The VICCard Cipher: Our Contribution to the Field of Playing Card Cryptography.

Isaac Reiter
63 Highland Drive
Fleetwood, PA 19522
Department of Mathematics
Kutztown University of Pennsylvania
2020

Advisor/Faculty Sponsor: Dr. Eric Landquist

Abstract

Before computers, military tacticians and government agents had to rely on pencil-and-paper methods to encrypt information. For modern agents that want to use low-tech options in order to minimize their digital footprint, non-computerized ciphers are an essential component of their toolbox. Consider a deck of cards. There are $52! \approx 2^{225.58}$ ways to mix a deck of cards. If each deck order is a key, this means that there are $52! \approx 2^{225.58}$ different ways to encrypt a given message. To create some perspective, most computer ciphers feature either 2^{128} or 2^{256} different ways of encrypting the same message. Hence, a cipher created from a deck of cards has the potential to emulate the security of many computer ciphers. The focus of this paper is the creation of a unique, secure playing card cipher: VICCard. Its security is rooted in its combination of numerous cryptographic principles, including a substitution checkerboard, columnar transpositions, lagged Fibonacci generators, and junk letters. As evidenced by certain randomness tests, VICCard has the potential to extensively randomize an English plaintext.

Contents

Acknowledgements

I gratefully acknowledge the support of the Honors Program of Kutztown University of Pennsylvania, for which this paper is intended. Also, I am very grateful for the opportunities provided by the KUBEARS grant (Kutztown University Bringing Experience About Research to Students). Much of the preliminary work for this research was completed under this grant in the summer of 2019. Most importantly, I would like to thank my research advisor, Dr. Eric Landquist. I am remarkably fortunate to have had Dr. Landquist as my professor and mentor throughout my undergraduate experience.

1 Introduction

1.1 Cryptography's Journey from Painting to PC

Cryptology is the science of safe, secure communication. It examines how to transform a message (called the plaintext) into an encoded form (called the ciphertext). An effective cryptologist must be proficient in two tasks: cryptography and cryptanalysis. Cryptography is the study of creating effective ciphers. Cryptanalysis is the study of breaking these ciphers. Cryptography is "code writing", and cryptanalysis is "code breaking." The creator of a secure cipher uses both of these skills. He first uses cryptography to create his cipher, and he then uses cryptanalysis to see whether his cipher is as secure as he thinks.

Egyptian hieroglyphics are one of the oldest instances of cryptography. For example, the tomb of Khnumhotep II featured a myriad of pictures and symbols. These pictographs told the story of the deceased with a beautiful visual display [10]. By contrast, cryptography is more frequently used for less aesthetic purposes: war and espionage.

Just as weapons of war have become more refined, cryptography has undergone careful attention and development. Humble ciphers such as Julius Caesar's cipher and the Vigenere cipher have given way to more advanced creations such as Rasterschlüssel 44 and VIC. As the ciphers became more complicated, they became more secure. However, they also became more impractical. As a result, cryptography's next step in its evolution was to enlist the help of machines. The most compelling example of this is the German Enigma machine. In order to break this cipher, the Allies enlisted cryptographers to fight fire with fire. To break the Enigma cipher they built an even better machine: a computer. Computer ciphers have now become the norm, encrypting everything from government secrets to emails between friends.

The advantage of computer ciphers is their ability to use formidable n-bit encryption. A cipher with n-bit encryption uses a pool of 2^n possible keys, meaning that there are 2^n possible ways of encrypting any given message. Typically, computer ciphers use 128-bit or 256-bit encryption. This prevents the ciphers from being cracked through brute force attempts that test every possible key.

Although cryptography has become mechanized since WWII, cryptographers have not discounted the strength and security of ciphers that are executed by hand. In fact, computerized ciphers can be based on the general cryptographic principles found in hand ciphers.

1.2 Why Playing Cards?

Here is an important observation. There are 52! ways to permute a deck of cards. This means that there are $52 \times 51 \times 50 \times 49 \times \cdots \times 3 \times 2 \times 1$ ways to arrange a deck of cards. To give you an idea of the scope of this number, consider the following scenario that was adapted from a quote of Stephen Fry. Imagine a trillion universes, each of which contains a trillion planets. Each of these planets contains a trillion people, and each person has a

trillion decks of cards. If everyone can shuffle all of their decks one time per second, it would take over two and a half trillion years before every possible deck order has been created [4]. Simply put, Fermilab estimates that there are approximately anywhere from 10^{49} to 10^{50} atoms that make up the earth [3]. This means that there are more ways to shuffle a deck of cards than there are atoms that compose the earth. Since $52! \approx 2^{225.58}$, a deck of cards has the potential to provide 225.58-bit encryption. This is enough to compete with the security provided by typical computer ciphers. From this arises the following question: can we use playing cards to create a secure, efficient hand cipher?

1.3 Existing Playing Card Ciphers

Given that the field of playing card ciphers is remarkably specialized, not a lot of playing card ciphers have been created. Aaron Toponce has a great website that lists most if not all of the publicly known playing card ciphers [16]. Before creating my own cipher, it was important to look at the work that has already been done. Performing cryptanalysis on existing ciphers can help determine both the strengths and weaknesses that tend to occur in playing cards ciphers. With this knowledge, one is better equipped to maximize the former and minimize the latter. Two playing card ciphers in particular are of interest.

First, Card-Chameleon is a playing card cipher created by Matthew McKague for his master's thesis [9]. His intention was to create a hand version of the computer algorithm RC4. At first glance, Card-Chameleon's straightforward, easy to remember algorithm makes it attractive. However, scrutiny of this cipher revealed a fatal weakness. Assuming a random key for each letter, Card-Chameleon encrypts any given letter into the exact same letter with probability $\frac{1}{13}$. Here's why this is a weakness. For each plaintext letter, the encryption algorithm should be such that every letter has the same probability of occurring. In other words, a plaintext letter should have a $\frac{1}{26}$ probability of encrypting to any other letter. With Card-Chameleon, however, it is disproportionately likely that a letter will encrypt to itself. Unfortunately, this deviation from the magic $\frac{1}{26}$ probability is too significant to overlook (for more information, see the paper that Dr. Landquist and I wrote on this cipher [11]).

Second, Chaocipher is a cryptosystem that was created by John F. Byrne in 1918 [6]. Although Chaocipher has been around for over a century, the disclosure of the Chaocipher algorithm occurred as recently as 2010 [13]. As he was examining previously invented playing card ciphers, Toponce had the idea of adapting the Chaocipher algorithm to playing cards [15]. Given the respectable security of Chaocipher, I did not find a weakness that was as severe as that in Card-Chameleon. The closest thing to a weakness is the existence of plaintext/ciphertext pairs (or pt/ct pairs). A pt/ct pair is when two identical plaintext letters encrypt to the same ciphertext characters, such as two `a`'s encrypting to two `o`'s. Greg Mellen noticed that when he divided messages encrypted by Chaocipher into blocks of 13 letters, pt/ct pairs rarely occurred within these blocks [12]. Moshe Rubin hypothesized that pt/ct pairs will only occur if the two plaintext letters are separated by a distance of

eight letters [12]. In order to put a rest to this question, I wrote a program that took two `a`'s and tried every 1-letter, 2-letter, 3-letter, 4-letter, and 5-letter combination between these two `a`'s. After testing all 12,356,630 of these cases, the program did not find any pt/ct pairs. However, it did find pt/ct pairs with certain 6-letter combinations. As a result, we can say for certain that at least six letters must be between two plaintext characters for a pt/ct pair to occur.

1.4 The Current Approach

Analyzing existing ciphers revealed a general trend among them. Most if not all playing card ciphers are stream ciphers. This means that they encrypt plaintexts one letter at a time. The typical strategy is to first encrypt a letter and then alter the deck order before encrypting the next letter. In creating a unique cipher, I used a different approach. I focused my efforts on creating a block cipher. With a block cipher, the plaintext is encrypted in blocks of letters. Specifically with our cipher, we are encrypting the entire message at once in one large block.

2 VICCard

2.1 The Hollow Nickel Case

In 1953, Jimmy Bozart was a young 13-year-old boy living in Brooklyn. He delivered newspapers for the Brooklyn Eagle. On June 22, he was counting his tips when he noticed that one of his nickels was lighter than the others. As the nickel slipped from his fingers, it hit the floor and cracked neatly into two pieces. Inside, the nickel was completely hollow. Furthermore, it contained a tiny piece of microfilm with numbers on it [2].

When local police officers heard of this discovery, they scrambled to track down Jimmy and his nickel. Just in case he carelessly spent his valuable discovery, they examined the Bingo money from the church and ice cream money from a Good Humor vendor. They eventually found Jimmy, who willingly gave them the nickel. Realizing the potential gravity of what they possessed, the New York police officers turned the coin over to the FBI [1].

In their research, the FBI investigators looked into whether the coin was simply a trick nickel meant for gags or magic tricks. This theory failed due to the imprecision with which the nickel was made. The hollow part was not big enough to contain much. Being a magician myself, I have handled high-quality hollow coins. The craftsman has to balance two factors. First, they have to make sure that the hollow coin is not too big. Otherwise, it will excite suspicion from the audience. On the other hand, the coin cannot be too small. If it is, anything that the magician is trying to hide inside the coin can easily get stuck. No feeling is worse than realizing mid-performance that your props are not cooperating. Jimmy's hollow nickel was not crafted with this much precision [2].

The FBI had to solve two questions: what was the coin's purpose, and what was the meaning of the numbers on the microfilm? In an exceptional stroke of luck, both questions were answered by Russian spy Reino Häyhänen. Häyhänen did not begin his espionage career by choice. Because he was fluent in Finnish, he was drafted as a translator for the Communist secret police during the Finnish-Soviet war. Upon the end of the war, he remained in Finland in order to report anti-Soviet individuals. Häyhänen became a member of the Communist party in 1943, and in 1948 the KGB assigned him a new task. Assuming the identity of Eugene Nicolai Maki, an immigrant from America to Estonia, he was to act as a Soviet spy in the United States. In 1957, Häyhänen contacted the U.S. embassy in Paris, desiring to defect to the United States. Following his defection, Häyhänen gave FBI officials the details of his operations. Most importantly, he revealed how hollow coins, such as the one found by Jimmy, were used to exchange information. Soviet agents agreed on inconspicuous locations called "dead drops" in which they placed secret containers such as hollow coins [2].

The only remaining piece of the puzzle was to decrypt the message that was inside the coin. Häyhänen thoroughly explained the cipher that was used to encrypt the message on the microfilm inside the nickel [2]. It was encrypted using a Nihilist cipher called VIC [17]. The Nihilists were a Russian group that opposed the Russian tsar. In the 1880s, the Nihilists used ciphers in order to communicate, which became known as the Nihilist ciphers [5]. In a bizarre twist of fate, the message in the nickel was actually intended for Häyhänen himself. After he accidentally spent the nickel, it traveled from person to person until it eventually landed into Jimmy's inquisitive hands [1].

The CIA has an excellent description of how VIC works [7]. Häyhänen presented this description at the 1957 trial of Colonel Rudolf Abel. Since the CIA states that those attending the trial were either bored or confused by the description of VIC, I will spare you the details. Instead, I will describe the specific aspects of VIC that provided the inspiration for VICCard.

2.2 The First Version of VICCard

To pay homage to the mind-numbing security of VIC, I have decided to entitle my original cipher VICCard. VICCard is an original playing card cipher that combines numerous cryptographic strategies together. The basic strategy of VIC is to use a checkerboard to convert letters to cards and then to perform various operations on these cards. With VICCard, we are using a similar strategy. There are four steps to this cipher. First, use a checkerboard to convert the letters of the message into cards. Second, perform columnar transpositions on these cards. Third, apply lagged Fibonacci generators to these cards. Finally, use the same checkerboard to convert the cards back into letters. Although VICCard has gone through multiple versions, these four steps remained the fundamental structure. In order to demonstrate each of these steps, we will follow cryptographic tradition and encrypt the message `Attack At Dawn` as an example. The following deck of cards will be our key. In our notation,

$9\diamondsuit$ is the top card and $A\heartsuit$ is the bottom card when the deck is held face up.

[$A\heartsuit$, $7\heartsuit$, $K\clubsuit$, $8\heartsuit$, $K\diamondsuit$, $J\clubsuit$, $K\spadesuit$, $K\heartsuit$, $4\spadesuit$, $8\diamondsuit$, $4\heartsuit$, $7\clubsuit$, $3\clubsuit$, $10\diamondsuit$, $Q\heartsuit$, $10\clubsuit$, $5\diamondsuit$, $2\spadesuit$, $J\spadesuit$, $A\clubsuit$, $9\clubsuit$, $4\clubsuit$, $3\diamondsuit$, $3\heartsuit$, $8\clubsuit$, $7\diamondsuit$, $5\spadesuit$, $5\heartsuit$, $2\clubsuit$, $A\diamondsuit$, $8\spadesuit$, $10\spadesuit$, $6\heartsuit$, $9\spadesuit$, $10\heartsuit$, $6\diamondsuit$, $Q\diamondsuit$, $6\clubsuit$, $2\diamondsuit$, $J\diamondsuit$, $7\spadesuit$, $5\clubsuit$, $4\diamondsuit$, $J\heartsuit$, $Q\spadesuit$, $6\spadesuit$, $3\spadesuit$, $Q\clubsuit$, $9\heartsuit$, $2\heartsuit$, $A\spadesuit$, $9\diamondsuit$]

2.3 Step 1: Converting Letters to Cards

We will have cards represent letters according to the following table. Notice that the lowercase letters are represented by the black cards and that the uppercase letters are represented by the red cards.

	Spades (♠)												
Card	A	2	3	4	5	6	7	8	9	10	J	Q	K
Letter	a	b	c	d	e	f	g	h	i	j	k	l	m
	Clubs (♣)												
Card	A	2	3	4	5	6	7	8	9	10	J	Q	K
Letter	n	o	p	q	r	s	t	u	v	w	x	y	z
	Hearts (♡)												
Card	A	2	3	4	5	6	7	8	9	10	J	Q	K
Letter	A	B	C	D	E	F	G	H	I	J	K	L	M
	Diamonds (♢)												
Card	A	2	3	4	5	6	7	8	9	10	J	Q	K
Letter	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Table 1: Letter encoding for VICCard

In order to convert the letters of the plaintext into cards, we will use a checkerboard. The checkerboard in Table 2 is created by dealing the cards into 4 columns of 13 cards.

Suppose that we are encrypting the plaintext `Attack At Dawn`. We will convert these letters to cards one letter at a time. We will start with the letter `A`, which is represented by $A\heartsuit$. First, I find $A\heartsuit$ in the checkerboard. This card is in the **A** row and the ♣ column. Hence, `A` encrypts to $A\clubsuit$. Next, we move onto the letter `t`, which is represented by $7\clubsuit$. This card is in the **Q** row and the ♣ column. Hence, `t` encrypts to $Q\clubsuit$. Continuing this pattern, the plaintext `Attack At Dawn` is converted to $A\clubsuit, Q\clubsuit, Q\clubsuit, Q\diamondsuit, 8\diamondsuit, 6\heartsuit, A\clubsuit, Q\clubsuit, J\clubsuit, Q\diamondsuit, 3\heartsuit, 7\heartsuit$.

Typically with effective cryptographic checkerboards, each plaintext letter is represented by 2 or more numbers. Instead of using this string of cards, we will break up the cards into two rows. The first row is all of the face values of the cards, and the second row is all of the suits. The face values are represented with the numbers 0 through 12, and the suits are represented with the numbers 0 through 3.

At this point, it is important to take notice of a security feature of this checkerboard. Notice that every letter is represented by a black card and a red card. The black card

	♣ (0)	♡ (1)	♠ (2)	♢ (3)
A (1)	A♡	10♢	5♠	J♢
2 (2)	7♡	Q♡	5♡	7♠
3 (3)	K♣	10♣	2♣	5♣
4 (4)	8♡	5♢	A♢	4♢
5 (5)	K♢	2♠	8♠	J♡
6 (6)	J♣	J♠	10♠	Q♠
7 (7)	K♠	A♣	6♡	6♠
8 (8)	K♡	9♣	9♠	3♠
9 (9)	4♠	4♣	10♡	Q♣
10 (10)	8♢	3♢	6♢	9♡
J (11)	4♡	3♡	Q♢	2♡
Q (12)	7♣	8♣	6♣	A♠
K (0)	3♣	7♢	2♢	9♢

Table 2: Checkerboard from Deck Order

`A`	`t`	`t`	`a`	`c`	`k`	`A`	`t`	`D`	`a`	`w`	`n`
A♣	Q♣	Q♣	Q♢	8♢	6♡	A♣	Q♣	J♣	Q♢	3♡	7♡
1	12	12	12	8	6	1	12	11	12	3	7
0	0	0	3	3	1	0	0	0	3	1	1

Table 3: Letters Converted to Face Values and Suits

represents the lowercase version, and the red card represents the uppercase version. In the above example, I used the black cards to represent each lowercase letter and the red cards to represent each uppercase letter. However, I did not have to do this. For each plaintext letter, we can either use the black card or the red card to encrypt it. For example, instead of using A♡ for the letter `A`, we can instead use the black card option A♠. This is equivalent to regularly encrypting the letter `a`. Similarly, instead of encrypting `t` as Q♣, we can encrypt it as Q♢. This is exactly like regularly encrypting the letter `T`. In other words, the option of choosing either the black card or the red card for each letter is equivalent to the option of changing the case of each letter. For example, based on how we choose black and red cards, we can encrypt the message `Attack At Dawn` as `aTtacK AT daWN` . When the decoder reverses this process, he will get the latter plaintext message. The letters are in the wrong cases, but it is still readable. Hence, having this choice for each letter does not compromise the message.

This feature has great potential for increasing security. Suppose that we have a plaintext of N letters. Since there are two choices for each letter, a black card or a red card, the encoder has 2^N possible ways of using the same deck to encrypt a particular message. Recall that there are about $2^{225.58}$ possible decks. Combining these two together, there are $2^{225.58+N}$ possible ways of encrypting the same message. In other words, every letter in the plaintext adds a bit to the pool of possible keys.

2.4 Step 2: Columnar Transpositions

In Step 1, we performed a substitution: cards were substituted for letters. In Step 2, we will perform a transposition. Here, none of the numbers are going to be altered. Instead, they are going to be rearranged via a columnar transposition. We will use two columnar transpositions: one for the row of face values and one for the row of suits. Here is how we perform columnar transpositions. First, we need a key for each transposition. We will use the order of the clubs for the face values: [$K\clubsuit$, $J\clubsuit$, $7\clubsuit$, $3\clubsuit$, $10\clubsuit$, $A\clubsuit$, $9\clubsuit$, $4\clubsuit$, $8\clubsuit$, $2\clubsuit$, $6\clubsuit$, $5\clubsuit$, $Q\clubsuit$]. Also, we will use the order of the hearts excluding the king for the suits: [$A\heartsuit$, $7\heartsuit$, $8\heartsuit$, $4\heartsuit$, $Q\heartsuit$, $3\heartsuit$, $5\heartsuit$, $6\heartsuit$, $10\heartsuit$, $J\heartsuit$, $9\heartsuit$, $2\heartsuit$]. To transpose the face values, we create a grid with the clubs on top. Then, we fill in the grid from left to right with the face values.

$K\clubsuit$	$J\clubsuit$	$7\clubsuit$	$3\clubsuit$	$10\clubsuit$	$A\clubsuit$	$9\clubsuit$	$4\clubsuit$	$8\clubsuit$	$2\clubsuit$	$6\clubsuit$	$5\clubsuit$	$Q\clubsuit$
1	12	12	12	8	6	1	12	11	12	3	7	

Next, we read the face values out of the grid from top to bottom based on the numerical order of the clubs. We first read 6 from the $A\clubsuit$ column, 12 from the $2\clubsuit$ column, 12 from the $3\clubsuit$ column, and so forth to get the following new row of face values: (6 12 12 12 7 3 12 11 1 8 12 1). Similarly, to perform the transposition of the suits we create a grid with the hearts on top. Then, we again fill in the grid from left to right.

$A\heartsuit$	$7\heartsuit$	$8\heartsuit$	$4\heartsuit$	$Q\heartsuit$	$3\heartsuit$	$5\heartsuit$	$6\heartsuit$	$10\heartsuit$	$J\heartsuit$	$9\heartsuit$	$2\heartsuit$
0	0	0	3	3	1	0	0	0	3	1	1

Reading out the suits from the top to bottom based on the numerical order of the suits, we get the following new row of suits: (0 1 1 3 0 0 0 0 1 0 3 3). In summary, performing these columnar transpositions gives us the following two new rows of face values and suits:

6	12	12	12	7	3	12	11	1	8	12	1
0	1	1	3	0	0	0	0	1	0	3	3

2.5 Step 3: Lagged Fibonacci Generators

The third step of VICCard is to use two lagged Fibonacci generators. Frequently in cryptography, we employ the help of random strings of numbers. However, the problem with using humongous strings of random numbers is that they are wildly impractical. Instead, it is more common to use pseudorandom strings of numbers. These are strings of numbers that appear to be random and have a lot of the properties of randomness, even though they were not created in a purely random way.

A lagged Fibonacci generator is one such method of creating a pseudorandom string of numbers. Instead of sharing the entire string of numbers, the sender and receiver only share

a small string of a few digits. This is called the seed. For example, suppose that we are using the first five digits of π as the seed: (3 1 4 1 5). Here is the procedure for creating an indefinitely long string of numbers by using a lagged Fibonacci generator. We begin with the first two numbers: 3 and 1. We add these together to get 4, and we attach this number to the end of the seed: (3 1 4 1 5 4). Next, we move onto the next two numbers: 1 and 4. Adding these together gives us 5, which we attach to the end of the previous string of numbers: (3 1 4 1 5 4 5). Again, we add the next two numbers (4 and 1) to get 5, which is again attached to the end: (3 1 4 1 5 4 5 5). Continuing this process indefinitely, we can create a pseudorandom string of numbers of any desired length. Also, it is important to note that this addition is performed modulo 10. This means that if adding two numbers produces a number that is greater than 10, we divide this number by ten and use the remainder. For example, adding 5 and 7 gives us 12, which is 2 in modulo 10.

The convenience of a lagged Fibonacci generator is rooted in the fact that the sender and receiver only need to share the seed. In order to do this with VICCard, we will encode two seeds in the deck. The seed for the lagged Fibonacci generator of the face values is encoded in the order of the spades in the deck. The order of the spades in the current keyed deck is [$K\spadesuit$, $4\spadesuit$, $2\spadesuit$, $J\spadesuit$, $5\spadesuit$, $8\spadesuit$, $10\spadesuit$, $9\spadesuit$, $7\spadesuit$, $Q\spadesuit$, $6\spadesuit$, $3\spadesuit$, $A\spadesuit$]. The order of the face values of these cards yields the following seed: (13 4 2 11 5 8 10 9 7 12 6 3 1). As one more adjustment, we will represent the 13, which came from $K\spadesuit$, as 13 mod 13 = 0. Hence, the seed for the lagged Fibonacci generator of the face values is (0 4 2 11 5 8 10 9 7 12 6 3 1).

Similarly, the seed for the lagged Fibonacci generator of the suits is encoded in the order of the face values of the diamonds. The order of the diamonds in the current keyed deck is [$K\diamondsuit$, $8\diamondsuit$, $10\diamondsuit$, $5\diamondsuit$, $3\diamondsuit$, $7\diamondsuit$, $A\diamondsuit$, $6\diamondsuit$, $Q\diamondsuit$, $2\diamondsuit$, $J\diamondsuit$, $4\diamondsuit$, $9\diamondsuit$]. This gives us the following seed: (13 8 10 5 3 7 1 6 12 2 11 4 9). Since there are only four suits in a deck, we will express this seed in modulo 4. Hence, the seed for the lagged Fibonacci generator of the suits is (1 0 2 1 3 3 1 2 0 2 3 0 1).

We will now add these two string of numbers to the rows of face values and suits using modulo 13 and modulo 4 arithmetic, respectively. In this case, the plaintext is small enough so that we do not have to generate any more numbers. If the plaintext were longer, we would use each seed to create new numbers as detailed above. Adding the numbers from the spade lagged Fibonacci generator to the row of face values, we get the following:

	6	12	12	12	7	3	12	11	1	8	12	1
+	0	4	2	11	5	8	10	9	7	12	6	3
=	6	3	1	10	12	11	9	7	8	7	5	4

Adding the numbers from the diamond lagged Fibonacci generator to the row of suits, we get the following:

	0	1	1	3	0	0	0	0	1	0	3	3
+	1	0	2	1	3	3	1	2	0	2	3	0
=	1	1	3	0	3	3	1	2	1	2	2	3

This has the effect of continuing to randomize the face values and suits independently. In total, this gives us the following two new rows of face values and suits:

6	3	1	10	12	11	9	7	8	7	5	4
1	1	3	0	3	3	1	2	1	2	2	3

2.6 Step 4: Converting Cards Back into Letters

The final step in the VICCard cipher is to convert these two rows of numbers back into letters. In order to do this, we will use Table 2 and reverse the algorithm of Step 1. We will start with the first column of numbers, which contains a face value of 6 and a suit of 1 (♡). This tells us to look at the card in the sixth row and the ♡'s column of the checkerboard, which is $J\spadesuit$. Since $J\spadesuit$ represents the letter `k`, the first letter of the ciphertext is `k`. Next, the second column of numbers tells us to look at the card in the third row and the ♡'s column, which is $10\clubsuit$. This card represents the letter `w`, meaning that the next letter of the ciphertext is `w`. Continuing this pattern, we get the following ciphertext: `kwXUaB qF vFhQ`.

2.7 Summary: The Cards as a Key Container

Something that you might have noticed about VICCard is that playing cards are technically not needed to perform it. Using substitution checkerboards, columnar transpositions, and lagged Fibonacci generators are not unique to playing cards. In fact, this cipher can be entirely executed using numbers instead of face values and suits. However, suppose that we do not use playing cards to execute VICCard. Here, the one sending the message and the one receiving the message must share a remarkable amount of information. They must both know the order of letters in the checkerboard, the keywords used for the transpositions, and the seeds for the lagged Fibonacci generators. The reason for executing this cipher with playing cards is because all of this information is compactly contained in 52 playing cards. This way, the sender and receiver must only share the deck order.

Now that we have used cryptography to create VICCard, the next step is to use cryptanalysis to analyze its security. We will examine each element of VICCard: the substitution checkerboard, the columnar transpositions, and the lagged Fibonacci generators.

3 Substitution Checkerboard

Basic cryptographic substitutions can be found in cryptogram puzzle books. In these books, every English letter is represented by another letter. For example, every `e` is replaced with `w` and every `x` is replaced by `c`. There are $26! \approx 4.03 \times 10^{26}$ possible ways to substitute each English letter for another English letter. At first glance, it seems that a standard cryptogram is remarkably secure. However, if that were true, cryptogram books would not be available in giant puzzle books alongside Sudoku and crosswords. The insecurity of cryptograms is best exemplified by two common cryptanalysis methods. First, analyzing the frequency distribution of English letters is a useful technique of decoding a cryptogram. For example, `e` is the most common English letter. Since `e` encrypts to `w` in the above example, it is likely that `w` will be the most common letter in the ciphertext. Hence, a code breaker could deduce from the high frequency of `w` that it represents `e`. A second attack is the use of plaintext cribs. Whereas the first attack exploits the frequencies of certain letters, cribs are frequently occurring words. For example, if I see a letter by itself in the ciphertext, it is likely that it represents the letter `a`. Similarly, if I see a three letter word in the ciphertext, it is likely that this is either the word `the` or a pronoun. Hence, there is a high chance that I can ascertain the identities of three letters.

Instead of a basic one-to-one substitution, a more secure method is to represent each plaintext character by two or more characters. This technique is know as fractionation, and it is present in ciphers such as Bifid, Trifid, and straddling checkerboard [18]. With Bifid, each English letter is represented by two numbers. We substitute each letter in the plaintext with two numbers and shuffle these numbers around. Then, we substitute each pair of numbers in the final sequence for their English letter equivalents. Trifid substitutes three numbers for each English letter, and a straddling checkerboard encrypts some letters with one number and some letters with two numbers.

The substitution checkerboard in VICCard follows an approach that is similar to Bifid. Each letter is substituted for two numbers; one number represents a face value, and the other number represents a suit. However, an important difference is the ability of VICCard to perform two different substitutions for each letter. We can either use the corresponding red uppercase card or the corresponding black lowercase card. With Bifid and similar substitution checkerboards, we always encrypt a letter with the same group of numbers. Here, VICCard can encrypt the exact same letter in two different ways. This is especially useful when messages use the same letter numerous times throughout the message (such as the `a`'s in `Attack At Dawn`) or when words contain two of the same letter that are next to each other (such as the two `t`'s in `Attack`). In Bifid, a code breaker has a $\frac{1}{26}$ probability of correctly guessing the letter represented by the two numbers. In VICCard, the code breaker has to guess the two cards that represent the same letter, which he has a $\frac{1}{26} \times \frac{1}{26} = \frac{1}{676}$ probability of doing correctly. This further complicates the code breaker's task without severely complicating the encryption process.

4 Columnar Transpositions

On its own, the double columnar transposition is an alluring cipher: it is easy to learn, fast to implement, and not so trivial to break. In attempting to successfully break this cipher, the first option that comes to mind is simply testing every possible keyword [8]. For a nine column transposition, there are $9! = 362,880$ possible keys. A computer can quickly move through each of these keys, easily cracking the cipher. This is why columnar transpositions are typically performed in pairs. There are $(9!)^2 = 131,681,894,400$ possible permutations with a pair of nine column transpositions. A second possible attack is a dictionary attack [8]. Here, the code breaker has a database of about 1 million frequently used keywords, such as names of prominent historical figures. He then tests each keyword to see whether it successfully decodes the message. Yet a third strategy is hill climbing [8]. This involves picking a starting keyword and gradually making small changes to this keyword, such as swapping letters. If the new keyword seems to decode the message better, then this keyword replaces the starting one. With hill climbing, we continue to make these changes until we find the keyword.

A vital feature of all these strategies is that keywords are guessed until the ciphertext is undone in such a way that it "makes sense." This is why transpositions are frequently combined with substitutions. Consider when the letters in a plaintext message are substituted in some way for others. After this, the two columnar transpositions are performed. This complicates these common code-breaking techniques because now it is impossible for any reversal of the transpositions to "make sense." This is why VICCard combines substitutions with transpositions. In fact, VICCard uses three substitutions: initially converting letters to cards, applying lagged Fibonacci generators, and finally converting cards back into letters. Hence, the double columnar transposition in VICCard is valuable in and of itself. However, it becomes remarkably strong when combined with the other cryptographic techniques.

5 Lagged Fibonacci Generator

In assessing the security of a lagged Fibonacci generator, two features must be analyzed. First, what is the period? In other words, how many numbers does the lagged Fibonacci generator create before it starts repeating? In emulating true randomness, we do not want a string of numbers that repeats. Hence, we desire a lagged Fibonacci generator with a large period. Specifically, the security of a lagged Fibonacci generator is maximized when its period is larger than the length of the plaintext. Second, does the distribution of the numbers closely resemble the distribution produced by randomness? For example, there are four different numbers in the lagged Fibonacci generator of the suits. In a truly random string of 4 different numbers, each number occurs approximately $\frac{1}{4}$ of the time. Hence, this lagged Fibonacci generator resembles a random distribution if the 0's, 1's, 2's, and 3's all occur approximately $\frac{1}{4}$ of the time.

5.1 Modulo 4 Lagged Fibonacci Generator

In analyzing the security of the modulo 4 lagged Fibonacci generator, the seed of which is the order of the diamonds, I first created all the possible seeds. These periods each have three 0's, three 2's, three 4's, and four 1's. Hence, there are $\binom{13}{3}\binom{10}{3}\binom{7}{3} = 1,201,200$ seeds to consider. As a result, it was necessary to write a program that created each of these seeds and placed them into text files. After this, I created a program in order to ascertain the period of each seed. It did this by reading each seed in from the text files, used each seed to create a string of about 100,000 numbers, and searched the string of numbers to see when the string began to repeat. Analyzing all the seeds in this way, it became clear that there are three possible periods. 23 seeds have a period of 62, 1019 seeds have a period of 510, and the remaining 1,200,158 seeds have a period of 15,810. Overall, this is very good news: $\frac{1,200,158}{1,201,200} \approx 99.9\%$ of the seeds have a respectable period of 15,810. 15,810 characters can fill almost eight pages of a Word document in MLA format, assuming that there are no spaces. This is more than what is needed to send a typical encoded message.

Once I knew the period of each seed, I then determined the distribution of 0's, 1's, 2's, and 3's produced by each seed. I accomplished this by writing a program which used each seed to produce a string of numbers until right before it started repeating. In other words, the program produced strings of 62 numbers from the seeds that have a period of 62, produced strings of 510 numbers from the seeds that have a period of 510, and so forth. It then went through each string of numbers and counted the number of occurrences of 0's, 1's, 2's, and 3's. On average, each seed created a distribution that very closely approximated 25% of 0's, 1's, 2's, and 3's. Table 4 shows the average number of 0's, 1's, 2's, and 3's for each period.

Seeds with a period of 62			
Number of 0's	Number of 1's	Number of 2's	Number of 3's
15.3913	16.8696	14.6087	15.1304
Seeds with a period of 510			
Number of 0's	Number of 1's	Number of 2's	Number of 3's
127.421	128.495	126.579	127.505
Seeds with a period of 15,810			
Number of 0's	Number of 1's	Number of 2's	Number of 3's
3952.47	3953.32	3951.8	3952.4

Table 4: Average Distribution for the Modulo 4 lagged Fibonacci generator

Table 5 shows the percent error of each average relative to 25% of the period. As the period increases, the percent error decreases.

5.2 Modulo 13 Lagged Fibonacci Generator

In order to analyze the security of the modulo 13 lagged Fibonacci generator, which is encoded in the order of the spades, I followed a similar process. I first examined the

Seeds with a period of 62			
Percent Error of 0's	Percent Error of 1's	Percent Error of 2's	Percent Error of 3's
-0.7%	8.8%	-5.8%	-2.4%
Seeds with a period of 510			
Percent Error of 0's	Percent Error of 1's	Percent Error of 2's	Percent Error of 3's
-0.06196%	0.7804%	-0.7224%	0.003922%
Seeds with a period of 15,810			
Percent Error of 0's	Percent Error of 1's	Percent Error of 2's	Percent Error of 3's
$-7.590 \times 10^{-4}\%$	$2.075 \times 10^{-2}\%$	$-1.771 \times 10^{-2}\%$	$-2.530 \times 10^{-3}\%$

Table 5: Percent Errors of the Distribution for the Modulo 4 lagged Fibonacci generator

periods and then analyzed the distributions. Before any of this could be done, however, I had to determine which seeds to test. There are $13! = 6,227,020,800$ possible seeds, making it impractical to test all of them. Instead, I chose a random sampling of seeds to test. Specifically, I analyzed all the seeds that start with (8 4 12 5), all the seeds that start with (12 6 9 8), all the seeds that start with (1 9 12 7), all the seeds that start with (3 2 11 7), and all the seeds that start with (3 1 5 2). There are $9! = 362,880$ seeds in each of these categories, meaning that I analyzed $5 \times 362,880 = 1,814,400$ seeds. Of course, before testing any of these seeds, I had to create five text files containing each of these categories of seeds. I did this by creating a program that created every possible permutation of any number of objects. Since the seed contains 13 numbers, I used the 13 setting of my program. Furthermore, I filled in the first four numbers for each category (like 8 4 12 5). This program then created every possible permutation of the nine remaining numbers and wrote all the seeds to a text file with the appropriate label. For example, the program wrote all seeds beginning with (8 4 12 5) to a file entitled "EightFourTwelveFive.txt." Once I compiled this respectable sample size, I began testing each seed.

Recall that most of the seeds of the modulo 4 lagged Fibonacci generator have a period of 15,810. We are not as concerned with the exact periods of the modulo 13 seeds. As long as the periods of the latter are larger than those of the former, then the periods of the modulo 13 seeds are secure for our current purposes. In order to test this, I created a program that took each seed in the previously created files, used each to create a string of slightly more than 15,810 numbers, and searched the string to see if the pattern repeated. Of all the 1,814,400 seeds that were tested, none of them repeated within 15,810 numbers. Just out of curiosity, I took a random seed (8 4 12 5 0 7 9 10 2 3 1 11 6) and tried to determine its period. After creating a string of 3,000,000 numbers, the pattern of numbers still did not repeat. As a result, we can conclude that the modulo 13 lagged Fibonacci generator has a respectable, secure period.

Next, I needed to make sure that the lagged Fibonacci generator produces a uniform distribution of the numbers 0 through 12. In order to do this, I created yet another program that used each seed to create a string of 13,000 numbers. It then counted the number of

occurrences of each number in each string. Tables 6 through 11 show the average results for each of the five groups of seeds and the overall average for all 1,814,400 seeds.

Seeds That Begin with 8 4 12 5	
Number	Average Number of Occurrences
0	1003.31
1	998.494
2	1003.03
3	999.091
4	1001.6
5	998.142
6	998.786
7	1001.39
8	998.25
9	998.963
10	999.007
11	1000.6
12	999.329

Table 6: Average Distribution for Modulo 13 Lagged Fibonacci Generator (8 4 12 5)

Seeds That Begin with 12 6 9 8	
Number	Average Number of Occurrences
0	999.686
1	999.94
2	998.537
3	1001.64
4	1000.73
5	999.861
6	1001.15
7	1000.81
8	999.764
9	1000.17
10	999.243
11	997.879
12	1000.6

Table 7: Average Distribution for Modulo 13 Lagged Fibonacci Generator (12 6 9 8)

If these strings of 13,000 numbers were truly random, we would expect approximately 1000 of each number. As demonstrated by these tables, the lagged Fibonacci generator in question provides an incredible approximation of this distribution. As a result, we can conclude that the modulo 13 lagged Fibonacci generator is safe to use.

Seeds That Begin with 1 9 12 7	
Number	Average Number of Occurrences
0	1002.66
1	999.659
2	999.788
3	999.473
4	1003.42
5	999.352
6	999.374
7	997.423
8	1000.33
9	998.763
10	999.384
11	1001.46
12	998.911

Table 8: Average Distribution for Modulo 13 Lagged Fibonacci Generator (1 9 12 7)

Seeds That Begin with 3 2 11 7	
Number	Average Number of Occurrences
0	1000.46
1	997.528
2	1000.79
3	999.103
4	999.776
5	1004.38
6	998.065
7	1001.08
8	999.167
9	1001.5
10	999.008
11	1000.48
12	998.668

Table 9: Average Distribution for Modulo 13 Lagged Fibonacci Generator (3 2 11 7)

6 The Updated Version of VICCard

The above description of VICCard is the first version of the cipher that was used to outline the basic structure. It is composed of the following basic steps:

1. Convert the plaintext letters to cards using the checkerboard.
2. Perform one columnar transposition on both the face value row and the suit row.
3. Add one pseudorandom string of numbers to both the face value row and the suit row.
4. Use the same checkerboard in Step 1 to convert the cards back into letters.

Seeds That Begin with 3 1 5 2	
Number	Average Number of Occurrences
0	1000.46
1	998.955
2	1000.42
3	998.035
4	1001.65
5	998.509
6	1000.09
7	1003.48
8	1001.21
9	998.733
10	999.764
11	997.494
12	1001.19

Table 10: Average Distribution for Modulo 13 Lagged Fibonacci Generator (3 1 5 2)

Results for all 1,814,400 Seeds		
Number	Average Number of Occurrences	Percent Error
0	1001.3152	0.13152%
1	998.9152	-0.10848%
2	1000.513	0.0513%
3	999.4684	-0.05316%
4	1001.4352	0.14352%
5	1000.0488	0.00488%
6	999.493	-0.0507%
7	1000.8366	0.08366%
8	999.7442	-0.02558%
9	999.6258	-0.03742%
10	999.2812	-0.07188%
11	999.5826	-0.04174%
12	999.7396	-0.02604%

Table 11: Average Distribution for Modulo 13 Lagged Fibonacci Generator Overall

In an attempt to make improvements, VICCard has gone through multiple versions. This basic four-step structure remains the same throughout each update. The following details the fifth version of the cipher VICCard 5.0, which is the most recent update. This version uses three additional cryptographic strategies.

6.1 Plaintext Preparations

First, I added two small options to prepare the plaintext before encryption. First, I added the option of placing junk letters at both the beginning and the end of the message. Second, the encoder can "cut" the message before encrypting it. This is similar to cutting a deck of cards. In order to mark where he cuts the cards, he should place a marker, such as `xx`, where the intelligible words begin. Let's apply these two strategies to encrypting `Attack At Dawn`. First, we add random characters to the beginning and the end to get `EdfFxxAttackAtDawndsjfSDRmk`. Second, we will "cut" the message at the `k` in `Attack`, giving us `kAtDawndsjfSDRmkEdfFxxAttac`. Now, we can move on to the four steps of VIC-Card 5.0 to encrypt the message. When the decoder decrypts the ciphertext, he will get `kAtDawndsjfSDRmkEdfFxxAttac` as his message. In order to read it, all he needs to do is "cut" the marker `xx` to the end: `AttackAtDawndsjfSDRmkEdfFxx`. Removing the junk letters at the end, he can now read the message: `AttackAtDawn dsjfSDRmkEdfFxx`.

6.2 Triangular Columnar Transpositions

In the first version of VICCard, exactly one columnar transposition is performed on the face values and the suits. However, to increase security, columnar transpositions are typically performed in pairs. Hence, the current version of VICCard applies double columnar transpositions to the face values and to the suits. This is a total of four columnar transpositions. Thus, we will use the following four keys. For the face values, the orders of $A\clubsuit$ through $7\clubsuit$ and of $A\heartsuit$ through $6\heartsuit$ shall be the keys for the first and second transpositions, respectively. For the suits, the orders of $7\heartsuit$ through $K\heartsuit$ and of $8\clubsuit$ through $K\clubsuit$ shall be the keys for the first and second transpositions, respectively.

Furthermore, following the pattern of VIC, we shall make the second transposition a little different. For the second transposition, the VIC cipher reads numbers out of the grid in exactly the same way: from the top to the bottom based on the keyword. However, VIC places numbers into the grid differently.

For example, we will use the following order of six hearts to perform a face value transposition: [$5\heartsuit$, $6\heartsuit$, $3\heartsuit$, $A\heartsuit$, $2\heartsuit$, $4\heartsuit$]. Before filling the transposition grid, we will use the key to divide the grid into triangular sections. We will represent these sections by filling them with the letter **T**. We will start with the smallest number of the key, which is 1. We section off all cells to the right of and including the part of the grid under the 1. We repeat this process on the next row, starting at the next column over. We continue to section off cells until we run out of columns, producing a triangular pattern.

We do this for all numbers in the key. The following grid shows the results of sectioning off cells based on the numbers 1-4.

We will transpose the following sequence of face values: (4 0 8 12 5 1 6 8 9 0 1 9 1 8 7 11 12 8 9 6 3 2 2 12 7 0 12 7). Since there are 28 face values, we know that we will completely

5	6	3	1	2	4
			T	T	T
				T	T
					T
				T	T
					T
		T	T	T	T
			T	T	T
				T	T
					T
					T

fill four rows and partially fill a fifth row. With this in mind, we begin by filling in all the cells that have not been sectioned off.

5	6	3	1	2	4
4	0	8	T	T	T
12	5	1	6	T	T
8	9	0	1	9	T
1	8	7	11	T	T
12	8	9	6		T

Now, we place the rest of the face values in the sectioned off cells.

5	6	3	1	2	4
4	0	8	3	2	2
12	5	1	6	12	7
8	9	0	1	9	0
1	8	7	11	12	7
12	8	9	6		

Finally, we finish the transposition by reading the numbers out from the grid: (3 6 1 11 6 2 12 9 12 8 1 0 7 9 2 7 0 7 4 12 8 1 12 0 5 9 8 8).

This unique transposition contributed to the remarkable security of VIC. As a result, it makes sense to apply it to VICCard. With VICCard 5.0, the first transposition for the face values and the suits is a normal columnar transposition. By contrast, the second transposition for the face values and the suits is this special transposition.

6.3 Junk Letters and Diffusion

Finally, these last two cryptographic strategies are performed before each columnar transposition. First, VICCard 5.0 adds random face values and suits during the columnar transpositions. For example, we will work with the following rows of face values and suits:

1	12	12	12	8	6	1	12	11	12	3	7
0	0	0	3	3	1	0	0	0	3	1	1

Consider the face values. We will perform the following columnar transposition with seven columns:

7	3	1	4	2	6	5
1	12	12	12	8	6	1
12	11	12	3	7		

Notice that the bottom row is partially filled. In the first version of VICCard, we performed the transposition with this partially filled row. In VICCard 5.0, we fill this bottom row with junk face values before completing the transposition. If the bottom row is completely filled, we will add an entire row of junk face values.

7	3	1	4	2	6	5
1	12	12	12	8	6	1
12	11	12	3	7	5	9

Now, we perform the columnar transposition: (12 12 8 7 12 11 12 3 1 9 6 5 1 12). Next, we will add four random face values to make the total number divisible by 6: (12 12 8 7 12 11 12 3 1 9 6 5 1 12 2 5 6 3). We then execute a special triangular columnar transposition with six columns:

1	4	3	5	6	2
7	12	11	12	3	1
12	9	6	5	1	12
12	8	2	5	6	3

These two transpositions produce the following string of face values: (7 12 12 1 12 3 11 6 2 12 9 8 12 5 5 3 1 6).

Notice that during the transpositions, the random numbers are spread throughout the stream of face values. When the decoder is undoing the transpositions, the random characters are easily eliminated. For example, undoing the above transposition produces the following stream of numbers: (12 12 8 7 12 11 12 3 1 9 6 5 1 12 2 5 6 3). Currently, the random characters are at the end of the message. However, the decoder still needs to determine how many random characters there are. All they have to do is divide the string of numbers into sections of seven numbers and eliminate what remains: (12 12 8 7 12 11 12 | 3 1 9 6 5 1 12 | 2 5 6 3). The reason for this is that the first columnar transposition produces a string of numbers that is divisible by 7. Since the second columnar transposition adds anywhere from 1 to 6 more numbers, all the leftover numbers must be junk letters.

Second, the last new security feature is a method of creating diffusion. Diffusion refers to when a given plaintext letter has an effect on how other plaintext letters are encrypted. Suppose that I am going to perform a seven-column transposition with the following string of face values: (12 5 3 7 5 8 9 3 11 10 6 7 4 8 5 3). There are sixteen numbers, which means that we must add 5 more random numbers. We will do that now before placing the face values into the transposition grid: (12 5 3 7 5 8 9 3 11 10 6 7 4 8 5 3 2 7 5 4 0). Before performing the transposition, we will divide this message into groups of seven: (12 5 3 7 5 8 9 | 3 11 10 6 7 4 8 | 5 3 2 7 5 4 0). We will add the numbers of the first group to those of the second group and add the new numbers of the second group to those of the third group, using modulo 13 addition. Adding the first group to the second group gives the following result: (12 5 3 7 5 8 9 | 2 3 0 0 12 12 4 | 5 3 2 7 5 4 0). Adding the second group to the third group gives the following result: (12 5 3 7 5 8 9 | 2 3 0 0 12 12 4 | 7 6 2 7 4 3 4). Now, we place these numbers into the grid to transpose them. This step is done before each transposition. We divide the numbers into groups of seven for the seven-column transpositions and groups of six for the six-column transpositions. Also, we use modulo 13 arithmetic for the face values and modulo 4 arithmetic for the suits. This increases diffusion because a change in any number will result in changes of multiple numbers after it when they are added together.

Full Example of VICCard 5.0

For the last time, we will encrypt our favorite message `Attack At Dawn`. With VICCard 5.0, we will perform quite a few modifications to the plaintext before encryption. First, we will change some of the cases `ATTacKaTdaWN`. Second, we will add some random letters to the beginning and the end `jWvxxATTacKaTdaWNMvHsi`. Third, we will cut the message at a random point `acKaTdaWNMvHsijWvxxATT`. Notice that we purposefully added to the two `x`'s to mark where the message begins. After all of this prep work, we can begin encryption.

Step 1: For Step 1, we use the substitution checkerboard method that was described in the first version. Here is the result of using the checkerboard to convert the plaintext into face values and suits:

a	c	K	a	T	d	a	W	N	M	v	H	s	i	j	W	v	x	x	A	T	T
12	8	5	12	0	9	12	1	4	8	8	4	12	8	6	1	8	6	6	1	0	0
3	3	3	3	1	0	3	1	2	0	1	0	2	2	2	1	1	0	0	0	1	1

Step 2:

As usual, we will start with the double columnar transposition of the face values. The first columnar transposition is nothing special. It is the exact same type of transposition that we have been doing. After adding six random face values to the end so the number of face values is divisible by seven (12 8 5 12 0 9 12 1 4 8 8 4 12 8 6 1 8 6 6 1 0 0 3 4 5 3 0 3), we create diffusion based on blocks of seven (12 8 5 12 0 9 12 0 12 0 7 4 8 7 6 0 8 0 10 9 7 6

3 12 5 0 9 10). Then we perform a plain columnar transposition based on the order of $A\clubsuit$ through $7\clubsuit$.

$7\clubsuit$	$3\clubsuit$	$A\clubsuit$	$4\clubsuit$	$2\clubsuit$	$6\clubsuit$	$5\clubsuit$
12	8	5	12	0	9	12
0	12	0	7	4	8	7
6	0	8	0	10	9	7
6	3	12	5	0	9	10

The result is (5 0 8 12 0 4 10 0 8 12 0 3 12 7 0 5 12 7 7 10 9 8 9 9 12 0 6 6).

The second transposition begins normally. We add two random face values so the total is divisible by six (5 0 8 12 0 4 10 0 8 12 0 3 12 7 0 5 12 7 7 10 9 8 9 9 12 0 6 6 7 7), and we create diffusion based on blocks of six letters (5 0 8 12 0 4 2 0 3 11 0 7 1 7 3 3 12 1 8 4 12 11 8 10 7 4 5 4 2 4). Before placing the face values in the grid, however, we need to create triangular sections based on the key, which is the order of $A\heartsuit$ through $6\heartsuit$. Since we are only dealing with 30 face values, we only have to worry about the top five rows.

$A\heartsuit$	$4\heartsuit$	$3\heartsuit$	$5\heartsuit$	$6\heartsuit$	$2\heartsuit$
T	T	T	T	T	T
	T	T	T	T	T
		T	T	T	T
			T	T	T
				T	T

We first fill in the parts of the grid that are not sectioned off,

$A\heartsuit$	$4\heartsuit$	$3\heartsuit$	$5\heartsuit$	$6\heartsuit$	$2\heartsuit$
T	T	T	T	T	T
5	T	T	T	T	T
0	8	T	T	T	T
12	0	4	T	T	T
2	0	3	11	T	T

and then fill in the triangular sections:

$A\heartsuit$	$4\heartsuit$	$3\heartsuit$	$5\heartsuit$	$6\heartsuit$	$2\heartsuit$
0	7	1	7	3	3
5	12	1	8	4	12
0	8	11	8	10	7
12	0	4	4	5	4
2	0	3	11	2	4

The result is: (0 5 0 12 2 3 12 7 4 4 1 1 11 4 3 7 12 8 0 0 7 8 8 4 11 3 4 10 5 2).

Now will we do the same thing for the suits. We add five random suits so the total is divisible by seven (3 3 3 3 1 0 3 1 2 0 1 0 2 2 2 1 1 0 0 0 1 1 3 3 2 0 0 2), create diffusion based on blocks of seven (3 3 3 3 1 0 3 0 1 3 0 1 2 1 2 2 0 0 1 2 2 3 1 3 2 1 2 0), and then transpose the suits based on the order of 7♡ through K♡. The result is (3 0 2 3 3 1 2 1 3 1 2 0 1 1 1 1 0 2 2 2 3 0 0 2 3 3 0 3).

7♡	8♡	K♡	Q♡	10♡	J♡	9♡
3	3	3	3	1	0	3
0	1	3	0	1	2	1
2	2	0	0	1	2	2
3	1	3	2	1	2	0

In preparation for the second transposition, we add two random suits so the total is divisible by six (3 0 2 3 3 1 2 1 3 1 2 0 1 1 1 1 0 2 2 2 3 0 0 2 3 3 0 3 1 1), create diffusion based on blocks of six letters (3 0 2 3 3 1 1 1 1 0 1 1 2 2 2 1 1 3 0 0 1 1 1 1 3 3 1 0 2 2), and create triangular sections in the grid based on the order of 8♣ through K♣. Since there are 30 suits, we only are concerned with the first five rows of the grid.

K♣	J♣	10♣	9♣	8♣	Q♣
				T	T
					T
			T	T	T
				T	T
					T

We then fill the grid based on the triangular sections and transpose the suits.

K♣	J♣	10♣	9♣	8♣	Q♣
3	0	2	3	1	1
3	1	1	1	1	1
0	1	1	3	3	1
2	2	2	1	0	2
1	3	0	0	1	2

The result is (1 1 3 0 1 3 1 3 1 0 2 1 1 2 0 0 1 1 2 3 1 1 1 2 2 3 3 0 2 1).
In summary, here are the two rows of face values and suits:

0	5	0	12	2	3	12	7	4	4	1	1	11	4	3	⋯
1	1	3	0	1	3	1	3	1	0	2	1	1	2	0	⋯

⋯	7	12	8	0	0	7	8	8	4	11	3	4	10	5	2
⋯	0	1	1	2	3	1	1	1	2	2	3	3	0	2	1

Step 3: For Step 3, we add the numbers from the spade lagged Fibonacci generator to the face values,

	0	5	0	12	2	3	12	7	4	4	1	1	11	4	3	$\cdots$
+	13	4	2	11	5	8	10	9	7	12	6	3	1	4	6	$\cdots$
=	0	9	2	10	7	11	9	3	11	3	7	4	12	8	9	$\cdots$

	$\cdots$	7	12	8	0	0	7	8	8	4	11	3	4	10	5	2
+	$\cdots$	0	3	0	5	6	3	6	5	9	4	5	10	6	3	3
=	$\cdots$	7	2	8	5	6	10	1	0	0	2	8	1	3	8	5

and we add the numbers from the diamond lagged Fibonacci generator to the suits,

	1	1	3	0	1	3	1	3	1	0	2	1	1	2	0	$\cdots$
+	1	0	2	1	3	3	1	2	0	2	3	0	1	1	2	$\cdots$
=	2	1	1	1	0	2	2	1	1	2	1	1	2	3	2	$\cdots$

	$\cdots$	0	1	1	2	3	1	1	1	2	2	3	3	0	2	1
+	$\cdots$	3	0	2	0	3	2	2	1	3	1	2	3	1	3	2
=	$\cdots$	3	1	3	2	2	3	3	2	1	3	1	2	1	1	3

which gives us the following face values and suits:

0	9	2	10	7	11	9	3	11	3	7	4	12	8	9	$\cdots$
2	1	1	1	0	2	2	1	1	2	1	1	2	3	2	$\cdots$

$\cdots$	7	2	8	5	6	10	1	0	0	2	8	1	3	8	5
$\cdots$	3	1	3	2	2	3	3	2	1	3	1	2	1	1	3

Step 4: Finally, we use the substitution checkerboard from Step 1 to convert these face values and suits into the ciphertext: `OqLPmYJwConRscJfLchjIXOTgvewvk`.

7 Randomness Tests

The fundamental concept of cryptography is randomness. The more unpredictable a cipher is, the harder it usually is to break. For example, recall our discussion of lagged Fibonacci generators. To ensure that the lagged Fibonacci generators of VICCard are a

reliable option, it was necessary to examine the randomness of the numbers that it created. We did this by focusing on the periods and the number distributions.

To test the randomness of VICCard 5.0, I encrypted six plaintexts: the Declaration of Independence, "Paul Revere's Ride" by Henry Wadsworth Longfellow, the Gettysburg Address, the lyrics to "All I Ask Of You", the opening to *A Tale of Two Cities*, and Psalm 23. After encrypting each plaintext using VICCard 5.0, I analyzed the letter distributions of the ciphertexts. Since we are using uppercase and lowercase letters, the ciphertexts are composed of 52 different letters. In a truly random string of letters, each of the letters will occur about $\frac{1}{52}$ of the time. Hence, if the six ciphertexts have a letter distribution that closely resembles a random distribution, then we have strong evidence in favor of the randomness of VICCard 5.0.

7.1 Chi-Square Test on Ciphertexts

At this point, I enlisted the help of the Chi-Square test of fitness. This test tells us how closely a set of measured data resembles the expected data. In this case, I used the Chi-Square test to determine how closely the measured distributions of the six ciphertexts "fit" with truly random texts. Here is how the Chi-Square test works. First, we calculate the Chi-Square value of a particular ciphertext using the following formula.

$$\chi^2 = \sum_{i=1}^{n} \left(\frac{(O_i - E_i)^2}{E_i} \right)$$

In this formula[1], n is the number of possible outcomes, O_i represents the observed number of occurrences of an outcome, and E_i represents the expected number of occurrences of an outcome. For example, consider the ciphertext of the Declaration of Independence. Since there are 52 types of letters, $n = 52$. The ciphertext has 6600 letters according to the breakdown in Table 12.

To calculate this ciphertext's Chi-Square value, we first calculate each of the individual Chi-Square terms. For example, consider the letter `a`. This letter occurs 105 times. This is the observed number of occurrences, O_i. In a truly random string of letters, the letter `a` would occur about $\frac{1}{52}$ of the time. Since there are 6600 letters in the ciphertext of the Declaration of Independence, the expected number of occurrences E_i is $\frac{6600}{52} \approx 126.923$. Using the formula above, the Chi-Square term for `a` is $\frac{(O_i - E_i)^2}{E_i} \approx \frac{(105 - 126.923)^2}{126.923} \approx 3.7867$. This calculation is performed for each of the 52 letters, and the final Chi-Square value is the sum of all these 52 calculations: $\chi^2 \approx 54.4558$.

What does χ^2 tell us? This number signifies how well the data emulates perfect randomness by measuring the level of deviation from perfect randomness. The larger the Chi-Square value, the more the data deviates. To determine the amount of deviation, we use the Chi-

[1] χ^2 is the symbol for the Chi-Square value.

Letter	Number of Occurrences	Letter	Number of Occurrences
a	105	A	118
b	147	B	112
c	129	C	126
d	119	D	145
e	139	E	150
f	136	F	120
g	122	G	117
h	117	H	127
i	118	I	117
j	122	J	109
k	132	K	144
l	114	L	135
m	126	M	138
n	130	N	133
o	126	O	148
p	134	P	154
q	126	Q	115
r	113	R	124
s	122	S	138
t	114	T	127
u	125	U	117
v	143	V	122
w	117	W	119
x	139	X	130
y	135	Y	112
z	117	Z	136

Table 12: Letter Distribution of the Declaration of Independence

Square value to calculate the associated p-value. Looking up the above χ^2 in a p-value table, we see that this data has a p-value of 0.3444. This p-value means the following. If I create a string of 6600 letters by choosing each letter at random, there is a 0.3444 probability of getting a string of letters that has a Chi-Square value of 54.4558. In other words, there is a 0.3444 probability that the Declaration of Independence ciphertext is random. The relevance of the p-value is in its ability to measure the level of randomness in a ciphertext.

In order to have a respectable sample size, I encrypted six plaintexts with six different keyed deck and performed a Chi-Square test on each ciphertext. Table 13 contains a summary of the test data.

After compiling this data, I encrypted these six plaintexts a second time, using a different keyed deck for each encryption. Table 14 contains the data from this second round of tests.

Plaintext (length)	Ciphertext Length	χ^2	p-value
Psalm 23 (461)	468	42.2222	0.8044
Opening to *A Tale of Two Cities* (475)	480	52.1333	0.4296
"All I Ask Of You" (813)	822	57.6983	0.2415
Gettysburg Address (1149)	1158	56.7703	0.2688
"Paul Revere's Ride" (4054)	4062	39.8552	0.8705
Declaration of Independence (6591)	6600	54.4558	0.3444

Table 13: First Round of Chi-Square Test Results

Plaintext (length)	Ciphertext Length	χ^2	p-value
Psalm 23 (461)	468	53.1111	0.3929
Opening to *A Tale of Two Cities* (475)	480	41.3000	0.8320
"All I Ask Of You" (813)	822	37.3285	0.9237
Gettysburg Address (1149)	1158	63.5060	0.1123
"Paul Revere's Ride" (4054)	4062	45.6928	0.6838
Declaration of Independence (6591)	6600	47.2703	0.6226

Table 14: Second Round of Chi-Square Test Results

7.2 Chi-Square Test on One-Time Pads

A second randomness test is a slight variation of the previous Chi-Square test. The first Chi-Square test measures the randomness of the ciphertexts. The second Chi-Square test measures the randomness of the associated one-time pads.

A one-time pad is a random string of numbers that is used to encrypt a message. For example, suppose that I want to encrypt `attack at dawn`. Since my plaintext is 12 letters long, I randomly choose a string of 12 numbers to be my one-time pad: 5 3 6 14 22 19 10 8 2 23 11 15. To encrypt the plaintext, I "add" the one-time pad to the plaintext. Since I cannot add numbers to letters, I first convert `attack at dawn` to numbers as per the following: `a` is represented with the number 1, `b` becomes 2, `c` becomes 3, and so forth. Now, I can add the one-time pad to the plaintext.

	a	t	t	a	c	k	a	t	d	a	w	n
	1	20	20	1	3	11	1	20	4	1	23	14
+	5	3	6	14	22	19	10	8	2	23	11	15
=	6	23	26	15	25	4	11	2	6	24	8	3
	f	w	z	o	y	d	k	b	f	x	h	c

Converting the sum back into letters gives us the ciphertext `fwzoyd kb fxhc`. The reason that we are concerned with one-time pads is because it is the only proven way to create perfect encryption. This is because the key is a truly random string of numbers, and there is no way to crack the cipher other than by trying every key by brute-force.

In this example, we applied a one-time pad to a plaintext in order to generate a ciphertext.

Consider doing the reverse process. If we take a ciphertext that we have generated and subtract the plaintext from it, we get the one-time pad that was used to encrypt the plaintext. However, let's say that the plaintext was not encrypted with a one-time pad. In this case, subtracting the plaintext from the ciphertext tells us that the encryption process has the same effect as the calculated one-time pad. If this one-time pad is sufficiently random, this would act as evidence in favor of the cipher's security.

For my second test, I encrypted the same six ciphertexts using six different keys. Instead of performing another Chi-Square test on the ciphertexts, I first subtracted the plaintexts from the ciphertexts in order to find the six one-time pads. I then carried out a Chi-Square test on each one-time pad. Table 15 lists the results. Since the plaintexts are slightly shorter than the ciphertexts, the one-time pads are the same length as the corresponding plaintexts.

Plaintext	χ^2	p-value
Psalm 23	49.1866	0.5460
Opening to *A Tale of Two Cities*	60.4358	0.1717
"All I Ask Of You"	50.5326	0.4921
Gettysburg	29.0783	0.9942
"Paul Revere's Ride"	32.1944	0.9817
Declaration of Independence	63.8245	0.1072

Table 15: First Round of Chi-Square Test Results (One-Time Pad)

Table 16 contains the results from a second round of tests in which the same six plaintexts were encrypted with six different decks.

Plaintext	χ^2	p-value
Psalm 23	62.0456	0.1382
Opening to *A Tale of Two Cities*	63.7200	0.1089
"All I Ask Of You"	47.2066	0.6251
Gettysburg	52.0688	0.4321
"Paul Revere's Ride"	40.7884	0.8463
Declaration of Independence	60.8264	0.1631

Table 16: Second Round of Chi-Square Test Results (One-Time Pad)

7.3 The Washington Test

In the previous tests, I encrypted different plaintexts with different deck keys. With this third test, I encrypted different plaintexts with the same keyed deck. I took the first 29,120 letters of George Washington's Farewell Address and divided them into 28 groups of 1040 letters. I then encrypted each group using the same deck to create 28 ciphertexts of length

1044. Table 17 contains the Chi-Square values and p-values of each ciphertext. The average p-value is 0.4614.

	χ^2	p-value
1	49.49425287	0.533610953
2	43.31800766	0.769024139
3	55.8697318	0.29694992
4	65.63218391	0.08162056
5	60.651341	0.166941401
6	51.5862069	0.450724746
7	64.13793103	0.10236771
8	59.75478927	0.187574286
9	58.65900383	0.215146577
10	45.01149425	0.709235389
11	36.5440613	0.936595831
12	48.79693487	0.561608159
13	49.59386973	0.529616108
14	52.38314176	0.420092715
15	35.8467433	0.946712769
16	53.57854406	0.375647609
17	42.81992337	0.785476088
18	64.9348659	0.090831388
19	63.54022989	0.111762895
20	68.72030651	0.049573757
21	52.38314176	0.420092715
22	55.47126437	0.309943572
23	45.90804598	0.675579632
24	45.21072797	0.701860096
25	62.84291188	0.123565899
26	47.20306513	0.625220692
27	40.92720307	0.842450584
28	38.53639847	0.900501941

Table 17: Chi-Square Test Results of Washington's Farewell Address

7.4 Interpreting the Results

The rationale for performing these tests is as follows. An English plaintext tends to have a predictable distribution. Letters such as `e` and `t` occur very frequently, whereas letters such as `q` and `z` are comparatively rare. These tests examine whether VICCard can transform a typical English letter distribution into something more like pure randomness.

Executing these Chi-Square tests provides reasonable evidence as to VICCard 5.0's ability to turn an English plaintext into a random message. Notice that most of the ciphertexts and

one-time pads have respectable p-values with the occasional outlier. The first and second Chi-Square tests on the ciphertext yielded average Chi-Square values of 0.4932 and 0.5945, respectively. In other words, on average there is about a 50/50 chance that the ciphertext is random. This is further reflected by the average p-value of 0.4614 in the data from Washington's Farewell Address. The first and second Chi-Square tests on the one-time pads yielded average Chi-Square values of 0.5488 and 0.3856, respectively.

Fortunately, many of the p-values are very high. Unfortunately, there are just as many p-values that are less than optimal. Still, this is not necessarily a bad thing. These low p-values are cause for concern if they are the result of a bias within VICCard 5.0. In other words, is there a security weakness of VICCard 5.0 that is making it give us low p-values?

To ensure that this is not the case, I performed what is called a drill-down test. I randomly picked the ninth group from Washington's Farewell Address, encrypted it differently, and found that the letter `D` occurred 31 times, a recognizable deviation from the expected $\frac{1044}{52} \approx 20.0769$ times. I then tracked each `D` through the encryption process to see if there was a feature of the cipher that caused a bias towards encrypting `D` more than any other letter. Given the extensive amount of substitutions that are performed (the checkerboard, the cipher block chaining, and the lagged Fibonacci generators), I was unable to find any factors that steered the cipher towards the letter `D`.

Instead, it appears that the high Chi-Square values are the result of an expected level of variation. For example, the second Gettysburg Address ciphertext has a Chi-Square value of 63.5060 and a p-value of 0.1123. Since the ciphertext had 1158 letters, each letter was expected to occur about $\frac{1158}{52} \approx 22.2692$ of the time. The high Chi-Square is not because every letter significantly deviates from occurring 22.2692 of the time. Instead, there were 7 outlying letters that occurred about 10 times more or less than the expected number. This is an allowable level of variation and is not necessarily a sign of a weakness.

To further confirm this, we will use the Empirical Rule. The Empirical Rule states that with normal data distributions and binomial data distributions, the data tends to follow a bell-shaped curve. Numerically, this means that about 68% of the data falls within one standard deviation of the mean, about 95% of the data falls within two standard deviations, and nearly all of the data falls within three standard deviations [2]. For example, consider the encryption of the declaration of independence. Since we expect each letter to occur $\frac{1}{52}$ of the time and since there are 6600 letters in the ciphertext, the standard deviation is $\sqrt{\left(\frac{1}{52}\right)\left(1-\frac{1}{52}\right)\left(\frac{1}{6600}\right)} \approx 0.0016905$. Furthermore, the expected number of occurrences of each letter is $\frac{6600}{52} \approx 126.92$. This means that we expect 68% of the data to be within $(0.0016905)(6600) \approx 11.16$ of 126.92, 95% of the data to be within $(2)(0.0016905)(6600) \approx 22.31$ of 126.92, and almost all of the data to be within $(3)(0.0016905)(6600) \approx 33.47$ of 126.92. Tables 18 through 21 confirm that the data for the six plaintexts follow this trend.

[2] The symbol for standard deviation is σ.

Plaintext	Percent in σ	Percent in 2σ	Percent in 3σ
Psalm 23	69.23%	96.15%	100.00%
Opening to *A Tale of Two Cities*	69.23%	96.15%	100.00%
"All I Ask Of You"	63.46%	94.23%	100.00%
Gettysburg	61.54%	96.15%	100.00%
"Paul Revere's Ride"	71.15%	100.00%	100.00%
Declaration of Independence	67.31%	96.15%	100.00%

Table 18: Empirical Rule for First Round of Chi-Square Test Results

Plaintext	Percent in σ	Percent in 2σ	Percent in 3σ
Psalm 23	55.77%	96.15%	98.08%
Opening to *A Tale of Two Cities*	69.23%	98.08%	100.00%
"All I Ask Of You"	76.92%	100.00%	100.00%
Gettysburg	61.54%	86.54%	100.00%
"Paul Revere's Ride"	69.23%	94.23%	100.00%
Declaration of Independence	73.08%	96.15%	98.08%

Table 19: Empirical Rule for Second Round of Chi-Square Test Results

Plaintext	Percent in σ	Percent in 2σ	Percent in 3σ
Psalm 23	69.23%	92.31%	100.00%
Opening to *A Tale of Two Cities*	65.38%	94.23%	98.08%
"All I Ask Of You"	69.23%	98.08%	100.00%
Gettysburg	75.00%	100.00%	100.00%
"Paul Revere's Ride"	80.77%	98.08%	100.00%
Declaration of Independence	69.23%	90.38%	98.08%

Table 20: Empirical Rule for First Round of Chi-Square Test Results (One-Time Pad)

Plaintext	Percent in σ	Percent in 2σ	Percent in 3σ
Psalm 23	71.15%	96.15%	98.08%
Opening to *A Tale of Two Cities*	61.54%	92.31%	98.08%
"All I Ask Of You"	73.08%	94.23%	100.00%
Gettysburg	63.46%	98.08%	100.00%
"Paul Revere's Ride"	75.00%	96.15%	100.00%
Declaration of Independence	63.46%	92.31%	100.00%

Table 21: Empirical Rule for Second Round of Chi-Square Test Results (One-Time Pad)

8 Closing Thoughts

Ernő Rubik, the inventor of the Rubik's cube, had a fond way of describing his creation. He affirmed that the Rubik's cube "embodies the tension of our most basic contradictions: simplicity and complexity... and so forth" [14]. The cube is simple because a brief glance is enough to figure out the goal of the puzzle. Only a few seconds are needed to discover how

the puzzle moves. However, determining the correct sequence of these moves is what makes it complex. It is this blend of simplicity and complexity that has driven the Rubik's cube's worldwide popularity [14].

This is the driving force behind playing card ciphers. A deck of cards is compact, portable, and readily accessible. However, its disarming simplicity is belied by the 225-bits of entropy that are packed into it. Furthermore, drawing out this wellspring of entropy is far from straightforward. The nascent field of playing card ciphers has brought to light many fascinating methods of doing so. In this research, I have presented my own contribution to this developing field.

VICCard 5.0 combines numerous cryptographic techniques. Certain features are reminiscent of VIC, which intensely addled the FBI during the Cold War. VICCard 5.0 also makes unique contributions of its own. It creates a novel substitution checkerboard, and it affords the incredible convenience of containing numerous keys in a single deck. In these regards, VICCard 5.0 distinguishes itself among other playing card ciphers. Furthermore, as demonstrated by the Chi-Square tests, it has the potential to create ciphertexts with respectable levels of randomness.

In a time when computer ciphers have become the industry standard, it is useful to not completely discount low-tech options. Analyzing the features that made hand ciphers secure for hundreds of years continues to inform and inspire our understanding of information security as a whole. In creating VICCard 5.0, my goal has been to show that computer ciphers have not entirely superseded hand ciphers. Additional innovation is still yielding formidable ciphers and fascinating cryptographic principles.

References

[1] Jim Dwyer. Sidelight to a Spy Saga: How a Brooklyn Newsboy's Nickel Would Turn Into a Fortune. https://www.nytimes.com/2015/11/04/nyregion/how-a-brooklyn-newsboys-nickel-helped-convict-a-soviet-spy.html, November 2015. Accessed July 8, 2020.

[2] FBI. Hollow Nickel/Rudolf Abel. https://www.fbi.gov/history/famous-cases/hollow-nickel-rudolph-abel. Accessed July 8, 2020.

[3] FermiLab. Physics Questions People Ask Fermilab. https://www.fnal.gov/pub/science/inquiring/questions/atoms.html, April 2014. Accessed July 20, 2020.

[4] Stephen Fry. Qi Card Shuffling - 52 Factorial. https://www.youtube.com/watch?v=SLIvwtIuC3Y, November 2012. Accessed July 23, 2020.

[5] Edy Victor Haryannto, Muhammad Zulfadly, Daifiria, Muhammad Barkah Akbar, and Ivy Lazuly. Implementation of Nihilist Cipher Algorithm in Securing Text Data With Md5 Verification. *Journal of Physics: Conference Series*, 1361, 2019.

[6] Jeffrey A. Hill. Chaocipher: Analysis and Models. http://www.chaocipher.com/HillDocs/H03H09.pdf, April 2009. Accessed June 8, 2020.

[7] David Kahn. Number One From Moscow. https://www.cia.gov/library/center-for-the-study-of-intelligence/kent-csi/vol5no4/html/v05i4a09p_0001.htm#top, July 2008. Accessed July 8, 2020.

[8] James Lyons. Cryptanalysis of the columnar transposition cipher. http://practicalcryptography.com/cryptanalysis/stochastic-searching/cryptanalysis-columnar-transposition-cipher/, 2012. Accessed July 23, 2020.

[9] Matthew McKague. Design and analysis of RC4-like stream ciphers. Master's thesis, University of Waterloo, Waterloo, ON, Canada, 2005.

[10] Sanjay Kumar Pal, Bimal Datta, and Amiya Karmakar. Cryptography and Network Security: A Historical Transformation. *SCHOLEDGE International Journal Of Multidisciplinary & Allied Studies*, 7(2):30–44, 2020.

[11] Isaac Reiter and Eric Landquist. Determining Biases in the Card-Chameleon Cryptosystem. *Communications on Number Theory and Combinatorial Theory.* Vol. 2, Article 1. Available at: https://research.library.kutztown.edu/contact/vol2/iss1/1.

[12] Moshe Rubin. The Chaocipher challenge: Further work in progress. http://www.mountainvistasoft.com/chaocipher/chaocipher-001.htm, November 2009. Accessed July 23, 2019.

[13] Moshe Rubin. John F. Byrne's Chaocipher Revealed: An Historical and Technical Appraisal. *Cryptologia*, 35:328–379, 2011.

[14] Ian Scheffler. *Cracking the Cube*. Touchstone, New York, New York, 2016.

[15] Aaron Toponce. The Chaocipher Cipher. https://aarontoponce.org/wiki/crypto/card-ciphers/chaocipher, October 2018. Accessed May 24, 2019.

[16] Aaron Toponce. Playing card ciphers. https://aarontoponce.org/wiki/crypto/card-ciphers, October 2018. Accessed May 28, 2020.

[17] Wikipedia contributors. VIC cipher — Wikipedia, the free encyclopedia, July 2020. Accessed June 3, 2020.

[18] Wikipedia contributors. Transposition cipher — Wikipedia, the free encyclopedia, October 2020. Accessed June 3, 2020.