistical properties as the original table, but not identical contents. By producing a series of shadow PUF tables and using each for less time than the TTL associated with the shadow PUF table, there is not enough information for an attacker to reconstruct the PUF' table in use.

D. TTP/CA

If signatures are exchanged, the sender can encrypt and not have to send the entire key to each other, thereby potentially revealing that key to sniffers. The key is developed from the signature at the beginning of the message. Leaving the seed numbers with the file does not hurt if the file is stolen since the system is needed to develop the key. Naturally, at no time should a user send the entire signature if they can avoid doing so. Certainly sending the signature from the machine it belongs to should be avoided. Therefore, it is desirable to use a TTP or CA that holds many signatures as the source of the partial signature exchange. Each node must read its own signature and report it to the TTP/CA for distribution. The exchange of the partial signature happens one time and only between the node and the TTP. In addition, the TTP should also append false data to the signature stored as a cryptonull. The nodes agree to what parts of the signature are valid and also agree on the order in which the signature

is sent. All that is required is a way to derive the key on both the sender and receiver's systems that appears completely random but results in the same key. In order to do so, a method known as wave randomization is used to achieve this randomness.

E. PRNGs

Randomization is a major concern of cryptography. Violation of this principle is the basis of the very effective Venona attack [11], [12], [29] used by the US during most of the 20^{th} century. However, it is mathematically and practically impossible to generate random keys at two different nodes out of direct observational contact with each other. Consider the sources of randomness such as coin tosses [15], cosmic events [30], lava lamps [31], and white or pink noise [32], [33]. All these sources of randomness fall into one of two categories - physical events and mathematical functions. Physical events are observed and recorded as a source of random event sequences. However, if both users cannot see the same event, they cannot use it. The only way to get around this is to record the event and physically transfer it between the two nodes. This is expensive and time-consuming, as well as susceptible to being monitored while the data is being transferred between users. So, while true randomness is the goal, achieving true randomness makes reliable encryption between widely separated users impossible. Instead, What is required is deterministic sequences that look random for the length of the message. Nothing that can be calculated is truly random [34]. Further, attacks are now being made on the PRNGs as a way to recover encryptions [12], [35], [36]. Although PRNG sequences are calculated, sequences can be made to appear random over the length of the sequence. True randomness as a source generator (TRNG) is denoted by:

$$\forall\, t \in \{0 \leq t < \infty\} \rightarrow pr(t+1) = pr(t) \qquad (12)$$

where $pr(x)$ is the probability of the event or number of x occurring. Since is important to have deterministic randomness, a PRNG is used to achieve the appearance of randomness over the short-term performance of the sequence. All PRNGs cycle, with a period of λ accesses to the PRNG, and do not repeat numbers during the cycle. A user can choose to start any place in the sequence with that point being called the "seed." Each access picks the next value in the RNG sequence and returns it. Unfortunately, an attacker can often pattern the PRNG and figure out what PRNG algorithm is being used. From that information, an attacker can determine where in the cycle the RNG is operating. A constantly changing function with a great deal of complexity and enough factors to make it difficult to determine the next number returned from the PRNG is required.

IV. Conclusion & Future Work

The computing world has entered an era in which both classical and quantum computing are both possible. One of the most important problems is that of secure encryption. Asymmetric encryption algorithms are based on problems known to be relatively easily broken by quantum computers. The National Institute of Standards and Technology (NIST) has identified quantum-safe algorithms (QSAs) [37], but the list has been diminished quickly by attacks based on classical computers [38]. Safe asymmetrical algorithms are possible in the quantum environment, but the key size for those algorithms would be too large to practically implement. Instead, polymorphic ciphers are a better solution.

Future work can be done to further improve the efficiency of polymorphic ciphers or introduce more components to increase the security of the data. Further research on securing the TTP/CA communication and signature-sharing handshake is also crucial. Database security of the TTP/CA is also a potential field of research. Another area of interest and importance is shard randomization algorithms - making sure those algorithms are more complex allows the cipher to be more secure. Since polymorphic ciphers are naturally parallelizable, they should be further researched for low-latency applications such as the Internet of Things (IoT) [39], [40], [41].

Polymorphic ciphers provide strong security in both the classical environment and PQE, given that they can be customized using symmetrical key algorithms and mutate as they are used. Combined with polymorphic random number generators (polyRNGs) [42], [43] and physically unclonable functions (PUFs) [25], [28], polymorphic ciphers provide methods to increase local entropy [9], [10] and prevent the heuristic use of patterns to decrypt

encrypted messages. As the era of quantum computing nears, polymorphic ciphers and polyRNGs will play a crucial role in protecting data.

REFERENCES

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

[2] Albert Henry Carlson and Robert LeBlanc. Polymorphic hardware engine, patent number 9,178,514, 2015.

[3] Gilbert Vernam. Secret signal system, patent number 1,310,719, 1977.

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

[5] Albert Carlson, Bhaskar Ghosh, and India K. Dutta. Using the collision attack for breaking cryptographic modes. *13th International Congress on Computing, Communication, and Networking Technologies*.

[6] Johannes A. Buchmann, Evangelos Karatsiolis, and Alexander Wiesmaier. *Introduction to Public Key Infrastructures*. Springer-Verlag, Berlin, Germany, 2013.

[7] Lov K. Grover. Quantum computing: How the weird logic of the subatomic world could make it possible for machines to calculate millions of times faster than they do today. *The Sciences*, pages 24 – 30, 1999.

[8] Richard Chirgwin. Quantum computers won't break rsa encryption any time soon, https://www.itnews.com.au/news/quantum-computers-wont-break-rsa-encryption-any-time-soon-590115: :text=they

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

[10] 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 Communications, ISNCC 2022*.

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

[12] Albert H. Carlson, Sai Ranganath Mikkilineni, Michael Totaro, and Christopher Briscoe. *A Venona Style Attack to Determine Block Size, Language, and Attacking Ciphers*, 2022.

[13] Albert Carlson. *Set Theoretic Estimation Applied to the Information Content of Ciphers and Decryption*. PhD thesis, University of Idaho, 2012.

[14] Albert Henry Carlson, Robert Le Blanc, Robert Carlson, Patrick Doherty, and Carlos Gonzales. Polymorphic one time pad matrix, patent number 10,069,805.

[15] Sheldon Ross. *A First Course in Probability*. MacMillan Publishing, Inc, New York, 1976.

[16] R. Lyman Ott and Michael T. Longnecker. *An Introduction to Statistical Methods and Data Analysis* 7^{th} *edition*. Cengage Learning, 2016.

[17] Albert Carlson, Bhaskar Ghosh, Indira Dutta, Shivanjali Khare, and Michael Totaro. Keyspace reduction using isomorphs.

[18] Bhaskar Ghosh, Indira Dutta, Shivanjali Khare, Albert Carlson, and Michael Totaro. Isomorphic cipher reduction.

[19] Albert H. Carlson, Robert E. Hiromoto, and Richard B. Wells. Breaking block and product ciphers applied across byte boundaries. In *The 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, pages 733–736, 2011.

[20] Richard Wells. *Applied Coding and Information Theory*. Prentice Hall, Upper Saddle River, 1999.

[21] Horst Feistel. Cryptography and computer privacy. *Scientific American*, 228(5):15 – 20, 1973.

[22] Albert Carlson, Torsten Gang, Garrett Gang, Bhaskar Ghosh, and Indira Dutta. Evaluating true cryptographic key spacesize.

[23] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644 – 654, 1976.

[24] D.W. Bauder. An anti-counterfeiting concept for currency systems. Technical report, Sandia National Labs, Albuquerque, NM, 1983.

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

[26] Bertrand Cambou. A xor data compiler: Combined with physical unclonable function for true random number generation. *2017 Computing Conference*, pages 819 – 827, 2017.

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

[28] 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)*.

[29] Bhaskar Ghosh, Albert Carlson, and Indira Dutta. A demonstrable break of pcbc mode. *International Symposium on Networks, Computers and Communications (ISNCC): Trust, Security and Privacy (ISNCC 2023), Dohar, Qatar*.

[30] Rupert Goodwins. Start-up generates random numbers from space. *ZDNet International*.

[31] Tom McNichol. Totally random. *Wired Magazine*.

[32] Hui-Hsiung Kuo. *White Noise Distribution Theory*. CRC Press, Boca Raton, 1996.

[33] Allen Downey. *Think Complexity*. O'Reilly Media.

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

[35] 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.

[36] Eric Drexler. 2009.

[37] R. A. Perlner and D. A. Cooper. Quantum resistant public key cryptography: A survey. pages 85 – 93, 2009.

[38] Albert H. Carlson and Keeper L. Sharkey. Nist quantum proof algorithm analysis. Technical report, Quantum Security Alliance.

[39] M. A. Obaidat, J. Brown, S. Obeidat, and M. Rawashdeh. A hybrid dynamic encryption scheme for multi-factor verification: A novel paradigm for remote authentication.

[40] M. O. Brwon and J. Brown. Two factor hash verification (tfhv): A novel paradigm for remote authentication.

[41] Matluba Khodjaeva, A. Muath, and Douglas Salane. *Mitigating Threats and Vulnerabilities of RFID in IoT through Outsourcing Computations Using Public Key Cryptography*. Springer.

[42] Benjamin Williams, Albert Carlson, and Robert Hiromoto. Novel innovations for improving the quality of strong prngs. *13th ICCCNT*, 2022.

[43] Benjamin Williams, Robert E. Hiromoto, and Albert Carlson. A design for a cryptographically secure pseudo random number generator. *2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, 2019.