How to Break Gifford's Cipher

Thomas R. Cain* and Alan T. Sherman†
Computer Science Department
University of Maryland Baltimore County
Baltimore, Maryland 21228-5398
cain@cs.umbc.edu, sherman@cs.umbc.edu

June 2, 1994

Abstract

We present and implement a ciphertext-only algorithm to break Gifford's cipher, a stream cipher designed in 1984 by David Gifford of MIT and used to encrypt New York Times and Associated Press wire reports. Applying linear algebra over finite fields, we exploit a time-space tradeoff to separately determine key segments derived from a decomposition of the feedback function. This work, the first proposed attack on Gifford's cipher, illustrates a powerful attack on stream ciphers and shows that Gifford's cipher is ill-suited for encrypting broadcast data in the MIT-based *Boston Community Information System (BCIS)*.

Gifford's cipher is a *filter generator*—a linear feedback shift register with nonlinear output. Our cryptanalytic problem is to determine the secret 64-bit initial fill, which is changed for each news article. Representing the feedback function as a binary matrix F, we decompose the vector space of register states into a direct sum of four F-invariant subspaces determined from the primary rational canonical form of F. The attack separately computes segments of the key corresponding to these invariant subspaces, which have dimensions 24, 5, 6, and 29, respectively. Because the dimension-24 subspace corresponds to a nilpotent transformation, Gifford's cipher effectively uses only 40 bits of key. With a novel hashing technique, we search these 40 bits in only 2^{27} steps. From the decomposition of F, we also compute the exact probability distribution of the leader and cycle lengths of all state sequences generated by Gifford's cipher.

Our attack runs in 2^{27} steps and 2^{18} bytes of memory, which is a significant shortcut over the 2^{64} steps required for a straightforward exhaustive search of all initial fills. Given ciphertext only from one encrypted article, our prototype implementation running on a loosely-coupled network of eight Sparcstations finds the article key within approximately four hours on average. Exploiting a key-management flaw of the BCIS, we also compute at no additional cost the corresponding master key, used for one month to encrypt all article keys in the same news section.

Keywords. Algorithms over finite fields, Boston Community Information System (BCIS), correlation attack, cryptanalysis, cryptography, cryptology, filter generators, Gifford's cipher, linear algebra over $GF(2)$, linear feedback shift registers (LFSRs), matrix decompositions, primary rational canonical form, similar matrices, similarity transformations, stream ciphers.

*Support for this research was provided in part by the University of Maryland Graduate School, Baltimore, through a 1991–92 Graduate Merit Fellowship.

†Part of this work was carried out while Sherman was a member of the Institute for Advanced Computer Studies, University of Maryland College Park.

1 Introduction

In 1982–84, David K. Gifford [14, 15, 16] and his research group at MIT designed and implemented a prototype system for transmitting up-to-the-minute New York Times and Associated Press wire reports to test subscribers in the Boston metropolitan area. Known as the *Boston Community Information System (BCIS)*, Gifford's system broadcast information streams on a subcarrier of MIT's FM radio station WMBR.[1] Each subscriber received and processed the streams using an IBM personal computer equipped with special-purpose receiver hardware. To protect against unauthorized access to the streams, and to be able to deny service to nonpaying customers, Gifford encrypted each stream. For this application, he devised and used a new stream cipher, which we shall call *Gifford's cipher*. The BCIS operated on an experimental basis from April 1984 through January 1988, providing a model for future community information systems. In this paper we analyze the security of Gifford's cipher, which had remained unbroken for almost a decade.

Gifford's cipher is a *filter generator*. This commonly-used type of cipher comprises a shift register, a linear feedback function, and a nonlinear output function. At each iteration, the feedback function is applied to the contents of the shift register to compute a feedback byte, which is shifted into the register. The output function is applied to four bytes of the shift register to produce a keystream byte. To encrypt a stream of plaintext bytes, each plaintext byte is exclusive-ORed (XORed) with a corresponding keystream byte to yield a ciphertext byte. The secret key is the 64-bit initial fill of the register. We present a new algorithm for computing the initial fill from ciphertext alone.

Several factors motivate us to study Gifford's cipher. First, since Gifford proposed his cipher for use in broadcast communications, it is important to know if this cipher might compromise valuable data. Second, we would like to further the understanding of filter generators so that system engineers can make prudent decisions regarding their implementation and appropriate use. Filter generators are interesting in part because they provide fast bulk encryption and because they can be easily implemented with limited resources. Third, Gifford's cipher provides a practical context in which to explore the general theme of exploiting algebraic decompositions in cryptanalysis.

Our goal is to evaluate the overall effectiveness of Gifford's cipher in protecting broadcast data in the BCIS, and more generally, to study the security of filter generators. To carry out this goal, we analyze the feedback and output functions; we develop cryptanalytic attacks against filter generators; and we implement a ciphertext-only attack against Gifford's cipher. In addition, we develop a new algorithm for computing a similarity transform between any binary matrix and its primary rational canonical form, as needed by our attack. This paper focuses on our detailed analysis of Gifford's cipher and on our implementation of one method for breaking it; for generalizations of our work and for analysis of other attacks on filter generators, see our companion paper [5].

Exploiting a decomposition of the feedback matrix F, we point out several ways to break Gifford's cipher. Our main result is the design and implementation of one of these methods, which computes the initial fill given ciphertext alone from one encrypted news article. This method applies a time-space tradeoff and runs in 2^{27} steps using 2^{18} bytes of memory; it does not require any statistical weaknesses of the output function. By contrast, our related statistical attack [5] on filter generators uses less space but assumes a slight statistical weakness in the output function; this alternate attack generalizes Siegenthaler's [41] correlation attack and runs in 2^{29} steps, or more

[1] BCIS information was represented as an FM signal, superimposed over the primary WMBR signal. Receiver hardware separated the signals.

generally 2^d steps, where d is the dimension of the largest subspace in any decomposition of the space of register states into a direct sum of F-invariant subspaces. Combining these two ideas achieves even faster attacks.

This paper explains in complete detail how to break a real cipher. The ingenuity and novelty of this work lies in its effective application of algorithmic and mathematical concepts—especially linear algebra over the finite field $GF(2)$—in a practical cryptanalytic context.

1.1 Previous Work

Although much is known about shift registers, we are aware of only four references to filter generators: Rueppel [39, pp. 83–93] outlines an application of Siegenthaler's [41] correlation attack to filter generators. Siegenthaler's attack is useful, but Rueppel's application of it appears not to be useful against Gifford's cipher. Rueppel [38, Ch. 5] also presents a framework in which to reason about filter generators using the algebraic normal form of the nonlinear output function. In their introductory survey on stream ciphers, Zeng, Yang, Wei, and Rao [44] briefly review how linear consistency and linear syndrome attacks can be applied to filter generators; and Dawson [8] states a few basic properties of filter generators.

For the classical theory of shift register cryptosystems, there are expositions by Beker and Piper [1, Ch. 5], Gill [17], Golomb [20], Rhee [36, Ch. 4], Ronse [37], and Rueppel [38]. A variety of attacks on stream ciphers have been published, including statistical attacks by Dawson and Clark [9], Golić and Mihaljević [18], Gollmann and Chambers [19], Klapper [26], Meier and Stafelbach [33], and Siegenthaler [41]. Many classical results are proven in Peterson and Weldon [35] and Berlekamp [2]. For a general survey of cryptanalytic techniques, see Brickell and Odlyzko [3].

For abstract linear algebra, we used standard texts by Hoffman and Kunze [22], Hungerford [23], and Jacob [24]. In addition, Watkins [43] surveys matrix theory, and Giesbrecht [13] presents some algorithms for matrix normal forms.

These references, however, do not adequately address the algorithmic aspects of efficiently applying linear algebra (including matrix decompositions) to cryptanalysis. Moreover, we found no previous work that describes in complete practical detail how to break any stream cipher.

1.2 Outline

The rest of this paper is organized as follows. Section 2 summarizes the main ideas of our attack. Section 3 introduces the Boston Community Information System, explains the role of encryption within this system, and reviews its key-management scheme. Section 4 describes Gifford's cipher in detail. Section 5 explains our decomposition of the feedback function. In particular, this section gives the primary rational canonical decomposition of the feedback function, a similarity transform between the feedback matrix and its decomposition, and our algorithms for carrying out these computations. Section 6 analyzes the cycle properties of the feedback function, giving an exact probability distribution on the leader and cycle lengths of all state sequences generated by Gifford's cipher.[2] Section 7 describes our ciphertext-only attack on Gifford's cipher, including experimental results of a prototype implementation of this attack. Section 8 analyzes two variations of Gifford's cipher. For example, in one variation, we show that an extremely small period would have resulted

[2]Any eventually periodic sequence consists of a leader and a cycle. The *leader* is the initial nonperiodic part; the *cycle* is the periodic part.

had a seemingly unimportant implementation detail of the feedback function been implemented differently. In addition, this section states several open problems. Section 9 summarizes our conclusions. Also, five appendices provide additional detailed information about our work.

2 Overview of Our Attack

Our attack on Gifford's cipher exploits an algebraic decomposition of the linear feedback function to reduce the effective number of secret key bits from 64 bits to 40 bits. To determine these unknown 40 bits from ciphertext alone, a variety of methods are possible; we implement a time-space tradeoff that runs in 2^{27} steps and uses 2^{18} bytes of memory. This section gives an overview of Gifford's cipher and our attack of it.

2.1 The Cipher

Gifford's cipher encrypts each news article under a separately chosen 64-bit article key s_0, known only to the sender and receiver. Each article is a sequence of 8-bit bytes $P_0, P_1, \ldots, P_{N-1}$; typically, $N \approx 10,000$. As shown in Figure 1, Gifford's cipher encrypts each article byte-by-byte, XORing each byte of plaintext with a corresponding keystream byte.

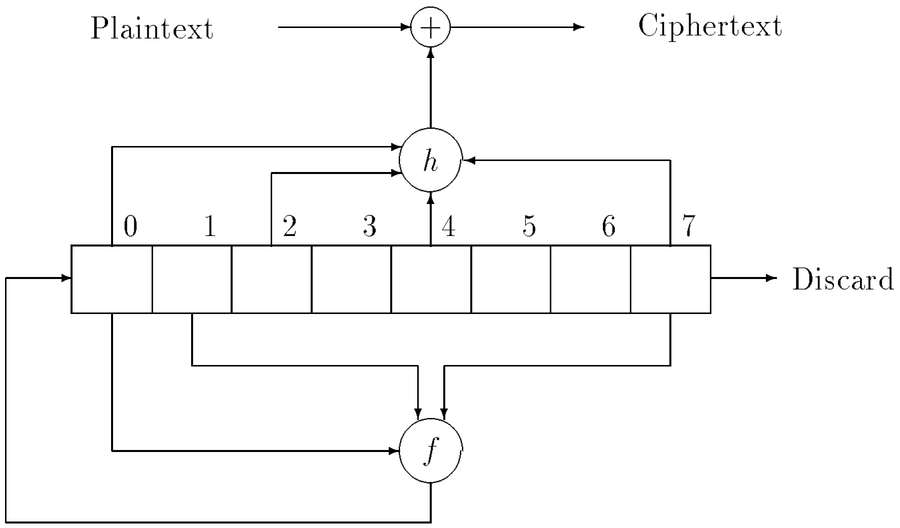

Figure 1: Gifford's stream cipher comprises an 8-byte shift register, a linear feedback function $f : \mathbb{Z}_2^{64} \rightarrow \mathbb{Z}_2^{64}$, and a nonlinear output function h. At each iteration, f computes a new register state as follows: A feedback byte is computed and shifted into the register from the left. Byte B_7 is discarded, and bytes B_0 through B_6 are shifted one byte to the right. The output function h computes an 8-bit keybyte from register bytes B_0, B_2, B_4, and B_7. The secret key is the initial fill of the register. Gifford's cipher generates a keybyte stream, which is XORed with the plaintext stream to produce a ciphertext stream. In this figure, + denotes XOR.

The keystream bytes are computed by applying a nonlinear output function h to the contents of a 64-bit shift register, which is implemented as a sequence of eight bytes. Specifically, for each $0 \leq t < N$, the tth byte of ciphertext is $C_t = P_t \oplus K_t$, where $s_t = f^t(s_0)$ is the tth state of the shift register; $K_t = h(s_t)$ is the tth keystream byte; and $\oplus$ denotes XOR. Here, $f : \mathbb{Z}_2^{64} \rightarrow \mathbb{Z}_2^{64}$ is the feedback function which is linear over the two-element Galois field $GF(2)$, and $h : \mathbb{Z}_2^{64} \rightarrow \mathbb{Z}_2^{8}$ is the output function which is nonlinear over $GF(2)$. The function h extracts eight bits from the product of two 16-bit integers derived from the state register. Gifford [15, p. 465] explained that "The security of the system depends on the period of the shift register, and on the ability of the nonlinear function to hide the contents of the register."

2.2 Our Attack

Given ciphertext from one article, our attack computes the article key s_0 by computing segments of the key corresponding to a decomposition of the feedback function.

In a one-time precomputation, we decomposed f as follows. First, from the detailed description of the cipher, we wrote down the 64×64 binary matrix F of the feedback function f. Second, with the help of math packages *Matlab* and *Macsyma*, we computed the primary rational canonical decomposition R of F, which is a 64×64 block-diagonal binary matrix. The matrix R has four blocks, and these blocks have dimensions 24, 5, 6, and 29, respectively. Third, using our own 59^3-step algorithm (which runs faster than the 64^6-step algorithm suggested by Gill [17]), we computed an invertible binary similarity matrix P satisfying $F = P^{-1}RP$. Our attack uses P^{-1} to move from the decomposed world to the original world, as needed to check candidate key segments.

As shown in Figure 2, the matrix R decomposes the 64-bit shift register into four subregisters $\mathcal{R}_0$, $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$, of lengths 24, 5, 6, and 29 bits, respectively. The similarity transformation P maps each register state into a corresponding sequence of four subregister states; thus, the key s_0 can be attacked in the four segments $Ps_0 = (d_0, d_1, d_2, d_3)$. Once any segment is known at any time, it is known for all future time: for each $0 \leq t < N$, it is true that $Ps_t = Pf^t(s_0) = (R_0^t d_0, R_1^t d_1, R_2^t d_2, R_3^t d_3)$, where R_0, R_1, R_2, R_3 are the four blocks of R. Furthermore, because R_0 is nilpotent, for all $t \geq 24$, $R_0^t d_0 = 0$. Thus, for all practical purposes, Gifford's cipher uses only $5 + 6 + 29 = 40$ bits of key.

Given ciphertext from one news article, we determine the key s_0 by searching over the initial fills of subregisters $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$. Our attack exploits two tricks to carry out this search in only 2^{27} steps. First, we recover the high-order bit of each keystream byte because Gifford represented each plaintext byte in extended 8-bit ASCII, with the leading bit 0. We check each candidate key against these bits. Second, instead of searching over all 2^{29} initial states of $\mathcal{R}_3$, we search over only 2^{29-N} states of $\mathcal{R}_3$ by checking every Nth state of $\mathcal{R}_3$: using a hashing technique, we check each candidate fill in expected constant time against all possible N positions of the ciphertext. With $N = 2^{13} = 8,192$, the total expected time for this search is $2^{5+6}2^{29-13} = 2^{27}$ steps.

Running on eight loosely-coupled Sparcstations, our implementation finds the secret key in approximately four hours on average. For example, when run on the ciphertext given in Appendix C, our implementation found the secret key within four hours and thirty minutes.

Finally, by computing the exponents of the irreducible factors of the characteristic polynomial for F, we prove that the period of state sequences generated from nonzero initial fills ranges from 21 to $349,502,963,061 \approx 3.5 \times 10^{11}$ states.

Figure 2: A decomposition of Gifford's cipher. Our attack exploits the primary rational canonical decomposition R of the feedback matrix F. This decomposition of F induces a decomposition of the space of register states into a direct sum of four invariant subspaces of dimensions 24, 5, 6, and 29, respectively. A similarity transformation P satisfying $F = P^{-1}RP$ maps each register fill s from the original world into four subfills $Ps = (d_0, d_1, d_2, d_3)$ in the decomposed world, corresponding to the invariant subspaces. In the decomposed world, the feedback transformation f_R is a direct sum of invariant transformations $f_R = f_0 \oplus f_1 \oplus f_2 \oplus f_3$. Our attack searches each subfill to determine the secret key. One version checks each candidate subfill using a time-space tradeoff; another checks each subfill separately using statistical correlations with the ciphertext stream. Each attack runs in at most 2^{29} steps, which is a significant shortcut over the 2^{64} steps required for a straightforward exhaustive search of all 2^{64} initial fills.

3 Data Security in the Boston Community Information System

In the *Boston Community Information System (BCIS)*, broadcast data were protected for two reasons: to provide the information service only to paying customers, and to restrict dissemination of data as required by its owners (*e.g.* some material was copyrighted). As described by Gifford [15], the BCIS protected its broadcast data through stream encryption and a key-management scheme.

3.1 Requirements and Design Decisions

Data security in the BCIS was designed with four requirements in mind: cryptographic strength, flexibility, limited receiver hardware, and computational efficiency. Gifford [15, p. 466] explained, "We needed a bulk encryption algorithm that was reasonably secure yet that was efficient enough to be implemented in software on contemporary personal computers." As for the level of security required, Gifford needed a system that would provide adequate protection for news articles—though Gifford did not identify any specific threats. To allow each customer the flexibility of subscribing to any subset of information streams, each stream was encrypted separately. Gifford [15, p. 465] summarily explained that, due to limited receiver hardware, he could not "utilize better-known encryption techniques such as DES or RSA" and therefore implemented his own algorithm in software. With each subscriber's IBM PC performing encryption and all other data processing, Gifford took as a design requirement for encryption not to consume more than approximately 12% of the available CPU.[3] In addition, he needed encryption rates in the 2 kbits/sec range.

3.2 Key Management

The BCIS used two levels of cryptographic keys—article keys and master keys. Each information stream was controlled by a separate 64-bit *master key*. Periodically, each subscriber received only those master keys controlling the streams she had selected and paid for. A table of master keys was supposed to have been sent to each subscriber monthly via U.S. mail; in practice, however, master keys were rarely changed, if ever.

Each news article was encrypted with Gifford's cipher using a randomly chosen 64-bit *article key*, separately chosen for each article. This cipher is described in Section 4. Each article key was encrypted under the appropriate master key and broadcast (in its encrypted form) along with the encrypted article.

A problem with this scheme is that there is no way to prevent customers from disclosing their monthly master keys or to limit loss caused by such disclosures. As Sherman had suggested to Gifford circa 1983, a more secure arrangement would be to equip each receiver hardware with a tamper-proof cryptographic unit, which would perform all decryption and key-management functions, and which would hold a private key unique to the unit. Using public-key cryptology, frequently-updated master keys could be transmitted separately to each unit (say, every thirty minutes), encrypted under the unit's public key. Without physical security, however, it is impossible to prevent a subscriber from disclosing her master keys. Gifford chose not to follow Sherman's suggestion on the grounds that it would be too difficult to implement.

[3] David K. Gifford, private correspondence (circa 1984).

3.3 Article Key Encryption

Each article key was encrypted by XOR with the master key. Therefore, compromise of any article key also compromised the corresponding master key, which remained valid for one month. Certainly Gifford knew better: he simply used the XOR method as a temporary method with the intention of eventually using DES or some other cipher. Yet, even after the reimplementation in late 1984 and early 1985, the XOR method remained a glaring weakness of the BCIS.

4 Gifford's Cipher

Gifford's cipher is a stream cipher with 64-bit key, used in the BCIS to encrypt news articles. Our first two steps in analyzing the cipher were to determine its exact operation and to choose a suitable mathematical model in which to reason about its properties. Since Gifford's [15, pp. 464–465] published description of his cipher is incomplete, we started with source code from the BCIS. Thus, we started with a low-level operational view of the cipher, which view sheds little insight on the cipher's properties. In this section we define and mathematically model Gifford's cipher, separately considering its source code, feedback function, output function, and use in article encryption.

4.1 BCIS Source Code

In the BCIS, Gifford's cipher was implemented by one page of source code written in the C programming language. Circa 1985, with Gifford's permission, Sherman obtained this source code from then graduate student Stephen Berlin, who worked on the BCIS. A facsimile of this code appears in Appendix A. The code was helpful to us in defining the feedback and output functions, which are incompletely described in Gifford's [15] journal article.

A peculiar feature of the source code is the last conditional statement, which is accompanied by the mysterious comment: "This [statement] is here purely to conform to the other C compiler and sign extension of right shifted variables." This conditional statement forces a certain right byte-shift operation in the computation of the output function to propagate the leftmost (sign) bit. As explained in Section 4.4, we call this type of shifting "sticky" shifting—as opposed to "zero-fill" shifting.[4] Investigation determined the following: As a feature of one BCIS compiler, the BCIS implementation of Gifford's cipher uses zero-fill left-shift but sticky right-shift. Because not all of the BCIS compilers use sticky right-shift, the conditional statement was added to force uniformity among all BCIS compilers. Earlier source code, however, did not include this statement.

Thus, initially, Gifford left to the compiler the decision whether to use sticky right-shift versus zero-fill right-shift, even though Gifford's [15, p. 465] description of the "bank view" of his cipher (see Section 4.6) suggests that Gifford may have at one time intended to use zero-fill right-shift. Remarkably, as we prove in Section 8.1, the compiler's action on this seemingly unimportant decision drastically affects the period of the feedback function. Fortuitously, Gifford's arbitrary use of sticky right-shift averted pathologically short periods of at most 9,198 states.

[4] Stick-shifting is sometimes called "arithmetic right-shifting" because it corresponds to dividing a two's complement integer by two.

4.2 Article Encryption and the Stream Cipher $\mathcal{G}$

Each news article is a *stream of plaintext bytes* $\{P_t\}_{t=0}^{N-1}$ of some length N. For each key, Gifford's cipher generates a *keybyte stream* $\{K_t\}_{t=0}^{N-1}$, which is XORed with the plaintext stream to yield a *ciphertext stream* $\{C_t\}_{t=0}^{N-1} = \{P_t \oplus K_t\}_{t=0}^{N-1}$. Throughout, every byte is a sequence of eight bits. The secret key is the initial fill of a 64-bit shift register.

We shall use the following notation throughout. Let $\mathbb{Z}_2 = \{0,1\}$, and let $\mathcal{P} = \mathcal{K} = \mathcal{C} = \mathbb{Z}_2^8$ denote, respectively, the sets of all possible plaintext bytes, keybytes, and ciphertext bytes. Similarly, let $\mathcal{S} = \mathbb{Z}_2^{64}$ denote the set of all possible *states* of the shift register. Although each article is approximately 10,000 bytes long, it is convenient to view plaintext and keybyte streams as infinite. To this end, let $\mathcal{P}^*$, $\mathcal{K}^*$, $\mathcal{C}^*$, and $\mathcal{S}^*$ be, respectively, the set of all possible *infinite sequences (streams)* of plaintext bytes, keybytes, ciphertext bytes, and shift register states.

To reason about Gifford's cipher, it is helpful to distinguish between the stream cipher $\mathcal{G}$ and the underlying filter generator G^*. *Gifford's filter generator* is a mapping $G^* : \mathcal{S} \to \mathcal{K}^*$, that for each initial fill $s_0 \in \mathcal{S}$, generates a sequence of keybytes $G^*(s_0) = \{K_t\}_{t=0}^{\infty}$. *Gifford's stream cipher* is the mapping $\mathcal{G} : \mathcal{S} \times \mathcal{P}^* \to \mathcal{C}^*$ defined by

$$\mathcal{G}(s_0, \rho) = \rho \oplus G^*(s_0) = \{P_t \oplus K_t\}_{t=0}^{\infty}, \tag{1}$$

whenever $s_0 \in \mathcal{S}$ is any initial fill and $\rho = \{P_t\}_{t=0}^{\infty} \in \mathcal{P}^*$ is any plaintext stream.

4.3 The Filter Generator G^*

Given any initial fill $s_0 \in \mathcal{S}$, Gifford's filter generator G^* computes a stream of keybytes $\{K_t\}_{t=0}^{\infty}$ by applying a nonlinear output function $h : \mathcal{S} \to \mathcal{K}$ to the contents of the shift register. At each iteration, a new state of the shift register is computed by a linear feedback function $f : \mathcal{S} \to \mathcal{S}$. Specifically, for any initial fill $s_0 \in \mathcal{S}$ and any $t \in \mathbb{N}$, the keystream byte K_t generated at time t is

$$K_t = G(s_0, t) = h(s_t) = (h \circ f^t)(s_0), \tag{2}$$

where $s_t = f^t(s_0)$ is the state of the register at time t. Formally, G is a function $G : \mathcal{S} \times \mathbb{N} \to \mathcal{K}$.

It is convenient to define the related mapping $G^*_{\mathcal{S}} : \mathcal{S} \to \mathcal{S}^*$, that for each initial fill s_0, generates the eventually periodic *sequence of register states* $G^*_{\mathcal{S}}(s_0) = \{s_t\}_{t=0}^{\infty}$. *Gifford's filter generator* is the mapping $G^* = h \circ G^*_{\mathcal{S}}$. Thus, each initial fill corresponds to a state sequence. An important feature of filter generators is that each relatively short key can determine an exponentially longer state sequence. In Section 6, we exactly determine the probability distribution of the leader and cycle lengths of these state sequences.

As shown in Figure 1, the shift register consists of eight 8-bit bytes $B_0, B_1, \ldots, B_7$. At each iteration, a new byte is computed and shifted into the register from the left (becoming the new B_0), and the rightmost byte, B_7, is discarded. Each of the old bytes B_0 through B_6 is shifted one byte to the right. The feedback function f depends only on bytes B_0, B_1, and B_7. As determined from the source code, the output function h depends only on bytes B_0, B_2, B_4, and B_7. Gifford [15, p. 465] explained that "for implementation efficiency, [the] register [is] shifted one byte at a time, rather than one bit at a time." Note, however, that the new byte is not simply $B_0 \oplus B_1 \oplus B_7$; as explained in Section 4.4, because f shifts bytes B_0 and B_7 bit-wise before computing an XOR, the feedback function is linear over $GF(2)$ but not over $GF(2^8)$.

Regarding his choice for f to depend solely on bytes B_0, B_1, and B_7, Gifford [15, p. 465] explained that "the tap positions were chosen to yield the longest period that could be obtained" if the new byte were computed as $B_0 \oplus B_1 \oplus B_7$. As we prove in Section 8.2, however, Gifford's choice of taps does not achieve this objective. Although Gifford does not explain his choice of input bytes to h, his choice was probably motivated in part by the following observation: Initially, bytes B_0 and B_2 are inputs to h. After five iterations, they become bytes B_5 and B_7. By having h depend on B_4 rather than on B_5, Gifford avoids the weakness of repeating exact inputs to h. Avoiding this repetition is especially important in light of the algebraic property—explained in Section 4.5—that Gifford's output function is invariant when bytes B_0 and B_2 are exchanged with bytes B_4 and B_7, respectively.

4.4 The Linear Feedback Function f

Gifford describes the feedback function $f : \mathcal{S} \to \mathcal{S}$ operationally in terms of its action on the eight bytes of the shift register. As shown in Figure 3, for any register state $s_t = (B_0, B_1, \ldots, B_7)$, the function f computes the next state of the register as

$$f(s_t) = (f_{new}(B_0, B_1, B_7), B_0, B_1, \ldots, B_6), \tag{3}$$

where the new feedback byte is computed by the function $f_{new} : \mathbb{Z}_2^{24} \to \mathbb{Z}_2^8$.

The feedback byte is the XOR of bytes B_0, B_1, and B_7, with byte B_1 sticky-shifted one bit to the right, and with byte B_7 zero-fill-shifted one bit to the left. Thus,

$$f_{new}(B_0, B_1, B_7) = B_0 \oplus (\gg_1^\star (B_1)) \oplus (\ll_1 (B_7)), \tag{4}$$

where $\gg_1^\star$ and $\ll_1$ denote, respectively, the sticky right-shift and zero-fill left-shift operations. Specifically, for any byte $B = (x_0, x_1, \ldots, x_7)$,

$$\gg_1^\star (B) = (x_0, x_0, x_1, x_2, \ldots, x_6) \qquad \text{and} \qquad \ll_1 (B) = (x_1, x_2, \ldots, x_6, x_7, 0). \tag{5}$$

The bit-shifting of bytes B_1 and B_7 complicates the feedback function in two respects: First, this bit-shifting causes the function f_{new} to be nonlinear over bytes—*i.e.* over $GF(2^8)$. It is easy to verify computationally that f_{new} and hence f are nonlinear over $GF(2^8)$. For example, representing each byte as two hexadecimal numerals, let $w = (00_{16}, 41_{16}, 00_{16})$. Then, as a result of bit-shifting byte B_1 to the right, $f_{new}(w) = 20_{16}$ but $f_{new}(2 * w) = f_{new}(00_{16}, 82_{16}, 00_{16}) = C1_{16}$. Thus, $f_{new}(2 * w) \neq 2 * f_{new}(w)$ modulo 2^8, even if the right-shift were implemented with zero-fill. Second, the bit-shifting causes f to be (slightly) noninvertible. Specifically, f loses one bit—the high-order (leftmost) bit of byte B_7.

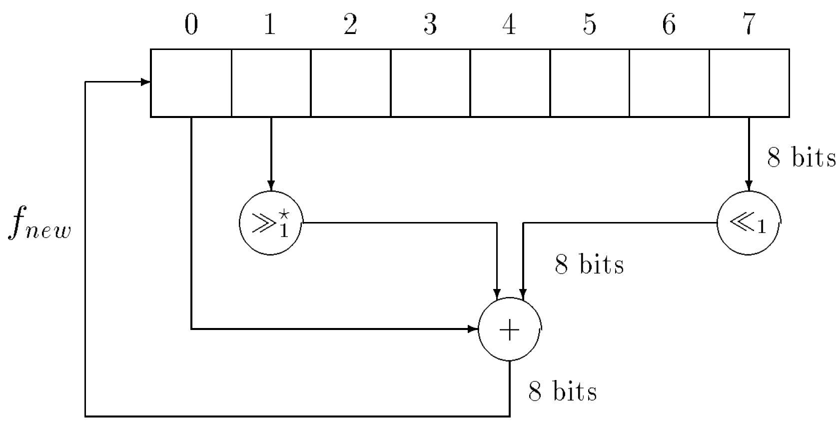

Figure 3: The linear feedback function $f_{new} : \mathbb{Z}_2^{24} \to \mathbb{Z}_2^8$ computes a feedback byte as the XOR of register bytes B_0, B_1, and B_7, with byte B_1 shifted one bit to the right, and with byte B_7 shifted one bit to the left. Here, $\gg_1^\star$ denotes sticky right-shift, and $\ll_1$ denotes zero-fill left-shift. As a result of the bit-shifting of bytes B_1 and B_7, the function f_{new} is linear over $GF(2)$ but not over $GF(2^8)$.

4.5 The Nonlinear Output Function h

As shown in Figure 4, the nonlinear output function $h : \mathcal{S} \rightarrow \mathcal{K}$ extracts an 8-bit byte from the product of two 16-bit integers derived from shift register bytes B_0, B_2, B_4, and B_7. Although h is formally defined on $\mathcal{S}$, it is convenient to express h in terms of its restriction $\hat{h} : \mathbb{Z}_{256}^4 \rightarrow \mathbb{Z}_2^8$. For any register state $s = (B_0, B_1, \ldots, B_7)$, define

$$h(s) = \hat{h}(B_0, B_2, B_4, B_7) = \mathrm{Extract\_Byte}\,((B_0 \| B_2) * (B_4 \| B_7)), \tag{6}$$

where $\|$ denotes concatenation; $*$ denotes integer multiplication; and the function Extract_Byte : $\mathbb{Z}_2^{32} \rightarrow \mathbb{Z}_2^8$ extracts the third byte from the left of any 32-bit number. Thus, for any $X \in \mathbb{Z}_2^{32}$, $\mathrm{Extract\_Byte}\,(X) = (\gg_8 (X)) \bmod 256$, where $\gg_8$ denotes right shift by eight bits. For example, representing each byte by two hexadecimal numerals, if $B_0 = 10_{16}$, $B_2 = 02_{16}$, $B_4 = 11_{16}$, and $B_7 = 03_{16}$, then $\hat{h}(B_0, B_2, B_4, B_7) = \mathrm{Extract\_Byte}\,(1002_{16} * 1103_{16}) = \mathrm{Extract\_Byte}\,(0110\underline{52}06_{16}) = 52_{16}$. Using similar arguments to that given for the nonlinearity of f over $GF(2^8)$, it is easy to verify that $\hat{h}$ and thus h are nonlinear, both over $GF(2)$ and over $GF(2^8)$.

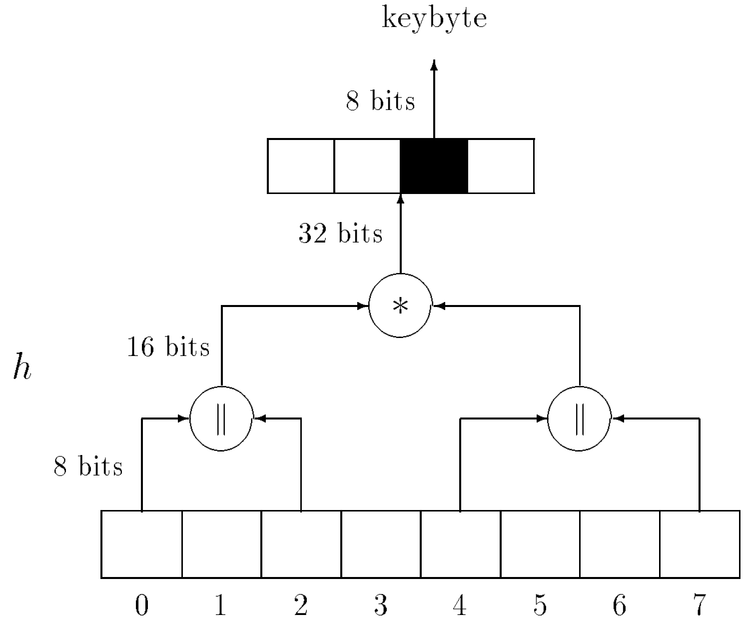

Figure 4: The nonlinear output function h extracts an 8-bit byte from the product of two 16-bit integers derived from register bytes B_0, B_2, B_4, and B_7. One of these 16-bit integers is the concatenation of bytes B_0 and B_2; the other is the concatenation of bytes B_4 and B_7. From the product of these integers, h extracts the third byte from the left. In this figure, $\|$ denotes concatenation of two 8-bit bytes, and $*$ denotes integer multiplication.

Suspecting possible weaknesses of his h function, Gifford [15, p. 465] admonished without elaboration: "This method may be too structured to resist cryptanalytic attack." Indeed, as we explain in Appendix E, there are many properties of h to be exploited. For example, by the commutative property of integer multiplication, for all $A, B, C, D \in \mathbb{Z}_2^8$, it is true that $\hat{h}(A, B, C, D) = \hat{h}(C, D, A, B)$. Also, if $A = B = 0$ or $C = D = 0$, then $\hat{h}(A, B, C, D) = 0$. Without explanation, Gifford [15, p. 465] also conjectured, "A system based on noninvertible table lookups may be more secure."

Gifford's inspiration for h came from a well-known idea of John von Neumann. According to Knuth [27, pp. 3–4], in 1946, von Neumann suggested generating a pseudorandom sequence of numbers $r_0, r_1, \ldots$ by the recurrence $r_i = \text{middle}(r_{i-1})^2$, for $i > 0$, where $\text{middle}(r_{i-1})$ denotes the middle digits of r_{i-1}. Knuth [27, p. 3] notes that "von Neumann's ... method ... proved to be a comparatively poor source of random numbers ... [because] the sequence tends to get into a ... short cycle." It does not necessarily follow, however, that Gifford's interpretation of this method suffers from the same problem.

4.6 Discussion—Modeling f and h

To reason about Gifford's cipher, and to apply mathematical tools to our cryptanalytic problem, it is necessary to adopt a suitable mathematical model of the component functions f and h. In this section we describe how we model these functions and we point out some alternative models.

Modeling f

We view the feedback function f as a linear transformation over $\mathbb{Z}_2$. Thus, we view $\mathbb{Z}_2$ as the two-element Galois field $GF(2)$, with operations addition and multiplication modulo 2. In this perspective, we view the set of register states $\mathcal{S} = \mathbb{Z}_2^{64}$ as a vector space over $\mathbb{Z}_2$: the vectors are bit sequences of length 64, and vector addition is bitwise XOR. The endomorphism f is linear over $\mathbb{Z}_2$ because it is the composition of linear operations (*i.e.* bit projections and XOR).

Although modeling f as a linear transformation over $\mathbb{Z}_2$ provides a simple and well-understood framework, other models are also possible. For example, given the operational definition of f acting on bytes, it is natural to consider the blocking $\mathbb{Z}_2^{64} \cong (\mathbb{Z}_2^8)^8$. Moreover, it is tempting to view $\mathbb{Z}_{256}$ as the Galois field $GF(2^8)$ and to view $\mathcal{S} \cong \mathbb{Z}_{256}^8$ as a vector space over $\mathbb{Z}_{256}$. We call this view the "byte view." Indeed, the byte view may lead to new insights and to speedups based on using a larger base field. Although this view reduces the number of variables needed to define the filter generator, this view requires a more complicated understanding of f because f is nonlinear over $\mathbb{Z}_{256}$.

Gifford's feedback function differs from traditional feedback functions in that it feeds back a byte rather than a bit. Therefore, the standard theory of shift registers cannot be directly applied over $\mathbb{Z}_2$. To deal with this situation, we view f as a function from $\mathcal{S}$ to $\mathcal{S}$.

Another view is what we call the "bank view." Without elaboration, Gifford [15, p. 465] remarked that "the ... shift register ... can be pictured as a bank of eight 8-bit shift registers ..., where the new bit in each of the eight registers is an XOR combination of 1 bit from the previous register, 1 bit from the register itself, 1 bit from the next register (with identical tap arrangements for each register)." The forward and backward chaining of the registers through the XOR operation results from bit-shifting bytes B_1 and B_7 within f. For the first register, due to the sticky right-shifting of byte B_1, the bit input for the previous register would be the second bit of the first register; it would not be the constant 0 as Gifford said. Similarly, due to the zero-fill left-shifting of byte B_7, for the last register, the bit input for the next register would be the constant zero. Thus, for $0 \leq j \leq 7$, the jth register would consist of the jth bit from each of the eight bytes of the original register.

To a cryptanalyst, one attraction of the bank view is that, were it not for the forward and backward chaining of the registers, each register would have a maximum period of 127 states, as proved in Section 8.2. Also, with this view it might be possible to attack each bank separately. If the effects of forward and backward chaining could be handled—for example, by modeling the effect as noise—this view might be productive. The chaining, however, seems to have a significant influence.

Although other views might also be productive, for simplicity, we chose to model f as a linear transformation over $\mathbb{Z}_2$. Having adopted this linear model, our next step was to write down the matrix representation F of f. Appendix B lists this matrix, which we explain in Section 5.1.

Modeling h

In the current implementation of our attack, we view h simply as a "black-box" function that maps register states to keybytes; that is, we assume we know how to compute h, but we assume nothing else about h. In our related statistical attack, we additionally assume a mild statistical weakness of h.

In our preliminary analysis of h, we also modeled this function as one or more Diophantine equations. For example, by interpreting Equation 6 algebraically over $GF(2^8)$, it follows that

$$\hat{h}(A, B, C, D) = (BC + AD + \lfloor BD/256 \rfloor) \bmod 256, \tag{7}$$

whenever $A, B, C, D \in \mathbb{Z}_{256}$, where $+$ denotes integer addition and where the implicit product operation is integer multiplication. From this model we proved, that by solving certain linear Diophantine equations derived from Equation 7, it is possible to invert $\hat{h}$ given its output and three of its inputs (for exact statements of this property, see Appendix E). This inversion property can be exploited to optimize some attacks.

Similarly, working over $GF(2)$, it is possible to model h as a system of eight Boolean equations that define the eight bits of output. Each of these equations can be defined in terms of up to 32 unknown bits of the shift register. More interestingly, exploiting our decomposition of f, each of these equations can be defined in terms of up to 40 unknown bits of subregisters $\mathcal{R}_1$ through $\mathcal{R}_3$. Although the Boolean equations defining h are rather messy due to the carries in the integer multiplication, they are not intractable. We conjecture it would be promising to study them further. To deal with the carries, it may be useful to apply the mathematical machinery of Lomonaco [29], who developed an algorithm for integer factoring based on solving Boolean equations.

5 Decomposition of the Feedback Function

Our attack exploits a decomposition of the feedback function f to search segments of the key separately. To begin, we view the state space $\mathcal{S} = \mathbb{Z}_2^{64}$ as a vector space over $GF(2)$, and we view f as a linear transformation of $\mathcal{S}$. We represent the feedback function f as a binary matrix F and work with its primary rational canonical form R. The binary matrix R is a block diagonal matrix, similar to F. This decomposition of F induces a decomposition of the state space into a direct sum of F-invariant subspaces.

In this section, we compute the canonical form R and an invertible binary similarity matrix P such that $F = P^{-1}RP$. To compute these matrices, we first derive the feedback matrix F, the minimal and characteristic polynomials for F, and the factors of these polynomials. In Section 6, from the exponents of these factors, we determine the exact probability distribution of the leader and cycle lengths of all state sequences generated by Gifford's cipher.

5.1 The Feedback Matrix F

To apply the tools of linear algebra to our cryptanalytic tasks, it is useful to represent the feedback function f as a matrix. Because f is linear over $GF(2)$, there exists a representation of f as a 64×64 binary matrix. Appendix B presents this matrix F, which we derived from Equations 3 and 4.

To derive F, we write down Boolean equations that express bits of the shift register at time $t+1$ with bits of the register at time t. For any $t \in \mathbb{N}$, let $s_t = (b_0^t, b_1^t, \ldots, b_{63}^t)$ and $s_{t+1} = (b_0^{t+1}, b_1^{t+1}, \ldots, b_{63}^{t+1})$ be states of the register at times t and $t+1$, respectively. From Equation 4 and from the byte-shifting, the following 64 identities hold:

$$
\begin{aligned}
b_0^{t+1} &= b_0^t + b_8^t + b_{57}^t, & (8)\\
b_j^{t+1} &= b_j^t + b_{j+7}^t + b_{j+58}^t \text{ for } j = 1, \ldots, 6, & (9)\\
b_7^{t+1} &= b_7^t + b_{14}^t + 0 = b_7^t + b_{14}^t, \text{ and} & (10)\\
b_j^{t+1} &= b_{j-8}^t \text{ for } j = 8, \ldots, 63, & (11)
\end{aligned}
$$

where $+$ denotes addition modulo 2.

Equations 8–10 compute the new feedback byte. Observe that the sticky right-shift of byte B_1 accounts for the b_8^t term in Equation 8. Similarly, the zero-fill left-shift of byte B_7 explains the constant 0 in Equation 10. Finally, Equation 11 describes the bytewise shifting of bytes B_0 through B_7. Each of these 64 identities defines a row in the matrix F.

As discussed in Section 4.4, the left bit-shift of byte B_7 loses one bit of information—b_{56}^t. Consequently, b_{56}^t does not appear on the right side of Equations 8–11, and column 56 of matrix F is a column of zeros (we index columns from 0 to 63). Therefore, F is not invertible. We computationally verified that the rank of F is exactly 63. In particular, we computed the row-echelon form of F and observed that it has exactly one row of zeros. In addition, we computed a basis for the kernel of F and thereby verified that $\dim(\ker(F)) = 1$.

In Section 5.4, we show that the noninvertibility of F is restricted to the leader of the state stream, which has at most 24 states. Specifically, in R, the only noninvertible block is the dimension 24 block that computes the leader. This fact has the following cryptanalytic significance: to determine the first state in the cyclic portion of the state stream uniquely, it suffices to find any state of the register in the cyclic portion of the state stream.

As shown in Appendix B, the matrix F has the block structure

$$
F = \begin{pmatrix} F_0 & F_1 & 0 & F_7 \\ & I_{56,56} & & 0 \end{pmatrix}, \tag{12}
$$

which arises from the byte operations within f. Corresponding to tapped bytes B_0, B_1, and B_7, the three 8×8 blocks F_0, F_1, F_7 calculate the feedback byte. The 56×56 identity matrix $I_{56,56}$ in the lower-left corner of F describes the shifting of bytes B_0 through B_6. Because byte B_7 is discarded, there is a 56×8 matrix of zeros in the last eight columns of F.

We now explain blocks F_0, F_1, and F_7. Block F_0 is an 8×8 identity matrix, expressing the fact that byte B_0 is not bit-shifted within f. By contrast, blocks F_1 and F_7 are, respectively, the upper- and lower-diagonal matrices

$$F_1 = \begin{pmatrix} 1&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \end{pmatrix} \quad \text{and} \quad F_7 = \begin{pmatrix} 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0 \end{pmatrix}, \tag{13}$$

expressing the right bit-shifting of byte B_1 and the left bit-shifting of byte B_7. Because byte B_1 is sticky-shifted, the upper-left bit of block F_1 is one.

5.2 The Characteristic and Minimal Polynomials of F and Their Factors

To compute R, and to characterize the periodic properties of f, it is helpful to know the characteristic and minimal polynomials of F and their irreducible factors. Intuitively, the characteristic polynomial for F encodes all of the information of F in a convenient algebraic form. In this section, we present these polynomials and explain how we computed them with the help of the math packages *Matlab* and *Macsyma* [31].

First, we review some basic concepts from linear algebra.[5] Let $\mathbb{Z}_2[x]$ denote the ring of polynomials over $\mathbb{Z}_2$, and let $f(x) \in \mathbb{Z}_2[x]$. The polynomial $f(x)$ is *irreducible over* $\mathbb{Z}_2$ if $f(x)$ cannot be written as the product of two polynomials in $\mathbb{Z}_2[x]$ of lower degree. The *characteristic polynomial* of F is the degree 64 polynomial $p_F(x) = \det(F - xI)$, where I is the 64×64 identity matrix. The *minimal polynomial* of F, denoted by $m_F(x)$, is the polynomial of smallest degree over $\mathbb{Z}_2$ such that $m_F(F) = 0$. By the Cayley-Hamilton theorem, $p_F(F) = 0$. Because the minimal polynomial is the unique generator of the principal ideal of polynomials over $\mathbb{Z}_2$ that annihilate F (see [22, p. 191]), it follows that $m_F(x)$ divides $p_F(x)$. Moreover, the minimal and characteristic polynomials contain the same irreducible factors (see [22, p. 193]); it is possible, however, that these irreducible factors may occur with higher multiplicities in $p_F(x)$.

For Gifford's cipher, it turns out that $m_F(x) = p_F(x)$, which fact simplifies some of the theory. In particular, as explained in Section 5.3, $m_F(x) = p_F(x)$ implies that the invariant subspaces in our decomposition are F-cyclic.

We computed the characteristic polynomial $p_F(x)$ of F by computing the determinant $\det(F - xI)$ using the numerical math package *Matlab*. Since *Matlab* is not equipped to perform arithmetic

[5] For a review of linear algebra, see Hoffman and Kunze [22], Hungerford [23], and Jacob [24].

over $\mathbb{Z}_2$, we performed the calculation over $\mathbb{R}$ and accepted the modulo 2 values of the coefficients from the resulting polynomial.[6] This computation produced the following polynomial for F:

$$\tilde{p}_F(x) = x^{64}+ x^{62}+ x^{61}+ x^{60}+ x^{59}+ x^{58}+ x^{57}+ x^{55}+ x^{54}+ x^{52}+ x^{50}+ x^{48}+ x^{44}+ x^{40}+ x^{24}. \quad (14)$$

Using *Macsyma*, we factored $\tilde{p}_F(x)$ into a product of irreducible polynomials $\tilde{p}_F(x) = p_0^{24}(x)\, p_1(x)\, p_2(x)\, p_3(x)$, where

$$\begin{aligned}
p_0(x) &= x, && (15)\\
p_1(x) &= x^5 + x^3 + x^2 + x + 1, && (16)\\
p_2(x) &= x^6 + x^5 + x^4 + x^2 + 1, \text{ and} && (17)\\
p_3(x) &= x^{29}+x^{28}+x^{26}+x^{22}+x^{20}+x^{19}+x^{18}+x^{16}+x^{14}+x^{13}+x^{10}+x^9+x^7+x^5+x^4+x^3+x^2+x+1. && (18)
\end{aligned}$$

Since runtime-errors (*e.g.* roundoff errors) might have affected Matlab's computation of $\tilde{p}_F(x)$, we verified these calculations in four steps: First, we verified that each of the polynomials $p_0(x)$ through $p_3(x)$ is irreducible over $\mathbb{Z}_2$ and that their product is $\tilde{p}_F(x)$. Marsh [32] lists polynomials $p_1(x)$ and $p_2(x)$ in his table of irreducible polynomials, and in Section 6, we prove the irreducibility of $p_3(x)$ within our proof of Proposition 2.

Second, using our own C language programs, we computationally verified that $\tilde{p}_F(F) = 0$; hence, $\tilde{p}_F(x)$ is a multiple of $m_F(x)$. Third, we computationally verified that x, $p_1(x)$, $p_2(x)$, and $p_3(x)$ are factors of the minimal polynomial.

Fourth, using our own programs, we computationally verified that the minimum integer $1 \leq t \leq 24$ such that $p_0^t(F)\, p_1(F)\, p_2(F)\, p_3(F) = 0$ is $t = 24$. Since the minimal and characteristic polynomials contain the same irreducible factors, and since $p_0(x)$ is the only irreducible factor of $\tilde{p}_F(x)$ that occurs with multiplicity greater than one, it follows that $\tilde{p}_F(x) = p_F(x) = m_F(x)$ is the unique characteristic polynomial for F.

Our decomposition of F depends on the *elementary divisors* of F, which by definition are the prime-power polynomial factors of $m_F(x)$. Specifically, these elementary divisors are the polynomials $m_i(x) = p_i^{t_i}(x)$, for $0 \leq i \leq 3$, where $t_0 = 24$ and $t_1 = t_2 = t_3 = 1$. Thus, $m_F(x) = m_0(x)\, m_1(x)\, m_2(x)\, m_3(x) = p_F(x)$. We computed these elementary divisors with *Matlab* and *Macsyma*; alternatively, we could have implemented an algorithm based on Gaussian elimination (for example, see [17, 22, 23]).

5.3 An Invariant Decomposition of the State Space

We decompose the state space into a direct sum of four F-invariant subspaces $\mathcal{S} = V_0 \oplus V_1 \oplus V_2 \oplus V_3$. Doing so enables us to search for the initial fill by separately searching segments of the initial fill corresponding to these four invariant subspaces. Each invariant subspace is determined by one of the elementary divisors $m_i(x)$ $(0 \leq i \leq 3)$ of F computed in Section 5.2. In this section we describe this decomposition of $\mathcal{S}$.

We base our decomposition of $\mathcal{S}$ on the well-known *Invariant Subspace Decomposition Theorem (ISDT)*, as described by Jacob [24, p. 390], and on the more refined *Cyclic Subspace Decomposition Theorem (CSDT)* presented by Hungerford [23, p. 356]. In Gifford's cipher, however, the

[6] We also tried using the symbolic math packages *Mathematica* and *Macsyma*. We found *Mathematica* poorly suited for doing arithmetic over $\mathbb{Z}_2$, and *Macsyma* ran too slowly. At the time, we did not have easy access to *Maple* [6].

decompositions resulting from ISDT and CSDT coincide because $m_F(x) = p_F(x)$. Nevertheless, we apply both theorems because each theorem provides some additional information, which we use in Sections 5.5 and 6. Theorem 1 applies the ISDT and CSDT to Gifford's cipher.

Before applying these decomposition theorems, we review some relevant concepts from linear algebra. Let V be any n-dimensional vector space over $\mathbb{Z}_2$, and let $T : V \to V$ be any linear transformation on V. We say that V is the *direct sum* of subspaces V_1 and V_2 if and only if $V_1 \cap V_2 = \emptyset$, and for all $v \in V$, there exist $v_1 \in V_1$ and $v_2 \in V_2$ such that $v = v_1 + v_2$. In this case, we write $V = V_1 \oplus V_2$. The notation $T|V_0$ refers to the restriction of T of V_0. A subspace $V_0 \subseteq V$ is *T-invariant* if and only if $T(v) \in V_0$ for all $v \in V_0$. The *kernel* of T is the set $\ker(T) = \{v \in V : T(v) = 0\}$.

To compute the primary rational canonical form R of F, and to compute the associated similarity transformation P, it is helpful to introduce the notion of a cyclic vector, as used in CSDT. We say that $v \in V$ is a *T-cyclic vector* for V if and only if the set $\{v, T(v), \ldots, T^{n-1}(v)\}$ forms a basis of V.[7] Furthermore, we say that V is a *T-cyclic vector space* if and only if V has a cyclic vector. Every T-cyclic vector space is T-invariant, but not every T-invariant space is T-cyclic.

Applying ISDT and CSDT to Gifford's cipher yields the following decomposition of $\mathcal{S}$ into four cyclic subspaces of dimensions 24, 5, 6, and 29, respectively, corresponding to the four elementary divisors of F.

Theorem 1. *Let F be the matrix of Gifford's feedback function, and let $m_F(x) = m_0(x)\, m_1(x)\, m_2(x)\, m_3(x)$ be the minimal polynomial for F, as defined in Section 5.2. It is true that $\mathcal{S} = V_0 \oplus V_1 \oplus V_2 \oplus V_3$, where for each $0 \le i \le 3$:*

1. *V_i is F-cyclic (and hence F-invariant),*
2. *$m_i(x)$ is the minimal polynomial of $F|V_i$, and*
3. *$V_i = \ker(m_i(F))$.*

Proof. Direct application of ISDT and CSDT. Since $m_F(x) = p_F(x)$, the decompositions resulting from ISDT and CSDT coincide. We use CSDT only to establish Property 1. ■

In the rest of this section we apply Theorem 1 in two ways: In Section 5.4 we give a decomposition of F corresponding to our decomposition of $\mathcal{S}$, and in Section 5.5 we exploit Property 1 of Theorem 1 to compute a similarity transformation P.

5.4 The Primary Rational Canonical Form R of F

The *primary rational canonical form (RCF)* for F, denoted $RCF(F)$, is a block-diagonal matrix corresponding to the F-cyclic decomposition of the state space $\mathcal{S} = V_0 \oplus V_1 \oplus V_2 \oplus V_3$ given in Section 5.3. We now present this matrix and explain how we computed it.

For each $0 \le i \le 3$, let $m_i(x) = p_i^{t_i}(x) = x^{n_i} + \sum_{j=0}^{n_i-1} \alpha_{ij} x^j$ be the minimal polynomial of $F|V_i$, as defined in Section 5.2, where n_i is the degree of $m_i(x)$, and α_{ij} is the jth (binary) coefficient of $m_i(x)$. Thus, $n_0 = 24$, $n_1 = 5$, $n_2 = 6$, and $n_3 = 29$.

[7] A basis for any n-dimensional vector space V is a set of n linearly independent, spanning vectors. Every vector in V can be written uniquely as a linear combination of the basis vectors.

By definition, the RCF for F is the 64×64 block-diagonal matrix

$$R = \begin{pmatrix} R_0 & & & \\ & R_1 & & \\ & & R_2 & \\ & & & R_3 \end{pmatrix}, \tag{19}$$

where for each $0 \leq i \leq 3$, block R_i is the *companion matrix* [22, p. 230]

$$R_i = \begin{pmatrix} 0 & 0 & \dots & 0 & \alpha_{i0} \\ 1 & 0 & \dots & 0 & \alpha_{i1} \\ 0 & 1 & \dots & 0 & \alpha_{i2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & \alpha_{i(n_i-1)} \end{pmatrix}. \tag{20}$$

The matrix R is a canonical representation of the equivalence class of all matrices similar to F; it is unique up to the order of the blocks.

Each companion matrix R_i is a lower-diagonal matrix, whose last column consists of the coefficients in the associated minimal polynomial $m_i(x)$. For example,

$$R_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \tag{21}$$

corresponding to the minimal polynomial $m_1(x) = x^5 + x^3 + x^2 + x + 1$ for $F|V_1$.

Appendix B gives the matrix R for Gifford's cipher, which we computed by inspection from the elementary divisors $m_0(x), \dots, m_3(x)$ of F. The blocks of R have dimensions 24, 5, 6, 29, and ranks 23, 5, 6, 29, respectively. Block R_0 of dimension 24 is the only singular block. The singularity arises from the last column, which is zero because the corresponding elementary divisor is $m_0(x) = x^{24}$.

Block R_0 plays a special role because it represents a nilpotent transformation with $R_0^{24} = 0$. To see that R_0 is nilpotent, observe that the only ones in R_0 are along the lower diagonal, which happens whenever the corresponding elementary divisor is a power of x. Note that R_0 is the only nilpotent block of R; in fact, when $p_F(x) = m_F(x)$, there can be at most one nilpotent block. Since the nilpotent block determines the leader length of any state sequence, the maximum leader length of any state sequence generated by Gifford's cipher is 24 states. For an independent proof of this fact, see Ronse [37, p. 54].

As explained in Section 7, our attack separately searches segments of the key corresponding to the four blocks of R.

5.5 A Similarity Transformation P from F to R

A *similarity matrix* from F to R is any invertible binary matrix P such that

$$F = P^{-1} R P. \tag{22}$$

Our attack uses such a matrix to move between the original and decomposed state spaces. For example, to check any candidate key in the decomposed world, we apply P^{-1} to transform the key into the original world, where we check if it decrypts the ciphertext. Appendix B presents our similarity matrix P and its inverse P^{-1}. In this section, we discuss the role of this matrix and explain how we computed it.

As illustrated in Figure 2, the matrix P transforms any fill $s \in \mathcal{S}$ into a vector of "subfills" $Ps = (d_0, d_1, d_2, d_3)$ corresponding to the F-invariant decomposition $\mathcal{S} = V_0 \oplus V_1 \oplus V_2 \oplus V_3$ derived in Section 5.3. Because our matrix P^{-1} does not preserve the block structure of R, individual subfills cannot be transformed separately by means of P^{-1}; instead, all subfills must be known before the transformation can be applied.

It is significant that matrix P^{-1} is dense. As a result, slight errors in any subfill multiply when the subfill is transformed by P^{-1}. Unfortunately, we have no reason to believe that any sparse or block-structured P exists.

Since F and R are similar matrices [23, p. 360], a similarity matrix from F to R exists. Although $RCF(F)$ is unique up to the order of its four blocks, there are many similarity matrices P: specifically, there are $2^{23} \cdot 31 \cdot 63 \cdot (2^{29} - 1)$ such matrices, as we prove in [5]. Our task was to compute one of these P's.

Although it would be possible to compute a similarity matrix from F to R as a byproduct of the computation of $RCF(F)$, the math packages we used to compute $RCF(F)$ did not do so. Therefore, we devised our own algorithm for computing a similarity matrix.

We define the *Similarity Transformation Problem (STP)* as follows.

Similarity Transformation Problem (STP)

Input: A positive integer n, and any similar $n \times n$ binary matrices A and B.

Output: Any invertible binary matrix P such that $PA = BP$, where all arithmetic is performed over $GF(2)$.

We are particularly interested in the special case of this problem in which the matrix B is given in primary rational canonical form. We refer to this special case of the problem as *STPR*. Although STP is syntactically similar to the *Graph Isomorphism Problem (GI)* [12], STP is less constrained than GI because GI additionally requires P to be a permutation matrix.[8]

There is a straightforward polynomial-time algorithm for solving STP: Let $n = 64$ and consider each of the n^2 entries of P as an unknown variable. Compute the values of these unknowns by solving the n^2 linear equations defined by the equation $PF - RP = 0$, and check if the resulting P matrix is invertible. Using Gaussian elimination, this straightforward algorithm would take $(n^2)^3 = n^6$ steps, which would be inconveniently slow since $64^6 = 2^{36}$, though it ought to be possible to speed up this calculation by exploiting the sparse nature of F. To our surprise, we found no insightful previous work on the problem of computing similarity matrices efficiently. For example, Gill [17, pp. 17–19] gives only the slow straightforward $\Theta(n^6)$ algorithm.

To compute our P matrix, we devised a faster method that exploits our cyclic decomposition of $\mathcal{S}$. This method requires finding a F-cyclic vector for each of the F-cyclic subspaces.[9] Our method runs in time $O(\sum_{i=0}^{3} \gamma_i)$, where γ_i is the time to find a F-cyclic vector for cyclic subspace V_i.

For our decomposition of F, finding F-cyclic vectors was easy. Because subspace V_0 corresponds to a nilpotent transformation, exactly half of all vectors in V_0 are F-cyclic; it suffices to find a vector $v \in V_0$ such that $F^{23}(v) \neq 0$. For the other subspaces, every nonzero vector is F-cyclic.

[8] A *permutation matrix* is a binary matrix that has exactly one one in every row and column.

[9] For a definition of cyclic vector, see Section 5.3.

This statement follows from the fact that, in Gifford's cipher, each of the elementary divisors $m_1(x)$, $m_2(x)$, $m_3(x)$ is irreducible. For more details, see our companion paper [5]. For each cyclic subspace $V_i = \ker(m_i(F))$ $(0 \leq i \leq 3)$, we found a F-cyclic vector in the standard basis set that we computed by Gaussian elimination.

We computed our P^{-1} matrix as follows. For each $0 \leq i \leq 3$, let $v_i \in V_i$ be any F-cyclic vector; let $n_i = \dim(V_i)$; and let $q_{ij} = F^j(v_i)$ for $0 \leq j \leq n_i - 1$. We defined P^{-1} to consist of the columns q_{ij}, for $0 \leq i \leq 3$ and $0 \leq j < n_i$. Although we still performed Gaussian elimination (to compute our basis set), we did so over smaller systems of equations consisting of at most $n - n_1 = 64 - 5 = 59$ equations, which took less than 2 seconds of computation time on our Sparcstation. The correctness of our method for computing P follows from Proposition 1. In this proposition, Q plays the role of P^{-1}; thus Q^{-1} is a similarity matrix from F to R.

Proposition 1. *Let $\mathcal{S} = V_0 \oplus V_1 \oplus V_2 \oplus V_3$ be the cyclic decomposition of the state space of Gifford's cipher given in Theorem 1, and let R be the primary rational canonical form of the feedback matrix F. For each $0 \leq i \leq 3$, let $n_i = \dim(V_i)$; let $v_i \in V_i$ be any F-cyclic vector; and let Q be the binary matrix whose columns are the vectors $q_{ij} = F^j(v_i)$ for $0 \leq j \leq n_i - 1$. The matrix Q represents a similarity transformation from R to F.*

Proof. We must show that Q is invertible and that $FQ = QR$. The invertibility of Q follows immediately from the fact that v_i is an F-cyclic vector for each cyclic subspace V_i, for $0 \leq i \leq 3$. To prove $FQ = QR$, we show that the corresponding equations holds for each block of R. To this end, let $0 \leq i \leq 3$.

The ith block of QR consists of the columns $q_{i1}, q_{i2}, \ldots, q_{i(n_i-1)}, \sum_{j=0}^{n_i-1} \alpha_{ij} q_{ij}$, where α_{ij} is the jth coefficient in the minimal polynomial $m_i(x)$ for $F|V_i$ given in Section 5.2. The corresponding columns of FQ are $Fq_{i0}, Fq_{i1}, \ldots, Fq_{i(n_i-1)}$. The equality of each pair of corresponding columns follows the fact that v_i is an F-cyclic vector and that $m_i(F)q_{i0} = 0$. ■

As a check of correctness, we verified that $F = P^{-1}RP$. Finally, we note that this identity further confirms the correctness of our characteristic polynomial $p_F(x)$: the matrix R is unique up to the order of its blocks; polynomial $p_F(x)$ determines R; and the identity $F = P^{-1}RP$ proves that F and R are similar.

6 The Probability Distribution of Leader and Cycle Lengths

For each initial fill $s_0 \in \mathcal{S}$, Gifford's filter generator computes an eventually periodic sequence of keybytes $G^*(s_0) = \{K_t\}_{t=0}^{\infty}$. Each sequence consists of a leader and a cycle, where the *leader* is the initial (transient) nonperiodic part, and the *cycle* is the periodic part. Although long periods do not guarantee high security, short periods create serious weaknesses. For example, if the period of the keystream is smaller than the length of the plaintext stream, then some keybytes will be reused to encrypt two or more plaintext bytes. Therefore, it is important to know the probability distribution of the leader and cycle lengths of the keystream. Similarly, it is important to understand the related periodic properties of the underlying sequence of register states $G^*_{\mathcal{S}}(s_0) = \{s_t\}_{t=0}^{\infty}$.

From the exponents of the elementary divisors of F, we compute the exact probability distribution of the leader and cycle lengths of the state stream. For example we prove, that excluding the degenerate $s_0 = 0$, leader lengths range from 0 to 24 states, and cycle lengths range from 21 to $349,502,963,061 \approx 3.5 \times 10^{11}$ states.

The leader and cycle lengths of the state stream are upper bounds on the leader and cycle lengths of the keystream. It is possible, however, that the keystream repeats before the state stream repeats—though if this happens, the length of the keystream cycle must properly divide the length of the corresponding state stream cycle. Since we do not know how to compute the exact distribution of leader and cycle lengths in the keystream, we experimentally looked for any initial fill whose keystream repeats before the state stream repeats. We found none.

6.1 Leaders, Cycles, and Maximum Periods

For any initial fill $s_0 \in \mathcal{S}$, let $\lambda_f(s_0)$ and $\pi_f(s_0)$ denote, respectively, the *leader length* and *cycle length* of the eventually periodic sequence $G^*_{\mathcal{S}}(s_0) = \{s_t\}_{t=0}^{\infty}$. Thus, $\pi_f(s_0) = \min\{p \in \mathbb{N} : s_{t+p} = s_t$ for all sufficiently large $t \in \mathbb{N}\}$. If $\lambda_f(s_0) = 0$, then we say that the sequence $G^*_{\mathcal{S}}(s_0)$ is *strictly periodic*.

Also, let λ^*_f and π^*_f denote, respectively, the maximum leader and cycle lengths over all initial fills. Thus,

$$\pi^*_f = \max\{\pi_f(s) \ : \ s \in \mathcal{S}\} \qquad \text{and} \qquad \lambda^*_f = \max\{\lambda_f(s) : s \in \mathcal{S}\}. \tag{23}$$

Note that $\pi_f(0) = 1$ and $\pi^*_f \leq 2^{64} - 1$.

In addition, let $\mathbb{P}_f = \{\pi_f(s) : s \in \mathcal{S}\}$ denote the set of all possible periods; and for any $p \in \mathbb{N}$, let $S_f(p) = \{s \in \mathcal{S} : \pi_f(s) = p\}$ be the set of states with period p. In the rest of this section we compute $\mathbb{P}_f$, S_f, λ^*_f, π^*_f, and an exact probability distribution on $\mathbb{P}_f$ for uniformly chosen initial fills. Apparently, Gifford did not know any of these quantities.

6.2 Periodic Properties of F

For any $s_0 \in \mathcal{S}$, $\pi_f(s_0)$ can be computed in terms of the exponents of the elementary divisors of F that generate the subspace to which s_0 belongs. In this section we explain how to perform this calculation.

Let $f(x) \in \mathbb{Z}_2[x]$. The *exponent of* f, denoted $\exp(f)$, is the least positive integer r such that $f(x) \mid x^r - 1$. If there is no such integer r, we say that the exponent is 0.

Theorem 2 states a relationship between exponents and periods. This relationship arises from applying the definition of period to the state stream $\{F^t(s_0)\}_{t=0}^{\infty}$: for any $s_0 \in \mathcal{S}$ and for any $t \geq \lambda^*_f$, it is true that $F^{\pi_f(s_0)+t}s_0 = F^t s_0$ and thus $(F^{\pi_f(s_0)} - I)F^t s_0 = 0$.

Theorem 2. *Let $\mathcal{S} = V_0 \oplus V_1 \oplus V_2 \oplus V_3$ be the direct sum decomposition of the state space of Gifford's cipher given in Theorem 1. For $0 \leq i \leq 3$, let $m_i(x) = p_i^{t_i}(x)$ be the elementary divisors of F given in Section 5.2, and let $e_i = \exp(m_i)$. Also, for each $1 \leq i \leq 3$, define $\beta_i = \min\{2^j : j \in \mathbb{N}$ and $2^j \geq t_i\}$, and let $\beta = \max\{\beta_1, \beta_2, \beta_3\}$. Let $v = (v_0, v_1, v_2, v_3) \in \mathcal{S}$. For each $1 \leq i \leq 3$, it is true that:*

1. *If $v_i \neq 0$, then $\pi_f(v_i) = \beta_i e_i = e_i$.*

2. *$\pi_f(v) = \beta' \operatorname{lcm}\{e_i : v_i \neq 0$ and $1 \leq i \leq 3\}$, where $\beta' = \max\{\beta_i : v_i \neq 0$ and $1 \leq i \leq 3\} = 1$.*

3. *$\pi^*_f = \beta \operatorname{lcm}(e_1, e_2, e_3) = \operatorname{lcm}(e_1, e_2, e_3)$.*

4. *$F^{t_0}(v) = F^{24}(v) \in V_1 \oplus V_2 \oplus V_3$, and*

5. $\lambda_f^* = t_0 = 24$.

Proof. First, observe that $t_0 = 24$, $t_1 = t_2 = t_3 = 1$, and $\beta = \beta_1 = \beta_2 = \beta_3 = 1$. The theorem follows from Theorem 1 and from Berlekamp [2, pp. 150–151]. For example, for any $1 \leq i \leq 3$, we prove Property 1 as follows: By Berlekamp [2, p. 151], $\exp(m_i) = \beta_i e_i$; and by Theorem 1, $m_i(F)(v_i) = 0$ since $V_i = \ker(m_i(F))$. Combining these facts yields $(F^{\beta_i e_i} - I)(v_i) = 0$. ■

Thus, each initial fill in $V_1 \oplus V_2 \oplus V_3$ generates a strictly periodic sequence, and each initial fill in V_0 generates a leader converging to the zero fill. Although other books (*e.g.* Ronse [37] and Gill [17]) have similar underlying theorems, we have not seen any previous application of this result to a real cipher.

6.3 Exponents of the Elementary Divisors of F

To interpret Theorem 2 numerically, we need to compute the exponents of the elementary divisors of F, given in Section 5.2. To carry out these calculations, it is helpful first to review some basic concepts from finite field theory, including the notions of cyclotomic and primitive polynomials. Throughout we work over the base field $GF(2)$.[10]

Let n be any positive integer. The *nth roots of unity* are the roots of the polynomial $x^n - 1$. These roots form a multiplicative cyclic group. If ζ is an nth root of unity that generates this group, then ζ is said to be a *primitive nth root of unity.* The *nth cyclotomic polynomial* $C_n(x)$ is the monic polynomial $C_n(x) = \prod_{i=1}^{r}(x - \zeta_i)$, where $\zeta_1, \zeta_2, \ldots, \zeta_r$ are the distinct primitive n^{th} roots of unity.

Cyclotomic polynomials are useful in computing exponents. Let $f(x) \in \mathbb{Z}_2[x]$ be any irreducible polynomial. Since all roots of $f(x)$ over $\mathbb{Z}_2$ have the same order in any extension field of $\mathbb{Z}_2$, it is true that $\exp(f)$ is the order of its roots in that extension field. Therefore, if $f(x)$ divides $C_k(x)$ for some cyclotomic polynomial $C_k(x)$, it follows that $\exp(f) = k$.

Any irreducible polynomial $f(x)$ of degree n is said to be a *primitive polynomial* if and only if $\exp(f) = 2^n - 1$. Any n-stage shift register achieves the maximum period of $2^n - 1$ states if and only if its characteristic polynomial is primitive [1, Ch. 5].

Proposition 2 computes the exponents of the elementary divisors of F. First, we review the following two well-known lemmas, which we apply to compute exponents.

Lemma 1. *Let n be any positive integer. If $S = \{f(x) : f(x)$ is an irreducible polynomial over $\mathbb{Z}_2$ of degree dividing $n\}$, then $x^{2^n} + x = \prod_{f \in S} f(x)$.*

Proof. See Berlekamp [2, p. 103]. ■

Lemma 2. *Let $n \in \mathbb{Z}^+$ and let $C_d(x)$ be the dth cyclotomic polynomial over $\mathbb{Z}_2$. If $2 \not| \, n$, then $x^n + 1 = \prod_{d|n} C_d(x)$.*

Proof. See Berlekamp [2, p. 91]. ■

[10] For a review of finite field theory, see Berlekamp [2] and Peterson and Weldon [35].

Proposition 2. *For each $0 \le i \le 3$, let $e_i = \exp(m_i)$ be the exponent of the elementary divisor $m_i(x)$ of F defined in Section 5.2. It is true that $e_0 = 0$, $e_1 = 31$, $e_2 = 21$, and $e_3 = 2^{29} - 1$. Consequently, $m_1(x)$ and $m_3(x)$ are primitive polynomials, but $m_0(x)$ and $m_2(x)$ are not primitive.*

Proof. We compute the exponent of each elementary divisor separately.

$m_0(x)$ is the polynomial x^{24}. Since x does not divide $x^n - 1$ for any positive integer n, it follows that $e_0 = 0$ and $m_0(x)$ is not primitive.

$m_1(x)$ is a degree 5 irreducible polynomial. We find the unique cyclotomic polynomial $C_k(x)$ such that m_1 divides $C_k(x)$ to establish that $e_1 = k$. By Lemma 1, m_1 divides $x^{2^5} + x = x(x^{31}+1)$; and by Lemma 2, $x^{31} + 1 = C_1(x)\, C_{31}(x)$. Because $C_1(x) = x + 1$, it follows that $m_1(x)$ divides $C_{31}(x)$. Therefore, $e_1 = 31$ and $m_1(x)$ is primitive.

$m_2(x)$ is a degree 6 irreducible polynomial; therefore, $e_2 \le 2^6 - 1 = 63$. By Lemma 2, $x^{63}+1 = C_1(x)\, C_3(x)\, C_7(x)\, C_9(x)\, C_{21}(x)\, C_{63}(x)$. It is easy to verify that $m_2(x)$ divides $C_{21}(x)$, but $m(x)$ does not divide $C_{63}(x)$, $C_9(x)$, or $C_7(x)$. Hence, $e_2 = 21$ and $m_2(x)$ is not primitive.

$m_3(x)$ is a degree $d = 2^{29} - 1$ polynomial; therefore, $e_3 | d$. We will prove that $e_3 = d$ by showing that $m_3(x)$ is primitive. Let α be any root of $m_3(x)$. To prove that $m_3(x)$ is primitive, it suffices to verify that $\alpha^r \ne 1$ for all $r < d$ such that $r|d$. Since $2^{29} - 1 = 233 \cdot 1103 \cdot 2089$ is the prime factorization of d, only six r must be checked. To carry out this verification, we implemented and ran an algorithm in Appendix C of Peterson [35]. Thus, $m_3(x)$ is irreducible and primitive and $e_3 = 2^{29} - 1$. ■

As a partial check of our calculations, note that Marsh [32] also lists the exponents of $m_1(x)$ and $m_2(x)$ as 31 and 21, respectively. We could not find any table that lists $m_3(x)$.

6.4 An Exact Characterization of Leaders and Cycles

We apply Theorem 2 and Proposition 2 to characterize exactly the set of eventually periodic state sequences that can be generated by Gifford's cipher. As part of this characterization, we compute the maximum period π_f^*, and the exact probability distribution of leader and cycle lengths taken over all possible initial fills.

As computed in Corollary 1, the maximum period of any state sequence is determined by invariant subspaces V_1, V_2, and V_3. Since elementary divisor $m_2(x)$ is not primitive, its contribution to π_f^* is not as large as possible for a dimension 6 subspace of $\mathcal{S}$. Moreover, 24 bits of the key are wasted in determining a short leader of length at most $\lambda_f^* = 24$; these 24 bits contribute nothing to the period.

Corollary 1. $\pi_f^* = 349,502,963,061$.

Proof. By Theorem 2 and Proposition 2, $\pi_f^* = \mathrm{lcm}(31, 21, 2^{29} - 1) = 31 \cdot 21(2^{29} - 1) = 349,502,963,061$, since the exponents 31, 21, $2^{29} - 1$ are relatively prime. ■

Table 1 characterizes completely the state sequences that can be generated by Gifford's cipher. Each initial fill s_0 determines four subfills $Ps_0 = (d_0, d_1, d_2, d_3)$ in our invariant decomposition of $\mathcal{S}$. These subfills belong to the invariant subspaces V_0, V_1, V_2, V_3 of dimensions 24, 5, 6, 29, respectively. There are eight possible periods $\mathbb{P}_f = \{0, 21, 31, 2^{29}-1, 21\cdot 31, 21\cdot 2^{29}-1, 31\cdot 2^{29}-1, 21\cdot 31\cdot 2^{29}-1\}$, corresponding to the eight possible ways in which up to three of the subfills d_1, d_2, and d_3 can be zero. In addition, cycles achieving these periods can occur with or without a leader, depending on whether subfill d_0 is zero. Thus, there are 16 equivalence classes of initial fills. For example, a 31-state cycle is created whenever $d_1 \neq 0$ and $d_2 = d_3 = 0$. Similarly, a maximum-length cycle occurs whenever $d_1 d_2 d_3 \neq 0$. For any subspace U, let U^+ denote the set of nonzero elements of U. Thus, $S_f(\pi_f^*) = V_0 \oplus V_1^+ \oplus V_2^+ \oplus V_3^+$.

As computed in Table 1, the probability of generating any one of these 16 possible equivalence classes of sequence lengths can be computed from the dimension of the subspace that generates the given sequence length. Specifically, for any subspace U, the probability that a randomly chosen initial fill lands in U is $|U| / |\mathcal{S}|$. For example, the probability of generating a length 31 cycle (with nonzero leader) is $|V_0^+ \oplus V_1^+| / |\mathcal{S}| = (2^{24} - 1)(2^5 - 1)/2^{64} \approx 2^{-(64-29)} = 2^{-35}$.

The shortest cycle, however, comprises 21 and not 31 states. Appendix D lists such a cycle. For dimension 6 subspace V_2, elementary divisor $m_2(x)$ is not primitive and generates one of three submaximal-length cycles of length 21. By contrast, because elementary divisors $m_1(x)$ and $m_3(x)$ are primitive, they always generate maximal-length cycles of lengths $2^5 - 1 = 31$ and $2^{29} - 1$, respectively.

With very high probability (> 0.9998) the cycle will contain at least $2^{29} - 1 \approx 5.4 \cdot 10^8$ states, and the maximum period $\pi_f^* \approx 3.5 \cdot 10^{11}$ occurs with probability approximately 0.9536. Yet with non-negligible probability of $2^{-11} \approx 0.0005 = 0.05\%$, the cycle length is only $2^{29} - 1 < 10^9$. This fact partially contradicts Gifford's [15, p. 465] experimental finding that "the period has consistently been found to exceed 10^9."[11]

[11] Gifford [15, p. 465] did not specify whether his experiments looked for cycles in the keystream or in the state stream.

Table 1: Probability distribution of leader and cycle lengths in the sequence of register states generated by Gifford's cipher with randomly chosen initial fill. The linear feedback matrix F yields a decomposition of the space of register states $\mathcal{S}$ into a direct sum of four F-invariant subspaces $\mathcal{S} = V_0 \oplus V_1 \oplus V_2 \oplus V_3$ of dimensions 24, 5, 6, and 29, respectively. Each 64-bit initial fill corresponds to a vector of four subfills of lengths 24, 5, 6, and 29, respectively. Subspace V_0 completely determines the leader and does not affect the period; the other invariant subspaces determine the period. There are 16 subspaces U of $\mathcal{S}$ corresponding to the 16 possible ways in which up to four of the subfills can be zero. For any such U, all initial fills $s_0 \in U$ generate state sequences with the same cycle length $\pi_f(s_0)$; in addition, if $\lambda_f(s) = 0$ for any $s \in U$, then $\lambda_f(s) = 0$ for all $s \in U$. For each of these 16 subspaces U, we list its cardinality, the associated leader and cycle lengths, and the probability that a randomly chosen initial fill lands in U. This probability is $|U|/|\mathcal{S}| = |U|/2^{64}$. For any subspace U, let U^+ denote the nonzero elements of U. The shortest non-degenerate cycles have length 21; these cycles are produced from initial fills in subspace V_2^+, whose characteristic polynomial $m_2(x)$ is not primitive. With probability $|V_1^+ \oplus V_2^+ \oplus V_3^+|/2^{64} + |V_0^+ \oplus V_1^+ \oplus V_2^+ \oplus V_3^+|/2^{64} \approx 0.9536$, a randomly chosen initial fill will generate a cycle with maximum period $\pi_f^* = 21 \cdot 31 \cdot 2^{29} - 1 \approx 3.5 \times 10^{11}$. The maximum leader length is $\lambda_f^* = 24$.

Subspace U with initial fill $s_0 \in U$	Subspace size $\lvert U\rvert$	Period $\pi_f(s_0)$	Leader length $\lambda_f(s_0)$	Prob$[s_0 \in U]$ $= \lvert U\rvert/2^{64}$
$\{0\}$	1	1	0	$1/2^{64}$
V_0^+	$2^{24} - 1$	1	$\le \lambda_f^*$	$\approx 1/2^{40}$
V_2^+	63	21	0	$\approx 1/2^{58}$
$V_0^+ \oplus V_2^+$	$(2^{24} - 1)63$	21	$\le \lambda_f^*$	$\approx 1/2^{34}$
V_1^+	31	31	0	$\approx 1/2^{59}$
$V_0^+ \oplus V_1^+$	$(2^{24} - 1)31$	31	$\le \lambda_f^*$	$\approx 1/2^{35}$
$V_1^+ \oplus V_2^+$	$63 \cdot 31$	$21 \cdot 31$	0	$\approx 1/2^{53}$
$V_0^+ \oplus V_1^+ \oplus V_2^+$	$63 \cdot 31(2^{24} - 1)$	$21 \cdot 31$	$\le \lambda_f^*$	$\approx 1/2^{29}$
V_3^+	$2^{29} - 1$	$2^{29} - 1$	0	$\approx 1/2^{35}$
$V_0^+ \oplus V_3^+$	$(2^{24} - 1)(2^{29} - 1)$	$2^{29} - 1$	$\le \lambda_f^*$	$\approx 1/2^{11} \approx 0.0005$
$V_2^+ \oplus V_3^+$	$63(2^{29} - 1)$	$21(2^{29} - 1)$	0	$\approx 1/2^{29}$
$V_0^+ \oplus V_2^+ \oplus V_3^+$	$63(2^{24} - 1)(2^{29} - 1)$	$21(2^{29} - 1)$	$\le \lambda_f^*$	$\approx 1/2^5 \approx 0.0308$
$V_1^+ \oplus V_3^+$	$31(2^{29} - 1)$	$31(2^{29} - 1)$	0	$\approx 1/2^{30}$
$V_0^+ \oplus V_1^+ \oplus V_3^+$	$31(2^{24} - 1)(2^{29} - 1)$	$31(2^{29} - 1)$	$\le \lambda_f^*$	$\approx 1/2^6 \approx 0.0156$
$V_1^+ \oplus V_2^+ \oplus V_3^+$	$31 \cdot 63(2^{29} - 1)$	π_f^*	0	$\approx 1/2^{24}$
$V_0^+ \oplus V_1^+ \oplus V_2^+ \oplus V_3^+$	$31 \cdot 63(2^{24} - 1)(2^{29} - 1)$	π_f^*	$\le \lambda_f^*$	≈ 0.9536

7 Attacks

The decomposition of the state space into a direct sum of invariant subspaces makes possible a variety of cryptanalytic attacks on filter generators that search segments of the key corresponding to this decomposition. In this section, we outline four such ciphertext-only attacks applied to Gifford's cipher:

1. a simple 2^{40}-step attack based on exhaustive search,
2. our novel time-space tradeoff attack, which uses 2^{27} steps and 2^{18} bytes of memory,
3. a 2^{29}-step correlation attack that adapts a correlation procedure of Siegenthaler [41], and
4. an application of Hellman's [21] time-space tradeoff, which requires a short chosen-plaintext and a 2^{40}-step precomputation.

These attacks have differing advantages and requirements. Attack (3) requires a slight statistical weakness in the output function; the other attacks require no such weakness. Attacks (2) and (3) require ciphertext from one news article (a few thousand bytes); attack (1) requires only seven bytes of ciphertext from ASCII-encoded English; and attack (4) requires only approximately one dozen such bytes of ciphertext. We implement our time-space tradeoff to demonstrate one effective method for breaking Gifford's cipher.

Combining attacks (2) and (3) yields an even faster attack: for example, the cryptanalyst could first search subregisters $\mathcal{R}_1$ and $\mathcal{R}_2$ with attack (3), and then search subregister $\mathcal{R}_3$ with attack (2). This combined attack would require only approximately 2^{16} steps on average. For our implementation, we estimate this attack would run in less than one minute using eight Sparcstations.

The rest of this section is organized in six parts: Section 7.1 gives an overview of the four attacks. Sections 7.2 through 7.5 explain each of the attacks, and Section 7.6 describes our implementation of our time-space tradeoff.

7.1 Overview of Attacks

This section explains three ideas that we apply to break Gifford's cipher. First, we summarize how our attacks exploit the decomposition of the feedback function. Second, we reveal how, in BCIS practice, Gifford's cipher leaked bits of the keystream. Third, we estimate the unicity distance for Gifford's cipher.

7.1.1 Exploiting the Decomposition of F

Each of our attacks exploits the decomposition $\mathcal{S} = V_0 \oplus V_1 \oplus V_2 \oplus V_3$ of the state space into the four F-invariant subspaces defined in Section 5. As illustrated in Figure 2, the central idea is to decompose the shift register into four smaller subregisters $\mathcal{R}_0$, $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$ and to attack them separately. Each subregister corresponds to one of the F-invariant subspaces of dimensions 24, 5, 6, and 29, respectively. Determining the bits of any subregister yields a big payoff: once the state of a subregister is known, its state at all future times can be easily calculated. Specifically, for $0 \leq i \leq 3$, if d_i is the initial state of subregister $\mathcal{R}_i$, then $R_i^t d_i$ is the state of $\mathcal{R}_i$ at time $t \in \mathbb{N}$, where R_i is the ith block of the matrix R given in Section 5.4. Moreover, since R_1, R_2, and R_3 are nonsingular, knowing the state of $\mathcal{R}_i$ for any $1 \leq i \leq 3$ determines all previous states of $\mathcal{R}_i$.

Because the feedback function R_0 is nilpotent, the 24-bit subregister $\mathcal{R}_0$ determines the leader and remains 0 after 24 iterations. Therefore, $\mathcal{R}_0$ affects at most the first 24 bytes of any ciphertext. Since a typical news article is approximately 10,000 bytes long, $\mathcal{R}_0$ has essentially no effect on article encryption. In particular, the cryptanalyst can first determine the subfills for $\mathcal{R}_1$ through $\mathcal{R}_3$, beginning with the the 25th ciphertext byte. Then, the cryptanalyst can determine the initial subfill of $\mathcal{R}_0$. Thus, although Gifford's cipher nominally uses a 64-bit key, the effective key length is only 40 bits. Most of these 40 bits are used to determine the initial state of $\mathcal{R}_3$. For simplicity, for the rest of this section, we assume that the initial subfill of $\mathcal{R}_0$ is zero.

The similarity matrix P derived in Section 5.5 maps states of the main register to states of the subregisters. Thus, for any initial fill $s_0 \in \mathcal{S}$, if $Ps_0 = (d_0, d_1, d_2, d_3)$ are the initial subfills corresponding to s_0, then $PF^t s_0 = R^t(d_0, d_1, d_2, d_3) = (R_0^t d_0, R_1^t d_1, R_2^t d_2, R_3^t d_3)$ are the subfills at time $t \in \mathbb{N}$.

Our attacks check candidate subfills in different ways. The exhaustive search attack maps an entire vector of candidate subfills back to the main register; our time-space tradeoff optimizes this idea by hashing into a table derived from the ciphertext. Our adaptation of Siegenthaler's correlation attack separately checks each subfill by correlating its state sequence with the ciphertext stream; our application of Hellman's time-space tradeoff checks candidate subfills using precomputed tables based on a sequence of chosen-plaintext.

7.1.2 Leaked Keystream Bits

In the BCIS, each news article was represented as a sequence of 8-bit extended-ASCII characters: each 7-bit ASCII character was placed in an 8-bit byte, with the high-order (leftmost) bit held as the constant zero. With this format, special 8-bit graphical characters (with high-order bit one) could also be transmitted. As a consequence, however, for standard news articles the high-order bit of each plaintext character was left as the constant zero. Therefore, except for rare graphical characters, every byte of ciphertext leaked at least one bit of the keystream. With high probability, the second-order bit was also leaked because most plaintext bytes represented lower-case alphabetic characters. Consequently, a ciphertext-only attack against Gifford's cipher reduces to a known-plaintext attack. At the cost of an additional computation, this serious weakness could have be avoided by compressing the data before encryption. Although our time-space tradeoff depends on this weakness, our other attacks do not.

7.1.3 Unicity Distance of Gifford's Cipher

When evaluating any cipher, it is important to know its unicity distance—the amount of ciphertext required in a ciphertext-only attack for the expected number of spurious decipherments to be approximately zero [40]. To estimate the unicity distance, assume the cipher is used to encrypt standard English, which has a redundancy of approximately 3.2 bits/character. Modeling Gifford's cipher as a random cipher with 40-bit key, the unicity distance is approximately $40/3.2 = 12.5$ characters. When used to encrypt ASCII-encoded English, the unicity distance is even smaller: If the information rate remains at 1.5 bits/character but the plaintext is represented in 8-bit bytes, then the unicity distance is approximately $40/(8-1.5) \approx 6.2$ bytes. In either case, ciphertext from one newspaper article ($\approx 10,000$ bytes) is more than sufficient to recover the initial fill uniquely. Furthermore, our application of Hellman's time-space tradeoff requires only an amount of ciphertext close to these information-theoretic bounds.

7.2 Exhaustive Search

The simplest way to break Gifford's cipher is to search exhaustively over all 2^{40} possible fills of subregisters $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$. Assuming $\mathcal{R}_0$ is zero, for each sequence of candidate subfills $d = (0, d_1, d_2, d_3)$ of these subregisters, check d as follows: Map the vector back to the original world by computing $P^{-1}d$, and use the resulting candidate fill to generate a candidate keystream. With known plaintext, compare the candidate keystream against the known keystream. With ciphertext only, compute the XOR of the candidate keystream with the ciphertext stream, and check if the resulting candidate plaintext stream appears to be valid plaintext language. For example, Ganesan and Sherman [10, 11] explain several statistical methods for recognizing valid plaintext.

Although this method may be too slow for some cryptanalysts, in Section 7.3, we show how to reduce its time drastically by using a time-space tradeoff. Nevertheless, 2^{40} steps is a significant shortcut over naively searching all 2^{64} initial fills in the main shift register.

7.3 Our Time-Space Tradeoff

To speed up the exhaustive search described in Section 7.2, we use a time-space tradeoff. Our algorithm is based on the following idea. Given any ciphertext of length $N = 2^n$, it is not necessary to search all 2^{29} subfills of subregister $\mathcal{R}_3$. Instead, it is sufficient to search every Nth subfill of $\mathcal{R}_3$ if one checks this subfill against each of the N positions in the ciphertext. Moreover, by storing the high-order bits of ciphertext in a hash table, it is possible to perform this checking in expected constant time and thereby speed up the search by a factor of N on average. Figure 5 gives pseudocode for our attack.

Since the minimal polynomial $m_3(x)$ for invariant subspace V_3 is primitive (see Proposition 2), subregister $\mathcal{R}_3$ always generates a maximal cycle of $2^{29} - 1$ states for any nonzero initial subfill. Different initial subfills follow this cycle from different starting points. Our attack searches every Nth subfill in this cycle. Because successive bytes of the ciphertext were produced in part using successive states of $\mathcal{R}_3$, by searching every Nth state of the cycle, the cryptanalyst is guaranteed to find at least one state that was used to encrypt some ciphertext byte.

It is easy to compute every Nth subfill of $\mathcal{R}_3$ on-line—simply compute $(R_3^N)^j d_3$ for $j = 0, 1, \ldots$ for any initial subfill d_3 of $\mathcal{R}_3$. To save a constant factor in time, however, we precompute a table T_3 of these $2^{29}/N = 2^{29-n}$ values.

Since subregisters $\mathcal{R}_1$ and $\mathcal{R}_2$ also affect the keystream, for each subfill of $\mathcal{R}_3$ tried, we also search over all $2^5 2^6 = 2^{11}$ possible subfills of these two registers. Therefore, the expected running time of the attack is $2^{29-n}2^{11} = 2^{40-n}$ steps. Our precomputation of table T_3 takes only 2^{29-n} steps. As we demonstrate, for suitable parameter choices, the time required to process hashing collisions and false alarms has a low-order effect in comparison to these bounds. If the initial subfills for $\mathcal{R}_1$ and $\mathcal{R}_2$ are already known (say, by using a correlation attack), then our algorithm would run 2^{11} times faster.

For each byte position of the ciphertext, we compute its "signature" σ, consisting of l consecutive high-order bits of the ciphertext beginning at the specified position. A hash table of size 2^l stores all ciphertext positions corresponding to each signature. To check any candidate vector of subfills d, we compute the sequence of high-order bits in the keystream $G^*(P^{-1}d)$ generated by d and check if this sequence matches any position signature in the hash table.

The space usage is controlled by parameters l and l', which also affect the collision and false-alarm rates. The hash table has 2^l slots (each of approximately $\lceil 2^{n-l} \rceil$ items), and the precomputed table T_3 requires 2^{29-n} entries (each of four bytes). The parameter l' specifies the number of bytes of ciphertext bytes that are used in a secondary test of candidate fills.

Candidate subfills are tested with a two-level check. A candidate subfill passes the first-level test if its signature of length l matches at least one position in the ciphertext. The second-level test checks if l' additional bits of the signature also match any of these detected positions. Assuming the high-order bits of the keystream are independent and uniformly distributed, the expected number of subfills passing the first check is 2^{40-l}. Therefore, the second-level check is executed 2^{40-l} times on average. Similarly, the expected number of subfills passing both checks is $2^{40-l-l'}$. Thus, with appropriate choice of l and l', the false alarm rate is negligible and the checking does not affect the dominant term of the running time.

In our implementation, we select $n = 13$, $l = 16$, and $l' = 52$, for which our ciphertext-only attack runs in $2^{40-n} = 2^{27}$ steps on average and uses $1 \cdot 2^l + 4 \cdot 2^{29-n} \approx 2^{18}$ bytes of memory.

Although our implementation exploits the leaked high-order bit of each keystream byte, a related associative-memory lookup technique could be implemented without these leaked bits. In this alternate technique, each position signature σ_j of the ciphertext would consist of, say, $l = 4$ consecutive bytes of ciphertext beginning at the specified position. An associative memory would support the following operation: given any sequence κ of l consecutive candidate keystream bytes, the memory would return each ciphertext position j whose signature σ_j had the property that $\sigma_j \oplus \kappa$ appeared to be valid plaintext.

7.4 A Correlation Attack

Another promising method is to apply a generalization of Siegenthaler's [41] correlation attack, which exploits a statistical weakness of the output function to search over all possible initial subfills for one or more of the subregisters. In comparison with our time-space tradeoff, this attack more cleanly decouples the search of subregister $\mathcal{R}_3$ from that of subregisters $\mathcal{R}_1$ and $\mathcal{R}_2$. In addition, it works under very mild assumptions about the nonuniformity of the plaintext stream. In our companion paper [5], we describe such an attack in detail and derive a formula for the amount of ciphertext required.

We explain this attack applied to subregister $\mathcal{R}_i$, for any $1 \le i \le 3$. Let n_i be the length of $\mathcal{R}_i$. For simplicity, we describe this method for a known-plaintext attack. The ciphertext-only version works in essentially the same way, replacing each keystream byte with the corresponding ciphertext byte.

Let $x_0 \in \mathbb{Z}_2^{n_i}$ be any candidate subfill for $\mathcal{R}_i$ to be checked. For each $t \in \mathbb{N}$, let $x_t = R_i^t x_0$ be the tth state of $\mathcal{R}_i$ generated by subfill x_0. In addition, let N denote the number of known plaintext bytes; let $\kappa = \{k_t\}_{t=0}^{N-1}$ be the known keystream; and let $\xi = \{x_t\}_{t=0}^{N-1}$ be the candidate state sequence for $\mathcal{R}_i$. We view x_t and k_t, respectively, as the values of some random variables X_t and K_t.

The correlation attack requires a statistic $\psi : (\mathbb{Z}_2^{n_i})^N \times \mathcal{K}^N \to \mathbb{R}$ by which each candidate subfill $x_0 \in \mathbb{Z}_2^{n_i}$ can be checked. This statistic tests the random variables X_t and K_t for independence by examining their sampled values ξ and κ. Whereas Siegenthaler [41] considered only the special case $|k_t| = 1 = |x_t|$, we have the more general case $5 \le |x_t| \le 29$ and $|k_t| = 8$.

To date, we have experimented with two statistics. First, for $\mathcal{R}_3$, we tried a statistic based on the Hamming distance $\mathcal{H}(\tilde{k}_t, k_t)$ between each keystream byte k_t and an approximate candidate keystream byte $\tilde{k}_t = h(P^{-1}(0,0,0,x_t)$. Specifically, we tried the statistic $\psi(\xi, \kappa) = \sum_{t=0}^{N} \mathcal{H}(\tilde{k}_t, k_t)$.

Our hope was that, if x_0 is the correct initial subfill of $\mathcal{R}_i$, then more than half of the bits of $\{\tilde{k}_t\}_{t=0}^{N-1}$ would be correct. Unfortunately, the high density of ones in our similarity matrix P rendered this function unusable.

Next, bypassing the matrix P, we tested the sequences ξ and κ for independence (of corresponding elements) by computing a χ-squared statistic on the frequency counts of keystream bytes corresponding to 8-bit projections of the states x_t of subregister $\mathcal{R}_3$. Specifically, we computed χ-squared of the χ-squared of these 256 frequency counts. Preliminary experiments showed that this statistic also did not work, even when computed separately for each of eight bits of each keybyte. Although our ψ and χ-squared statistics did not work, we still believe that suitable statistics exist.

7.5 Hellman's Time-Space Tradeoff

Another method is to apply Hellman's [21] time-space tradeoff to reduce the on-line time to search the space of 2^{40} subfills. This method requires the ciphertext to include the encryption of some short chosen sequence of plaintext bytes. We suggest using the seven-character string ".⊔⊔The⊔", where ⊔ denotes the blank character. This string appeared in every news report that we examined. We analyze the performance of this method in our companion paper [5]. A significant advantage of this attack is that it requires only a very small amount of ciphertext (approximately one dozen bytes of ASCII-encoded English).

This algorithm attacks the effective keyspace $\mathbb{Z}_2^{40}$ of subfills for subregisters $\mathcal{R}_1$ through $\mathcal{R}_3$ by precomputing the cycle structure of a one-way function $\phi : \mathbb{Z}_2^{40} \to \mathbb{Z}_2^{40}$. For example, this attack can be implemented using the function $\phi(d) = (K_0, K_1, \ldots, K_4)$ defined as follows: whenever $d = (0, d_1, d_2, d_3) \in \mathbb{Z}_2^{40}$ is a vector of subfills, for each $0 \leq t \leq 4$, let $K_t = h(P^{-1}R^t d)$ be the tth keystream byte generated from d. This attack requires a precomputation of 2^{40} steps; the on-line computation can be arranged to run in less than 2^{29} steps.

7.6 Experimental Results

We implemented our time-space tradeoff attack on a loosely-coupled network of eight Sparcstations. On average, it takes approximately four hours to recover an initial fill from ciphertext alone. Including our library of linear algebra operations over $GF(2)$, our cryptanalytic engine comprises approximately 2,500 lines of C code.

Control Parameters: Integers n, l, and l'.
The attack uses 2^{29-n} bytes of space and requires approximately $N = 2^n$ bytes of ciphertext.
Parameters l and l' control both the collision and false-positive error rates.
(We use $n = 13$, $l = 16$, and $l' = 52$, for which $N = 2^n = 2^{13} = 8,192$.)

Precomputation: A table T_3 of length $L = \lceil (2^{29} - 1)/N \rceil$ used only to optimize code.
This table contains every Nth state of the length $2^{29} - 1$ cycle of states for subregister $\mathcal{R}_3$.
(We use $= L = 2^{29-n} = 2^{29-13} = 2^{16} = 65,536$.)

Input: A sequence $\mathcal{C}$ of $N + l'$ consecutive ciphertext bytes.

Output: An initial fill $s_0 \in \mathcal{S}$ that decrypts $\mathcal{C}$ in the sense that
the high-order bits of the keybyte stream $G^*(s_0)$ match those of $\mathcal{C}$.

Begin

$\mathcal{C}_{bits} \leftarrow$ HighOrderBits($\mathcal{C}$)
$\mathcal{H} \leftarrow$ BuildHashTable($\mathcal{C}_{bits}, l$)
%% Search over initial subfills d for registers $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$.
$L \leftarrow \lceil (2^{29} - 1)/N \rceil$
for $i \leftarrow 0$ to $L - 1$
    $d_3 \leftarrow T_3[i]$
    for each $(d_1, d_2) \in \mathbb{Z}_2^5 \times \mathbb{Z}_2^6$
        $d \leftarrow (0, d_1, d_2, d_3)$
        %% Check d (level 1).
        $\sigma \leftarrow$ HighOrderBits($G^*(P^{-1}d), l$)
        PositionList $\leftarrow \mathcal{H}[\sigma]$
        for each $j \in$ PositionList
            %% Check (d, j) (level 2).
            $\sigma' \leftarrow$ HighOrderBits($G^*(P^{-1}d), l + l'$))
            if Match($\sigma', \mathcal{C}_{bits}, j, l, l'$)
                $s_0 \leftarrow$ FindInitialFill($\mathcal{C}, d, i, j$)
                return(s_0)

End

Figure 5: Pseudocode for our ciphertext-only attack on Gifford's cipher. The attack works on the high-order bit of each ciphertext byte, which bit equals the high-order bit from the corresponding keystream byte. These bits are arranged in a hash table $\mathcal{H}$ of size 2^l. For each bit string σ of length l, hash table entry $\mathcal{H}[\sigma]$ is a linked-list of all byte positions j in the ciphertext $\mathcal{C}$ that match σ in the following sense: σ equals the sequence of high-order bits of the l consecutive bytes of $\mathcal{C}$ beginning at position j. Given any integer l and any vector of subfills $d = (0, d_1, d_2, d_3)$, the candidate vector d is checked by computing the high-order bits of the first l bytes of the candidate keystream $G^*(P^{-1}d)$, where P is a similarity matrix from F to R. If σ matches some position in the ciphertext, then a similar second-level test is performed using additional candidate keystream bytes. The function FindInitialFill computes the initial fill s_0 corresponding to the subfill vector d. To speed up the code, every Nth state in the cycle for subregister $\mathcal{R}_3$ is precomputed in a table T_3. Since the expected length of each position list is one, the on-line running time is dominated by the outer two loops, which require 2^{40-n} steps. The precomputation takes only 2^{29-n} steps. As for space, the hash table has 2^l slots (each of approximately $\lceil 2^{n-l} \rceil$ items), and the precomputed table T_3 requires 2^{29-n} entries (each of four bytes).

8 Discussion

In this section we analyze two variations of Gifford's cipher which use a modified feedback function. In addition, we present some open problems motivated by our work.

8.1 Sticky versus Non-Sticky Bit-Shifting

We compute the maximum period of a variation of Gifford's cipher for which, in the computation of the feedback function, the sticky right-shift of byte B_1 is replaced with a zero-fill right shift. This calculation is instructive because, originally, Gifford left to the compiler the decision whether to use a zero-fill right-shift versus a sticky right-shift.

If the sticky shift of byte B_1 were replaced by a zero-fill shift, the feedback matrix would differ from F in exactly one bit: bit $b_{0,8}$ would be zero rather than one (see Appendix B and Equation 13 of Section 5.1). Let F' denote this modified matrix. Using methods described in Section 5.2, we computed the characteristic polynomial of F' to be

$$\begin{aligned} p_{F'}(x) &= x^{64} + x^{56} + x^{54} + x^{52} + x^{50} + x^{48} + x^{44} + x^{40} + x^{24} && (24) \\ &= [x^{12}(x^2+x+1)(x^3+x+1)(x^6+x^3+1)(x^9+x^8+x^7+x^5+x^4+x^3+1)]^2. && (25) \end{aligned}$$

The maximum period of the state sequences generated by this variation of Gifford's cipher is the exponent of $p_{F'}$, which we shall now compute. The only non-primitive irreducible factor in $p_{F'}(x)$ is x^6+x^3+1, whose exponent is 9. As for the other polynomials, $\exp(x^2+x+1) = 3$, $\exp(x^3+x+1) = 7$ and $\exp(x^9+x^8+x^7+x^5+x^4+x^3+1) = 511$. By Theorem 2, $\exp(p_{F'}) = 2 \cdot \mathrm{lcm}(3, 7, 9, 511) = 9,198$. Remarkably, changing one bit in the feedback matrix reduces the maximum period of Gifford's cipher from 349,502,963,061 states to only 9,198 states. Intuitively, it makes sense that this variation has a smaller maximum period because the removal of bit $b_{0,8}$ from matrix F simplifies the matrix: with this modification, block F_1 of F corresponds to a nilpotent transformation (see Equation 13).

This analysis explains a phenomenon observed by Gifford. When Gifford compiled his original source code on a new compiler, he noticed that the resulting machine code produced short cycles. He deduced that the short cycles were caused by the fact that the new compiler used a zero-fill right-shift rather than a sticky right-shift. Although Gifford was unable to explain how the zero-fill shifting caused the problem, he resolved the problem by modifying the source code to force sticky right-shifting (see Appendix A).

8.2 Byte-Shifting Only

We now compute the maximum period of a second variation of Gifford's cipher in which neither byte B_1 nor byte B_7 is shifted in the computation of the feedback function. There are two reasons for studying this greatly simplified variation of Gifford's cipher. First, this variation might be useful in attacking the cipher if the shifting of bytes B_1 and B_7 could be analyzed as an embellishment of this variation. Second, regarding his choice for f to depend solely on bytes B_0, B_1, and B_7, Gifford [15, p. 465] explained that "the tap positions were chosen to yield the longest period that could be obtained" if the new byte were computed as $B_0 \oplus B_1 \oplus B_7$. We prove that Gifford's choice of taps does not achieve this objective.

Without any bit-shifting, the feedback function would be linear over $GF(2^8)$, and its characteristic polynomial (acting on bytes) would be

$$g(x) = x^8 + x^7 + x^6 + 1 = (x+1)(x^7 + x^5 + x^4 + x^3 + x^2 + x + 1). \tag{26}$$

Since $\exp(x^7 + x^5 + x^4 + x^3 + x^2 + x + 1) = 127$, the longest state sequence produced by this simplified feedback function would be only 127 states (not 255 states).

8.3 Open Problems

We now state several open questions raised by our work. These questions include questions about Gifford's cipher and more general questions about some of the algorithmic tasks required by our attack.

1. Find statistics that work well in the correlation attack described in Section 7.4. More generally, what are efficient methods for finding effective statistics for this attack?
2. Are there faster attacks on Gifford's cipher than those mentioned in Section 7? For example, is it possible to exploit the sparsity of F? How can the algebraic properties discussed in Appendix E be used to advantage? Can linear approximations to h be exploited? Is it helpful to analyze f as a nonlinear function over $GF(2^8)$? And is it useful to work with the eight Boolean equations that describe the output of h?
3. For each initial fill, what is the period of the keystream generated by Gifford's cipher, as opposed to the period of the state sequence?
4. What is the complexity of computing the rational canonical form of a binary matrix?
5. What is the complexity of the similarity transform problem, as defined in Section 5.5?
6. What other types of algebraic decompositions are useful in cryptanalysis?

9 Conclusion

We have concretely demonstrated one effective method for breaking Gifford's cipher: given ciphertext from one news article, within approximately four hours on average, our implementation recovers the secret article key and corresponding master key used for one month. Thus, Gifford's cipher is not suitable for its intended use in broadcast encryption. Moreover, our work introduces a new powerful attack on filter generators and provides an instructive detailed example of how to apply linear algebra over $GF(2)$ in cryptanalysis.

Although we demonstrate how to break Gifford's cipher, the cipher did provide some degree of protection in the BCIS. It requires a significant amount of expertise, determination, and effort to design and implement a successful break of Gifford's cipher. For most people, this effort would not be worth value of being able to read news articles; moreover, reading news articles in this fashion would be illegal. Once someone has demonstrated how to break the cipher, however, it becomes much easier for others to implement their own breaks, or to run software from the initial break. To Gifford's credit, no one implemented a break of his cipher during the period 1984–1988 of its experimental use. Nevertheless, it is important for system engineers to realize that Gifford's cipher provides no security against a determined adversary, and hence the cipher should not be used to protect valuable data.

Filter generator stream ciphers are an interesting class of ciphers. They are fast; they are easy to implement in hardware and software; and they can be easily designed to have long periods. As we show, however, neither moderately long keys, long periods, nor nonlinear output functions guarantee high security. Furthermore, details of the construction of filter generators critically affect their security. In Section 8.1, we dramatically illustrate this last point by proving that zero-fill shifting (versus sticky shifting) of byte B_1 reduces the period of Gifford's cipher from over 3.4×10^{11} states to only 9,198 states, even though this choice affects only one bit in the feedback matrix.

To design strong ciphers, it is necessary to have a deep understanding of the consequences of all design decisions. In particular, when designing filter generators, it is dangerous to choose the register length, tap positions, feedback function, and output function in a careless fashion or to leave even one bit of these decisions to the arbitrary action of a compiler. Gifford's choice of taps effectively wastes 24 out of 64 bits of the key and allows the remaining 40 bits to be separately attacked in three segments of size 5, 6, and 29 bits, respectively.

At a minimum, it would be desirable to use a longer register and to use more carefully chosen tap positions than did Gifford. In addition, it would be helpful to add more complexity to the encryption process. For example, to this end, some cryptographers incorporate nonlinear feedback functions into their designs [25]. But these simple modifications do not guarantee security.

Weaknesses in the management of cryptographic keys in the BCIS demonstrate the need for a comprehensive plan for data security that extends beyond the cipher; they also demonstrate the human difficulty of meticulously carrying such plans in real systems.

Our work illustrates the familiar historical theme: what appears to be an intractable cryptanalytic problem can be computationally feasible when attacked with appropriate mathematical machinery. More concretely, our work illustrates that algebraic decompositions are powerful tools in cryptanalysis.

Acknowledgments

We thank Chuck Fiduccia, Samuel Lomonaco Jr., Dave Saunders, and A. Brooke Stephens for discussions on the similarity transform problem, and we thank Robert W. Baldwin, Xuejia Lai, Rainer A. Rueppel, and Richard Stein for editorial comments. All computer work was carried out on workstations at the University of Maryland Baltimore County.

Appendix A: Source Code for Gifford's Cipher

```
/* Copyright 1985 Massachussetts Institute of Technology        */
/* Note:  this module makes kludgey use of unions - it will
   not work under Lattice C unless compiled -a                  */

#include "libdefs.h"
#include <stdio.h>
#include "util.h"
#include "rcv.h"
#include "key.h"

#define DEBUG           FALSE

#define NTAP1           0
#define NTAP2           1
#define NTAP3           7


/* decrypt a string of length n starting at position s with keys mk XOR ak */
global null decrypt(s, n, mk, ak)
  char  *s;     /* start of string to be decrypted */
  long  n;      /* length of string to be decrypted */
  char  *mk;    /* master key */
  char  *ak;    /* article key */
{
  int   i;              /* loop counter to initialize key */
  char  SREG[8];        /* 8-byte shift register */
  int   keycount;       /* pointer into shift register in lieu of rotation */
  char  temp;           /* holds new byte while shifting shift register */

  union twobytes {
        char    bytes[2];
        int     both;
  } arg1, arg2;

  /* initialize key */
  for( i=0; i<8; i++ ) SREG[i] = ak[i] ^ mk[i];
  keycount = 0;

  while( n-- > 0) {
        /* generate a new byte */
        temp = SREG[ (keycount+NTAP1) & 07]
                ^ (SREG[ (keycount+NTAP2) & 07]) >> 1
                ^ ((SREG[ (keycount+NTAP3)& 07]) << 1) & 0377;
        /* This is here purely to conform to the other C compiler */
        /* and sign extension of right shifted variables        */
        if   ((SREG[ (keycount+NTAP2) & 07]) & 0x80 )
                temp ^= 0x80;
        /* shift the shift register and put in the new byte */
        keycount = (--keycount) & 07;
        SREG[keycount] = temp;

        /* XOR the new byte with the current char of text and advance;
           this is where we need to insert the nonlinear function. */

        arg1.bytes[0] = SREG[ (keycount+0) & 07];
        arg1.bytes[1] = SREG[ (keycount+2) & 07];
        arg2.bytes[0] = SREG[ (keycount+4) & 07];
        arg2.bytes[1] = SREG[ (keycount+7) & 07];
        temp = (arg1.both * arg2.both) >> 8;

        *(s++) ^= temp;
  }
}
```

Figure 6: C language source code for Gifford's cipher obtained from Stephen Berlin, while he worked on the Boston Community Information System.

Appendix B: Matrices F, R, P, P^{-1}

```
     0............1............2............3............4............5............6............7............

0    1000000010000000000000000000000000000000000000000000000001000000
.    0100000001000000000000000000000000000000000000000000000000100000
.    0010000000100000000000000000000000000000000000000000000000010000
.    0001000000010000000000000000000000000000000000000000000000001000
.    0000100000001000000000000000000000000000000000000000000000000100
.    0000010000000100000000000000000000000000000000000000000000000010
.    0000001000000010000000000000000000000000000000000000000000000001
.    0000000100000001000000000000000000000000000000000000000000000000
1    1000000000000000000000000000000000000000000000000000000000000000
.    0100000000000000000000000000000000000000000000000000000000000000
.    0010000000000000000000000000000000000000000000000000000000000000
.    0001000000000000000000000000000000000000000000000000000000000000
.    0000100000000000000000000000000000000000000000000000000000000000
.    0000010000000000000000000000000000000000000000000000000000000000
.    0000001000000000000000000000000000000000000000000000000000000000
.    0000000100000000000000000000000000000000000000000000000000000000
2    0000000010000000000000000000000000000000000000000000000000000000
.    0000000001000000000000000000000000000000000000000000000000000000
.    0000000000100000000000000000000000000000000000000000000000000000
.    0000000000010000000000000000000000000000000000000000000000000000
.    0000000000001000000000000000000000000000000000000000000000000000
.    0000000000000100000000000000000000000000000000000000000000000000
.    0000000000000010000000000000000000000000000000000000000000000000
.    0000000000000001000000000000000000000000000000000000000000000000
3    0000000000000000100000000000000000000000000000000000000000000000
.    0000000000000000010000000000000000000000000000000000000000000000
.    0000000000000000001000000000000000000000000000000000000000000000
.    0000000000000000000100000000000000000000000000000000000000000000
.    0000000000000000000010000000000000000000000000000000000000000000
.    0000000000000000000001000000000000000000000000000000000000000000
.    0000000000000000000000100000000000000000000000000000000000000000
.    0000000000000000000000010000000000000000000000000000000000000000
4    0000000000000000000000001000000000000000000000000000000000000000
.    0000000000000000000000000100000000000000000000000000000000000000
.    0000000000000000000000000010000000000000000000000000000000000000
.    0000000000000000000000000001000000000000000000000000000000000000
.    0000000000000000000000000000100000000000000000000000000000000000
.    0000000000000000000000000000010000000000000000000000000000000000
.    0000000000000000000000000000001000000000000000000000000000000000
.    0000000000000000000000000000000100000000000000000000000000000000
5    0000000000000000000000000000000010000000000000000000000000000000
.    0000000000000000000000000000000001000000000000000000000000000000
.    0000000000000000000000000000000000100000000000000000000000000000
.    0000000000000000000000000000000000010000000000000000000000000000
.    0000000000000000000000000000000000001000000000000000000000000000
.    0000000000000000000000000000000000000100000000000000000000000000
.    0000000000000000000000000000000000000010000000000000000000000000
.    0000000000000000000000000000000000000001000000000000000000000000
6    0000000000000000000000000000000000000000100000000000000000000000
.    0000000000000000000000000000000000000000010000000000000000000000
.    0000000000000000000000000000000000000000001000000000000000000000
.    0000000000000000000000000000000000000000000100000000000000000000
.    0000000000000000000000000000000000000000000010000000000000000000
.    0000000000000000000000000000000000000000000001000000000000000000
.    0000000000000000000000000000000000000000000000100000000000000000
.    0000000000000000000000000000000000000000000000010000000000000000
7    0000000000000000000000000000000000000000000000001000000000000000
.    0000000000000000000000000000000000000000000000000100000000000000
.    0000000000000000000000000000000000000000000000000010000000000000
.    0000000000000000000000000000000000000000000000000001000000000000
.    0000000000000000000000000000000000000000000000000000100000000000
.    0000000000000000000000000000000000000000000000000000010000000000
.    0000000000000000000000000000000000000000000000000000001000000000
.    0000000000000000000000000000000000000000000000000000000100000000
```

Figure 7: The 64×64 feedback matrix F. The first eight rows define the feedback byte, which is computed from shift register bytes B_0, B_1, and B_7. The large identity block describes the byte-shifting of the shift register. Because column 56 is zero, F is not invertible. If zero-fill right bit-shifting were used instead of sticky shifting, exactly one bit would change: the 8th bit of row zero would change from one to zero. Every eighth row and column is numbered.

```
     0...........1...........2...........3...........4...........5...........6...........7...........

0    0000000000000000000000000000000000000000000000000000000000000000
.    1000000000000000000000000000000000000000000000000000000000000000
.    0100000000000000000000000000000000000000000000000000000000000000
.    0010000000000000000000000000000000000000000000000000000000000000
.    0001000000000000000000000000000000000000000000000000000000000000
.    0000100000000000000000000000000000000000000000000000000000000000
.    0000010000000000000000000000000000000000000000000000000000000000
.    0000001000000000000000000000000000000000000000000000000000000000
1    0000000100000000000000000000000000000000000000000000000000000000
.    0000000010000000000000000000000000000000000000000000000000000000
.    0000000001000000000000000000000000000000000000000000000000000000
.    0000000000100000000000000000000000000000000000000000000000000000
.    0000000000010000000000000000000000000000000000000000000000000000
.    0000000000001000000000000000000000000000000000000000000000000000
.    0000000000000100000000000000000000000000000000000000000000000000
.    0000000000000010000000000000000000000000000000000000000000000000
2    0000000000000001000000000000000000000000000000000000000000000000
.    0000000000000000100000000000000000000000000000000000000000000000
.    0000000000000000010000000000000000000000000000000000000000000000
.    0000000000000000001000000000000000000000000000000000000000000000
.    0000000000000000000100000000000000000000000000000000000000000000
.    0000000000000000000010000000000000000000000000000000000000000000
.    0000000000000000000001000000000000000000000000000000000000000000
.    0000000000000000000000100000000000000000000000000000000000000000
3    0000000000000000000000000000100000000000000000000000000000000000
.    0000000000000000000000001000100000000000000000000000000000000000
.    0000000000000000000000000100100000000000000000000000000000000000
.    0000000000000000000000000010100000000000000000000000000000000000
.    0000000000000000000000000001000000000000000000000000000000000000
.    0000000000000000000000000000000000100000000000000000000000000000
.    0000000000000000000000000000010000000000000000000000000000000000
.    0000000000000000000000000000001000100000000000000000000000000000
4    0000000000000000000000000000000100000000000000000000000000000000
.    0000000000000000000000000000000010100000000000000000000000000000
.    0000000000000000000000000000000001100000000000000000000000000000
.    0000000000000000000000000000000000000000000000000000000000000001
.    0000000000000000000000000000000000010000000000000000000000000001
.    0000000000000000000000000000000000001000000000000000000000000001
.    0000000000000000000000000000000000000100000000000000000000000001
.    0000000000000000000000000000000000000010000000000000000000000001
5    0000000000000000000000000000000000000001000000000000000000000001
.    0000000000000000000000000000000000000000100000000000000000000000
.    0000000000000000000000000000000000000000010000000000000000000001
.    0000000000000000000000000000000000000000001000000000000000000000
.    0000000000000000000000000000000000000000000100000000000000000001
.    0000000000000000000000000000000000000000000010000000000000000001
.    0000000000000000000000000000000000000000000001000000000000000000
.    0000000000000000000000000000000000000000000000100000000000000000
6    0000000000000000000000000000000000000000000000010000000000000001
.    0000000000000000000000000000000000000000000000001000000000000001
.    0000000000000000000000000000000000000000000000000100000000000000
.    0000000000000000000000000000000000000000000000000010000000000001
.    0000000000000000000000000000000000000000000000000001000000000000
.    0000000000000000000000000000000000000000000000000000100000000001
.    0000000000000000000000000000000000000000000000000000010000000001
.    0000000000000000000000000000000000000000000000000000001000000001
7    0000000000000000000000000000000000000000000000000000000100000000
.    0000000000000000000000000000000000000000000000000000000010000001
.    0000000000000000000000000000000000000000000000000000000001000000
.    0000000000000000000000000000000000000000000000000000000000100000
.    0000000000000000000000000000000000000000000000000000000000010000
.    0000000000000000000000000000000000000000000000000000000000001001
.    0000000000000000000000000000000000000000000000000000000000000100
.    0000000000000000000000000000000000000000000000000000000000000011
```

Figure 8: The matrix R, the primary rational canonical form of the feedback matrix F. Each block of this block-diagonal matrix is a a companion matrix, whose last column consists of the coefficients of an elementary divisor of F. The four blocks have dimensions 24, 5, 6, and 29, respectively; they decompose the state space into four F-cyclic subspaces. Every eighth row and column is numbered.

	0...........1...........2...........3...........4...........5...........6...........7...........
0	0000000100000000100000001000000000000000000000000000000000000000
.	0000000100000000000000001100000010000000000000000000000000000000
.	0000000100000000000000001000000011000000100000000000000000000000
.	0000000100000000000000001000000010000000110000001000000000000000
.	0000000100000000000000001000000010000000100000001100000010000000
.	0000000100000000000000001000000010000000100000001000000011000010
.	0000010000000010100001000000000010000000100000001000000010000011
.	0000011100000001100001011000010000000000100000001000000010000010
1	0000010100000001000001001000010110000100000000001000000010000010
.	0000010000000001000001010000001001000010110000100000000010000010
.	0000010000000000000000100100001010000010010000101100001000000010
.	0000010100000001000010100000100100001010000010010000101100001000
.	0001000000010101001000010000101000000100100001010000001001000010 11
.	0001011000000110001110010010000100001010000001001000010100000010 01
.	0001001000000010000100010001110010010000100001010000001001000010 10
.	0001010000000110001011000010001000111001001000010000101000000100 1
2	0001001100000111001000000001011000010001000111001001000010000101 0
.	0001010000000111001011110010000000101100001000100011100100100001
.	0100001101010111100000001001011110010000000101100001000100011100 1
.	0111001100110000101100011000000100101111001000000010110000100010
.	0100010000110111101110001011000110000001001011110010000000101100
.	0101100100011101100001011011100010110001100000010010111100100000
.	0100000000011001100111011000010110111000101100011000000100101111
.	0101111000011110101001011001110110000101101110001011000110000001
3	0100100011100001000000110100011001101101000111100001000100110110
.	0010010011000101001110100100010100101011011100110000111100100111
.	0000011110001010010110000111110000101000001101010110001000111001
.	0011101010010100011011010001111000010001001101100010010001010100
.	1010100110010011010001100110110100011110000100010011011000100100
.	1000001001011101011101010110101101111110010010000100011100001101
.	0001101010011000011010010111010101101011011111100100100001000111
.	0000110011001001011100010000000100000101100100011001110010100010 1
4	1000101010000110010111010111000100000001000000101100100011001110 01
.	1111000110100100000101010011011000000111101001010010011000010111 0
.	1101111111110001011010110111111001001000010001110000110101000001
.	0100001011010001001101100001111101110010010100010111000100110111
.	0010110011111101001110110010100101101101001000110010000000100011 0
.	1100111001110001010100110010010001011011001111000101001000010111
.	0110110000110001000110110100110001010110000010100100110101100101
.	1000100101110110000100010000001000011111000000111011111011011110 10
5	1011011010101100001010000000111001110110011011110111011001001100
.	1001100000101110000111000010100000001110011101100110111101110110
.	1010111110100100010000100000001101011010010111111000001110101100 0
.	1011000000011111010001000100001000000001101011010010111111000001 11
.	0100110001101111001101010101101100110000010100100010101101101000
.	1001001001001101011110010010101000101001011000010010001100011100
.	0011100110101011001111110111100100101010001010010110000100100011
.	0100011001111111001001110011111101111001001010100010100101100001
6	1000000101010100010011110011100001001101001010000101101100011110
.	0111111101101101000110000101000001001010000111000101100101101100
.	1101100010100111010111010001100001010000010010100001110001011001
.	1111000010111011011101010100001001101010000000001001110110010101 1
.	0101011010100110000101110111010101000010011010100000000010011101 1
.	0011010011110001011111000000010000000011100010011000110110011011 0
.	0010111010001001000111110110001101111010010101100110001000101100
.	0001101010100111000011110000000000001000100101011001001110101010 1
7	1010101110110001011100010000111100000000000010001001010110010011 1
.	0000110100110101000110010110111001111101010100010110000000011100
.	0011100100110100010001110001100101101110011111010101000101100000
.	1100000011111001001101000100011100011001011011100111110101010001
.	1010001101100011001111110011010001000111000110010110111001111101
.	1011100110001001001000110010000000100011000010110011010000101100 1
.	1011001000000101101101101001000110010000000100011000010110011010 00
.	1001001110110010000111110111001001010001011100010011011100100001

Figure 9: An invertible similarity matrix P, which satisfies the equation $F = P^{-1}RP$. Note that P is dense and that P does not have block structure. Every eighth row and column is numbered.

```
  0...........1...........2...........3............4...........5...........6...........7...........

0 0110111110100001100000000010101000101101000110001101101001101111
. 1001100100100000000000000101100110111001010000001110000111010110 1
. 0101100001100000000000000110000100000011100010100111000001011010 1
. 0001100000000000000000000111111011011111011100000000000100100100 11
. 0001100000000000000000000001100101110101001111101111011110100000
. 0000000000000000000000000010000001101100110001101010101001000101 1
. 0000000000000000000000000101111111110010001000011001001101100100 0
. 0000000000000000000000000101101101010000111100000100011101101110 0
  ------------------------------------------------------------------
1 1011011111010000110000000001000100010110100011000110110100110111
. 1100110010010000000000000010111011010100101000000111000011101011 0
. 0010110000110000000000000011001010000001110001010011100000101101 0
. 0000110000000000000000000011110101101111011100000000000010010010 01
. 0000110000000000000000000010001000101010101001111101111011110100 00
. 0000000000000000000000000010100000011011001100011010101010010001 01
. 0000000000000000000000000011011011111000100010000110010011011001 00
. 0000000000000000000000000010111110100000011110000010001110110111 0
  ------------------------------------------------------------------
2 0101101111101000011000000000100010011011010001100011011010011011
. 1110011001001000000000000001010101101010010100000011100001110101 1
. 1001011000011000000000000001100101000000111000101001110000010110 1
. 1000011000000000000000000010111101011011111011100000000000100100 100
. 1000011000000000000000000010000000101010101001111101111011110100 0
. 1000000000000000000000000010100000001001100110001101010101001000 10
. 1000000000000000000000000011101001111000010001000011001001101100 10
. 0000000000000000000000000000101011101000000111100000100011101101 11
  ------------------------------------------------------------------
3 0010110111111010000110000100001000100110110100011000110110100110 1
. 0111001100100100000000000000101010111101001010000001110000111010 1
. 1100101100001100000000000100110010101000011100010100111000001011 0
. 0100001100000000000000000110110101010011111011100000000000100100 10
. 0100001100000000000000000101001000010101010100111110111101111010 0
. 0100000000000000000000000101011000000000110011000110101010100100 01
. 1100000000000000000000000111010011100000100010000110010011011001
. 0000000000000000000000000000101011100000000111100000100011101101 1
  ------------------------------------------------------------------
4 0001011011111010000110001100011000100110110100011000110110100110
. 0011100110010010000000000000010101010110100101000000111000011101 0
. 1110010110000110000000000010011001010100001110001010011100000101 1
. 0010000110000000000000000111011010100001111101110000000000010010 01
. 0010000110000000000000000101011000000101010100111110111101111010
. 0010000000000000000000000110100100000000110011000110101010100100 0
. 0110000000000000000000000000111110011000000100010000110010011011 00
. 0000000000000000000000000000010101110000000001111000001000111011 01
  ------------------------------------------------------------------
5 0000101101111101000011000110001100000011011010001100011011010011
. 0001110011001001000000000100001010100011010010100000011100001110 1
. 0111001011000011000000000001000100100010000111000101001110000010 1
. 1001000011000000000000000111011010100001111101110000000000010010 0
. 0001000011000000000000000101011110000001010101001111101111011111 01
. 1001000000000000000000000110110100000000110011000110101010100100
. 0011000000000000000000000001111100100000001000100001100100110110
. 0000000000000000000000000100000010110000000001111000001000111011 0
  ------------------------------------------------------------------
6 0000010110111110100001100011010110000001101101000110001101101001
. 0000111001100100100000000110001101010001101001010000001110000111 0
. 0011100101100001100000000100100010010001000011100010100111000001 0
. 0100100001100000000000000001111110100000011111011100000000000010 010
. 0000100001100000000000000110101111000000101010100111110111101111 10
. 0100100000000000000000000101100101000000000110011000110101010100 10
. 0001100000000000000000000100010111000000000010001000011001001101 1
. 0000000000000000000000000110000001010000000000011110000001000111 011
  ------------------------------------------------------------------
7 0000001011011111010000111001111011000000110110100011000110110100
. 0000011100110010010000000110011101000001101001010000001110000111
. 0001110010110000110000000110011001000000010000111000101001110000 01
. 0010010000110000000000000001101110100000011111011100000000000010 01
. 0000010000110000000000000011010111100000010101010011111011110111 1
. 0010010000000000000000000001011001010000000001100110001101010101 001
. 0000110000000000000000000010000011100000000000100010000110010011 01
. 0000000000000000000000000011000000100000000000001111000001000111 01
```

Figure 10: The matrix P^{-1}, the inverse of P. For each block R_i of R of dimension n_i, the associated columns of this P are powers $F^j v$ $(0 \leq j \leq n_i - 1)$ of some F-cyclic vector v of the cyclic subspace corresponding to R_i. Every eighth row and column is numbered.

Appendix C: An Example of Plaintext with Matching Ciphertext

Table 2: An example of plaintext with matching ciphertext produced by our implementation of Gifford's cipher. The plaintext is a portion of a *Washington Post* news article by Ken Ringle, published on January 24, 1994. This article is typical of news reports broadcast by the BCIS. Every plaintext byte is encrypted, including bytes representing special characters. We denote the blank character by ␣, and we denote the newline character by ▯. The ciphertext is arranged in 4-byte blocks, where each byte is represented by two hexadecimal numerals. When run on the complete 10,072 bytes of ciphertext from this article, our ciphertext-only attack determined the initial fill 48dab075 67588bc7$_{16}$ within four hours and thirty minutes. Note that Line 20 contains the common string "`.␣␣The␣`", which appeared in every news article we examined.

Line	Plaintext	Ciphertext				
1	`Only␣Sleuths␣Can␣Fin`	3e5e3fd0	d5c8e006	db843fdb	f001e307	c8943117
2	`d␣This␣Museum▯▯By␣Ke`	cc577677	e633ca05	13504da1	0f5c48e2	82ab9d2d
3	`n␣Ringle▯Washington␣`	048506e5	2ce22154	622e5c78	c397ac38	365eaf93
4	`Post␣Staff␣Writer▯▯I`	65d04b43	5afeb169	e87a06fb	5f0e54ca	ce4f9547
5	`n␣the␣shadowy␣world␣`	e6cfcdc1	726fa2ab	28e1d078	b35a867b	eeadcb50
6	`of␣U.S.␣intelligence`	f5bc9d1b	e4320966	40ca9a83	20cab367	941a633c
7	`␣agencies,␣the␣Natio`	632942fd	dda07610	3f17e741	221d9adb	7353959f
8	`nal␣Security▯Agency␣`	92aecd9d	d9916598	f4d92781	b22e1008	58491e52
9	`has␣always␣been␣the␣`	d9b797c4	3b0264de	50dbb9eb	f3edc044	544d2b31
10	`most␣clandestine␣of␣`	9f3d9a05	558f77d5	aa60b9d7	f2f50c1b	45123a32
11	`all.▯▯Some␣20,000␣pe`	1ce106f7	18af1fe9	f5634c64	879057a9	8c745032
12	`ople␣work␣at␣the␣mir`	7f354a5c	23941df1	7e7f7e32	8d003ca8	f59ed944
13	`ror-windowed␣complex`	2ce428bc	1d63d765	5721b0dc	aea986f1	c63b1df6
14	`␣at␣Fort␣Meade␣south`	2009745a	9dea9250	8486f771	465d7b08	cfa226af
15	`▯of␣Baltimore-Washin`	d046a7df	41f23949	a2a3fd9b	7b9e74a2	01e846a7
16	`gton␣International␣A`	bbc224e3	792b2ef1	ae01ca7e	a901265e	c75e6e15
17	`irport,␣but␣until␣19`	3436c75b	307ca3bb	6a89e20e	3f0ca858	6fbbf91c
18	`89␣there␣wasn't▯even`	d811e6b0	406b46cd	60e9f5e5	b6444f5d	00681620
19	`␣a␣sign␣in␣front␣of␣`	65bef3cf	d44c9be5	d8960879	5b1755e2	d6ab6241
20	`the␣buildings.␣␣The␣`	aba20648	a1c10b61	a7cd4893	b8ab8aaf	41d457f8
21	`1952␣executive␣order`	815eac95	8f076549	b9c12ddd	de1ee64c	e4f5ff19
22	`␣that▯created␣the␣ag`	aae6ce62	a4690961	3347c035	4ad6a437	ace85102
23	`ency␣was␣itself␣clas`	7736ce4c	e81bdb78	8fe6213b	54f8b8c8	b8eb7f8d
24	`sified.␣␣For␣years␣i`	22b2ce16	0d8b3e6f	fab5b315	c2d07f75	b82cac4b
25	`t␣was␣a␣federal▯crim`	9c52c623	ed608444	59f797f2	cc168a0b	a86c5392
26	...	...				

Appendix D: A Length 21 Cycle

Table 3: An example of a shortest period of Gifford's cipher for nonzero initial fill. The initial fill $s_0 = 3c668596\ 77e72892$ from subspace $V_0^+ \oplus V_2^+$ generates a sequence of register states with a four-state leader and a 21-state cycle. The leader is determined by subregister $\mathcal{R}_0$, and the cycle is determined by subregister $\mathcal{R}_2$. For each iteration $0 \leq t \leq 27$, we list the shift register contents $s_t = f^t(s_0)$, the keybyte $K_t = h(s_t)$, and the four subfills $Ps_t = (d_0, d_1, d_2, d_3)$ of subregisters $\mathcal{R}_0$, $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$. These subregisters correspond to a decomposition $\mathcal{S} = V_0 \oplus V_1 \oplus V_2 \oplus V_3$ of the state space into a direct sum of four F-invariant subspaces of dimensions 24, 5, 6, 29, respectively. Throughout, each register state is represented as a sequence of bytes, where each byte is represented with two hexadecimal numerals. For example, the initial state of 6-bit subregister $\mathcal{R}_2$ is $35_{16} = 0011_2\ 0101_2 = 110101_2$. Each state of the main shift register is described by two strings of eight hexadecimal numerals, where each string represents four bytes of the register.

Iteration t	Shift Register State s_t		Subregister States d_0	d_1	d_2	d_3	Keybyte K_t	Comment
0	3c668596	77e72892	00000f	00	35	00000000	56	Start leader.
1	2b3c6685	9677e728	000007	00	31	00000000	8b	
2	652b3c66	859677e7	000003	00	33	00000000	85	
3	be652b3c	66859677	000001	00	32	00000000	87	
4	62be652b	3c668596	000000	00	19	00000000	53	Enter cycle.
5	9162be65	2b3c6685	000000	00	27	00000000	a1	
6	aa9162be	652b3c66	000000	00	38	00000000	8d	
7	aeaa9162	be652b3c	000000	00	1c	00000000	87	
8	03aeaa91	62be652b	000000	00	0e	00000000	b1	
9	8203aeaa	9162be65	000000	00	07	00000000	1c	
10	498203ae	aa9162be	000000	00	28	00000000	2e	
11	f4498203	aeaa9162	000000	00	14	00000000	f5	
12	14f44982	03aeaa91	000000	00	0a	00000000	58	
13	cc14f449	8203aeaa	000000	00	05	00000000	02	
14	92cc14f4	498203ae	000000	00	29	00000000	fd	
15	2892cc14	f4498203	000000	00	3f	00000000	ea	
16	e72892cc	14f44982	000000	00	34	00000000	00	
17	f7e72892	cc14f449	000000	00	1a	00000000	5a	
18	96f7e728	92cc14f4	000000	00	0d	00000000	92	
19	8596f7e7	2892cc14	000000	00	2d	00000000	0f	
20	668596f7	e72892cc	000000	00	3d	00000000	19	
21	3c668596	f7e72892	000000	00	35	00000000	d6	
22	2b3c6685	96f7e728	000000	00	31	00000000	8b	
23	652b3c66	8596f7e7	000000	00	33	00000000	85	
24	be652b3c	668596f7	000000	00	32	00000000	9d	
25	62be652b	3c668596	000000	00	19	00000000	53	Repeat cycle.
26	9162be65	2b3c6685	000000	00	27	00000000	a1	
27	aa9162be	652b3c66	000000	00	38	00000000	8d	

Appendix E: Properties of the Nonlinear Output Function h

In our preliminary study of Gifford's cipher, we analyzed the nonlinear output function h to gain insight into its behavior and to identify properties that might be useful in cryptanalysis. First, we computationally determined the exact distribution of h. Second, we proved several algebraic properties of h. For example, Proposition 5 provides a partial inversion of h: given its output keybyte and three of its four input bytes, certain remaining input bits are uniquely determined as solutions to certain linear modular equations. This appendix presents our findings.

Throughout, we study the restriction $\hat{h} : \mathbb{Z}_{256}^4 \to \mathcal{K}$, where $\mathcal{K} = \mathbb{Z}_{256}$. But instead of working with Gifford's operational definition of $\hat{h}$ as the third byte from the product of two 16-bit integers (see Section 4), we view $\hat{h}$ as the nonlinear Diophantine equation

$$\hat{h}(B_0, B_2, B_4, B_7) = (B_2 B_4 + B_0 B_7 + \lfloor B_2 B_7 / 256 \rfloor) \bmod 256, \tag{27}$$

whenever $B_0, B_2, B_4, B_7 \in \mathbb{Z}_{256}$. Although our time-space tradeoff does not depend on any of the properties from this appendix, some of these properties (*e.g.* the partial inversion of $\hat{h}$) may facilitate other attacks.

E.1 The Distribution of $\hat{h}$

To investigate statistical properties of $\hat{h}$, we determine the preimage set $\hat{h}^{-1}(y) = \{x : \hat{h}(x) = y\}$ for each $y \in \mathcal{K}$. These preimage sets induce a partition of $\mathcal{K}$ into nine equivalence classes $W_0, W_1, \ldots, W_8$ of keystream bytes whose preimages have the same cardinality. Table 4 lists each equivalence class and the cardinality of its associated preimage set. We determined the preimage sets by computing $\hat{h}(x)$ for each $x \in \mathbb{Z}_{256}^4 \cong \mathbb{Z}_{2^{32}}$.

From these preimages sets, the distribution of the random variable $\hat{h}(X)$ can be computed for any random variable X on $\mathbb{Z}_{2^{32}}$. For example, for each $y \in \mathcal{K}$, the last column of Table 4 gives the *a priori* probability $p_y = \text{Prob}[\hat{h}(X) = y] = |\hat{h}^{-1}(y)|/2^{32}$, assuming X is uniformly distributed over $\mathbb{Z}_{2^{32}}$. For this case, $\hat{h}(X)$ has a nonuniform distribution: except for $y \in W_1$, the values of p_y deviate from the uniform probability $1/256 \approx 0.003906$. The most frequent keybyte is 0, which appears with probability $p_0 \approx 0.003967$. The least frequent keybytes are the odd keybytes, each of which appears with probability $p_1 \approx 0.003898$.

Table 4 also reveals the following structure. For each $0 \leq i \leq 8$, $W_i = \{2^i(2n+1) \bmod 256 : n \in \mathbb{N}\}$. For each $0 \leq i < 8$ and for each $y \in W_i$, $|\hat{h}^{-1}(y)| = 2^{24} + (i-1)2^{15}$. As for $W_8 = \{0\}$, $|\hat{h}^{-1}(0)| = 2^{24} + 2^{18}$. By contrast, were $\hat{h}$ uniform, each preimage set would have cardinality $2^{32}/2^8 = 2^{24}$.

The nonuniformity of the distribution of $\hat{h}$ may be helpful in finding a suitable statistic for our correlation attack. In addition, it may yield a statistical test for recognizing ciphertext generated by Gifford's cipher.

Table 4: The distribution of $\hat{h}$. There are nine equivalence classes $W_0, W_1, \ldots, W_8$ of keystream bytes whose preimages have the same cardinality. For each equivalence class, the last column lists the probability $\text{Prob}[\hat{h}(X) = y] = |\hat{h}^{-1}(y)|/2^{32}$, when the random variable X is uniformly distributed over $\mathbb{Z}_{2^{32}}$.

i	$y \in W_i$	$\lvert W_i \rvert$	$\lvert\hat{h}^{-1}(y)\rvert$	$\lvert\hat{h}^{-1}(y)\rvert/2^{32}$
0	$\{1, 3, 5, \ldots, 255\}$	128	$2^{24} - 2^{15}$	0.003898
1	$\{2, 6, 10, \ldots, 254\}$	64	2^{24}	0.003906
2	$\{4, 12, 20, \ldots, 252\}$	32	$2^{24} + 2^{15}$	0.003914
3	$\{8, 24, 40, \ldots, 248\}$	16	$2^{24} + 2 \cdot 2^{15}$	0.003921
4	$\{16, 48, 80, \ldots, 240\}$	8	$2^{24} + 3 \cdot 2^{15}$	0.003929
5	$\{32, 96, 160, 224\}$	4	$2^{24} + 4 \cdot 2^{15}$	0.003937
6	$\{64, 192\}$	2	$2^{24} + 5 \cdot 2^{15}$	0.003944
7	$\{128\}$	1	$2^{24} + 6 \cdot 2^{15}$	0.003952
8	$\{0\}$	1	$2^{24} + 8 \cdot 2^{15}$	0.003967

E.2 Algebraic Properties

From Equation 27 we prove several algebraic properties of $\hat{h}$, which may be helpful in cryptanalytic attacks that involve inverting $\hat{h}$.

E.2.1 Elementary Properties of $\hat{h}$

We first state two simple properties that arise from the operational definition of h. Proposition 3 shows how inputs to $\hat{h}$ commute, and Proposition 4 gives a sufficient condition for the output of $\hat{h}$ to be zero.

Proposition 3. *$\hat{h}(B_0, B_2, B_4, B_7) = \hat{h}(B_4, B_7, B_0, B_2)$ for all $B_0, B_2, B_4, B_7 \in \mathbb{Z}_{256}$.*

Proof. From Equation 27 and the fact that $B_2 B_4 + B_0 B_7 + \lfloor B_2 B_7/256 \rfloor = B_7 B_0 + B_4 B_2 + \lfloor B_7 B_2/256 \rfloor$. ■

Proposition 4. *If $B_0 = B_2 = 0$ or $B_4 = B_7 = 0$, then $\hat{h}(B_0, B_2, B_4, B_7) = 0$.*

Proof. If $B_0 = B_2 = 0$ or $B_4 = B_7 = 0$, then each term in $B_2 B_4 + B_0 B_7 + \lfloor B_2 B_7/256 \rfloor$ is zero. ■

E.2.2 Partial Inversion of $\hat{h}$

We now analyze how to invert $\hat{h}$ when some of its inputs are known. Our main results are Proposition 5 and its corollaries, which give conditions for inverting $\hat{h}$ when the output byte and three of the input bytes are known. These results may be useful in applying or optimizing certain attacks on Gifford's cipher. For example, our partial inversion of $\hat{h}$ may be helpful in applying linear consistency attacks [44].

For $1 \le i \le 4$, let $\hbar_i : \mathbb{Z}_{256} \to \mathbb{Z}_{256}$ be the restriction of $\hat{h}$ whose ith input is unknown and whose three other inputs are known constants. For example, $\hbar_3(x) = \hat{h}(B_0, B_2, x, B_7)$ where bytes B_0, B_2, and B_7 are known constants.

To prove Proposition 5, we solve certain linear Diophantine equations derived from Equation 27. Lemma 3 reviews some well-known theory about such equations.

Lemma 3. *Let $a, b, n \in \mathbb{Z}$; let $d = \gcd(a, n)$; and let $g(x) = ax \bmod n$. The equation $g(x) = b$ has a solution for $x \in \mathbb{Z}_n$ if and only if $d|b$. Moreover, if a solution exists, then there are exactly d solutions, and these solutions are*

$$x = \left(\frac{b}{d}x_0 + t\frac{n}{d}\right) \text{ for } t = 0, 1, \ldots, d-1, \tag{28}$$

where x_0 is any solution to $(a/d)x \bmod n/d = 1$.

Proof. See Nivan and Zuckerman [34, p. 42]. ∎

Furthermore, the solutions from Lemma 3 can be efficiently computed by extended Euclid's algorithm [7]. We now apply Lemma 3 to establish the following partial inversion of $\hat{h}$.

Proposition 5. *Let B_0, B_2, B_7 be any fixed bytes, and let $d = \gcd(B_2, 256)$. For any constant $z \in \mathbb{Z}_{256}$, the equation $\hbar_3(x) = z$ has a solution for $x \in \mathbb{Z}_{256}$ if and only if $d \mid z - ((B_0B_7 + \lfloor B_2B_7/256 \rfloor) \bmod 256)$. Moreover, if $\hbar_3(x) = z$ has a solution, then there are exactly d solutions, and each solution has the same low-order $8 - \lg d$ bits.*

Proof. By Equation 27, $x \in \mathbb{Z}_{256}$ is a solution of $\hbar_3(x) = z$ if and only if

$$z = (B_2x + B_0B_7 + \lfloor B_2B_7/256 \rfloor) \bmod 256. \tag{29}$$

Hence, x is a solution of $\hbar_3(x) = z$ if and only if

$$B_2x \equiv z - \alpha \bmod 256, \tag{30}$$

where $\alpha = (B_0B_7 + \lfloor B_2B_7/256 \rfloor) \bmod 256$.

By Lemma 3, Equation 30 has a solution if and only if $d \mid z - \alpha$. Furthermore, if Equation 30 has any solution, then it has exactly d solutions, each of the form

$$x = \left(\frac{z-\alpha}{d}x_0 + t\frac{256}{d}\right), \text{ for } t = 0, 1, \ldots, d-1, \tag{31}$$

where x_0 is any solution to $(B_2/d)x \bmod 256/d = 1$.

By Equation 31, any two solutions of $\hbar_3(x) = z$ differ only by some multiple of $256/d$. Since $256/d$ is a perfect power of 2, it follows that the low-order $\lg(256/d) = 8 - \lg d$ bits of each solution are the same. ∎

Proposition 6 characterizes the structure of the set $\text{image}(\hbar_3) = \{\hbar_3(x) : x \in \mathbb{Z}_{256}\}$.

Proposition 6. *Let B_0, B_2, B_7 be any fixed bytes, and let $d = \gcd(B_2, 256)$. It is true that $|\text{image}(\hbar_3)| = 256/d$ and that $\text{image}(\hbar_3) = \{(B_0B_7 + \lfloor B_2B_7/256 \rfloor + kd) \bmod 256 : k = 0, 1, 2, \ldots, (256/d) - 1\}$.*

Proof (Sketch). The formula for $\text{image}(\hbar_3)$ follows from Equation 29. To determine the cardinality of $\text{image}(\hbar_3)$, we also apply Proposition 3, Lagrange's Theorem, and the fact that B_2 generates an additive subgroup of $\mathbb{Z}_{256}$. ∎

Combining Propositions 3 and 5 yields analogous results for $\hbar_1$.

Corollary 2. *Propositions 5 and 6 remain true if $\hbar_3$ is replaced by $\hbar_1$, and the expression $B_0 B_7 + \lfloor B_2 B_7/256 \rfloor$ is replaced by $B_2 B_4 + \lfloor B_7 B_2/256 \rfloor$.*

Proof. By Proposition 3, $\hbar_1(x) = \hat{h}(x, B_2, B_4, B_7) = \hat{h}(B_4, B_7, x, B_2)$. ∎

Corollary 3 identifies when $\hbar_1$ and $\hbar_3$ can be solved for x uniquely.

Corollary 3. *Let $z \in \mathbb{Z}_{256}$ be any constant. The equation $\hbar_3(x) = z$ has a unique solution for $x \in \mathbb{Z}_{256}$ if and only if B_2 is odd. Similarly, the equation $\hbar_1(x) = z$ has a unique solution if and only if B_7 is odd.*

Proof. By Proposition 5 and the fact that $\gcd(B_2, 256) = 1$ if and only if B_2 is odd. ∎

Finally, Proposition 7 counts the size of the preimage set $\hat{h}^{-1}(z)$ when two of the inputs to $\hat{h}$ are fixed.

Proposition 7. *Let $B_0, B_2 \in \mathbb{Z}_{256}$ be any constants, and define $Q : \mathbb{Z}_{2^{16}} \to \mathbb{Z}_{256}$ by $Q(x, y) = \hat{h}(B_0, B_2, x, y)$ whenever $(x, y) \in \mathbb{Z}_{256}^2$. For every $z \in \mathbb{Z}_{256}$, $|Q^{-1}(z)| = 256$.*

Proof (Sketch). Let $z \in \mathbb{Z}_{256}$; let $d = \gcd(B_2, 256)$; and let $S = \{y \in \mathbb{Z}_{256} : \hat{h}(B_0, B_2, x, y) = z$ has a solution for $x \in \mathbb{Z}_{256}\}$. From Proposition 5 it can be shown that $|Q^{-1}(z)| = d|S|$. From Equation 30, and using an argument similar to that given in Proposition 6, it can be shown that $|S| = 256/d$. Therefore, $|Q^{-1}(z)| = d \cdot 256/d = 256$. ∎

These properties illustrate how number theory can be used to gain insight into the structure of the nonlinear output function in Gifford's cipher.

References

[1] Beker, Henry; and Fred Piper, *Cipher Systems: The Protection of Communications*, John Wiley (New York, 1982).

[2] Berlekamp, Elwyn R., *Algebraic Coding Theory*, Aegean Park Press (Laguna Hills, CA, 1984).

[3] Brickell, Ernest F.; and Andrew M. Odlyzko, "Cryptanalysis: A survey of recent results," *Proceedings of the IEEE*, **76**:5 (May 1988), 578–593.

[4] Cain, Thomas R., "How to break Gifford's cipher," CMSC-693 Project, Computer Science Department, University of Maryland Baltimore County (May 28, 1993). 57 pages.

[5] Cain, Thomas R.; and Alan T. Sherman, "Cryptanalysis of filter generators using the rational canonical decomposition of the feedback function" (1994), in preparation.

[6] Char, Bruce W.; K. Geddes; G. Gonnet; B. Leong; M. Monagan; and S. Watt, *First Leaves: A Tutorial Introduction to Maple V*, Springer-Verlag (1992).

[7] Cormen, Thomas H.; Charles E. Leiserson; and Ronald L. Rivest, *Introduction to Algorithms*, MIT Press and McGraw-Hill (Cambridge, MA, and New York, 1990).

[8] Dawson, Ed, "Linear feedback shift registers and stream ciphers" in [30], Chapter 8 (1990), 106–119.

[9] Dawson, Ed; and Andrew Clark, "Divide and conquer attacks on certain classes of stream ciphers," *Cryptologia*, **XVIII**: 1 (January 1994), 25–40.

[10] Ganesan, Ravi; and Alan T. Sherman, "Statistical techniques for language recognition: An introduction and guide for cryptanalysts," *Cryptologia*, **XVII**:4 (October 1993), 321–366.

[11] Ganesan, Ravi; and Alan T. Sherman, "Statistical techniques for language recognition: An empirical study using real and simulated English" (September 27, 1993), *Cryptologia*, to appear.

[12] Gary, Michael R.; and David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman (1979).

[13] Giesbrecht, Mark, "Fast algorithms for matrix normal forms" in *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, ACM Press (1992), 121–130.

[14] Gifford, David K.; Dawn Heitmann; David A. Segal; Robert G. Cote; Kendra Tanacea; and David E. Burmaster, "Boston Community Information System 1986 experimental test results," technical report MIT/LCS/TR-397, MIT Laboratory for Computer Science (August 1987).

[15] Gifford, David K.; John M. Lucassen; and Stephen T. Berlin, "The application of digital broadcast communication to large scale information systems," *IEEE Journal on Selected Areas in Communications*, **SAC-3**:3 (May 1985), 457–467.

[16] Gifford, David K.; and David Andrew Segal, "Boston Community Information System 1987–1988 experimental test results," technical report MIT/LCS/TR-422, MIT Laboratory for Computer Science (May 1989).

[17] Gill, Arthur, *Linear Sequential Circuits: Analysis, Synthesis, and Applications*, McGraw-Hill (New York, 1966).

[18] Golić, Jovan Dj.; and Miodrag J. Mihaljević, "A generalized correlation attack on a class of stream ciphers based on the Levenshtein distance," *Journal of Cryptology*, **3**:3 (1991), 201–212.

[19] Gollmann, Dieter; and William G. Chambers, "Clock-controlled shift registers: A review," *IEEE Journal on Selected Areas in Communications*, **7**:4 (May 1989), 525–533

[20] Golomb, Solomon, *Shift Register Sequences*, Aegean Park Press (Laguna Hills, CA, 1982).

[21] Hellman, Martin E., "A cryptanalytic time-memory trade-off," *IEEE Transactions on Information Theory*, **IT-26**:4 (1980), 401–406.

[22] Hoffman, Kenneth; and Ray Kunze, *Linear Algebra*, second edition, Prentice-Hall (1971).

[23] Hungerford, Thomas W., *Algebra*, Springer-Verlag (New York, 1974).

[24] Jacob, Bill, *Linear Algebra*, W. H. Freeman and Company (New York, 1990).

[25] Key, Edwin L., "An analysis of the structure and complexity of nonlinear binary sequence generators," *IEEE Transactions on Information Theory*, **IT-22**:6 (November 1976), 732–736.

[26] Klapper, Andrew, "The vulnerability of geometric sequences based on fields of odd characteristic," *Journal of Cryptology*, **7**:1 (winter 1994), 33–51.

[27] Knuth, Donald E., *Seminumerical Algorithms* in *The Art of Computer Programming*, Vol. 2, second edition, Addison-Wesley (Reading, MA, 1981).

[28] Koblitz, Neal, *A Course in Number Theory and Cryptography*, Springer-Verlag (New York, 1987).

[29] Lomonaco, Samuel J. Jr., "Factoring large integers by solving Boolean equations," Computer Science Department, University of Maryland Baltimore County (1989), unpublished manuscript.

[30] Loxton, J. H., ed., *Number Theory and Cryptography*, Cambridge University Press, London Mathematical Society Lecture Note Series, No. 154 (Cambridge, Great Britain, 1990).

[31] *Macsyma Reference Manual*, version ten, Mathlab Group, MIT Laboratory for Computer Science (January 1983).

[32] Marsh, R. W., *Table of Irreducible Polynomials Over GF*(2) *Through Degree* 19, National Security Agency (Washington D.C., 1957).

[33] Meier, Willi; and Othmar Stafelbach, "Fast correlation attacks on certain stream ciphers," *Journal of Cryptology*, **1**:3 (1989), 159–176.

[34] Nivan, Ivan; and Herbert S. Zuckerman, *An Introduction to the Theory of Numbers*, John Wiley (New York, 1980).

[35] Peterson, W. Wesley; and E. J. Weldon, *Error-Correcting Codes*, MIT Press (Cambridge, Mass. 1972).

[36] Rhee, Man Young, *Cryptography and Secure Communications*, McGraw-Hill (Singapore, 1994).

[37] Ronse, Christian, *Feedback Shift Registers*, Lecture Notes in Computer Science 169, G. Goos and J. Hartmanis, eds., Springer-Verlag (Berlin, 1984).

[38] Rueppel, Rainer A., *Analysis and Design of Stream Ciphers*, Springer-Verlag (New York, 1986).

[39] Rueppel, Rainer A., "Stream ciphers" in [42], Chapter 2 (1992), 65–134.

[40] Shannon, Claude E., "Communication theory of secrecy systems," *Bell System Technical Journal*, **28** (October 1949), 656–715.

[41] Siegenthaler, T., "Decrypting a class of stream ciphers using ciphertext only," *IEEE Transactions on Computers*, **C-34**:1 (January 1985), 81–85.

[42] Simmons, Gustavus J., editor, *Contemporary Cryptology: The Science of Information Integrity*, IEEE Press (Piscataway, NJ, 1992).

[43] Watkins, David S., *Fundamentals of Matrix Computations*, Wiley and Sons (New York, 1991).

[44] Zeng, Kencheng; Chung-Huang Yang; Dah-Yea Wei; and T. R. N. Rao, "Pseudorandom bit generators in stream-cipher cryptography," *Computer*, **24**:2 (February 1991), 8–17.