Attacks on the Firekite cipher

Thomas Johansson[1], Willi Meier[2] and Vu Nguyen[1]

[1] Lund university, Lund, Sweden, thomas.johansson@eit.lth.se, vu.nguyen@eit.lth.se
[2] FHNW, Windisch, Switzerland, willi.meier@fhnw.ch

Abstract. Firekite is a synchronous stream cipher using a pseudo-random number generator (PRNG) whose security relies on the hardness of the *Learning Parity with Noise* (LPN) problem. It is one of a few LPN-based symmetric encryption schemes and it can be very efficiently implemented on a low-end SoC FPGA. The designers, Bogos, Korolija, Locher and Vaudenay, demonstrated appealing properties of Firekite such as requiring only one source of cryptographically strong bits, small key size, high attainable throughput, and a concrete measurement for the bit level security depending on the selected practical parameters.
We propose distinguishing and key-recovery attacks on Firekite by exploiting the structural properties of its PRNG. We adopt several *birthday-paradox* techniques to show that a particular sum of Firekite's output has a low Hamming weight with higher probability than the random case. We achieve the best distinguishing attacks with complexities $2^{66.75}$ and $2^{106.75}$ for Firekite's parameters corresponding to 80-bit and 128-bit security, respectively. By applying the distinguishing attacks and an additionally suggested algorithm, one can also recover the secret matrix used in the Firekite PRNG, which is built from the secret key bits. This key recovery attack works on most large parameter sets and has slightly larger complexity, for example $2^{69.87}$ on the 80-bit security parameters $n = 16384, m = 216, k = 216$.

Keywords: PRNG · Firekite PRNG · Birthday paradox · k-list algorithm · LPN · LPN-based symmetric encryption

1 Introduction

Since Shor [Sho99] in his seminal work introduced quantum algorithms that effectively broke the *discrete-log* and *factoring* problems, researchers have set their sights to cryptographic alternatives that promise to be quantum-resistant such as *lattice-based* or *code-based* cryptography. In particular, cryptographic primitives whose security relies on *learning problems*, such as *Learning Parity with Noise* (LPN), *Learning with Errors* (LWE), and the closely related Ring-LPN, are receiving great attentions as they are built on supposedly hard problems[1]. Moreover, Impagliazzo and Levin showed that cryptography is only possible if efficient learning is not [IL90]. Besides the absence of an LPN-solving quantum algorithm, LPN-based constructions are desired as they can be efficiently implemented using mainly XOR ("exclusive or") operations, thus achieving popularity in lightweight cryptography on constrained, low-powered devices. However, most of such proposed constructions are inclined towards asymmetric cryptography and they have their own disadvantages. This includes producing and extracting randomness (cryptographically secure bits) from an entropy-limited source, causing a significant overhead cost [HDWH12, Sho99], and they also often require large public keys.

Bogos, Korolija, Locher and Vaudenay [BKLV21] have proposed *Firekite*, a synchronous symmetric cipher, using an LPN-based PRNG which requires only one cryptographically

[1] LPN with adversarial errors is $\mathcal{NP}$-hard.

strong bit vector to construct the secret matrix key. A small key size is attained by moving from an LPN problem to a Ring-LPN problem [HKL+07]. Their study conjectures that the corresponding Ring-LPN instance remains hard to solve when using said matrix instead of a fully random matrix. They demonstrated that using the Firekite noise distribution for an LPN instance is still secure and there is a "partial" transformation to an LPN instance. Using the best BKW-style algorithm proposed by Levieil and Fouque [LF06], Firekite's designers measured the complexity to break the transformed LPN instances, thus derived concrete complexity results for attacking their cipher. The cipher's efficiency was tested in terms of the throughput, which is the number of bytes encrypted or decrypted per second using both desktop computers and FPGAs. They also showcased that, given dedicated hardware, the Firekite PRNG can be parallelized, hence throughput improved substantially for larger parameters.

One can draw many parallels between Firekite and the closely related LPN-C [GRS08b]; in particular, both involve computing a noisy product using a secret random matrix $\mathbf{M}$ and a random error vector $\mathbf{e}$. However, LPN-C further requires an error correcting code $\mathcal{C}$ and the error vectors are drawn from a Bernoulli distribution, as opposed to being bounded as in the Firekite PRNG. This could make the decrypting process fail once the error weight exceeds the code's error capacity. This drawback could be amended by truncating the binomial distribution to make sure not too many bits are set in the error vectors. However, it is speculated that doing so may have a negative impact to the security of LPN-C [BKLV21]. Furthermore, LPN-C inherently requires a large random secret matrix and draws two uniformly random vectors for every invocation of the encryption algorithm. Hence, it becomes infeasible to implement it efficiently when implemented in a constrained environment. Firekite, besides avoiding such undesirable features, surpasses LPN-C by not requiring fresh random bits for each output block.

Even more important than constructing schemes that are potentially quantum secure, it is crucial to try to attack them with the most suitable approaches to better understand their security.

1.1 Contributions

In this work, we propose both distinguishing and key-recovery attacks for Firekite. We observe that the secret matrix is fixed throughout every round of encryption. Hence, if the vector components in the internal states collide to the zero codeword, the output by Firekite, when combined together appropriately, result in unusually low weight sums and can be detected. In other words, finding such occurrences amounts to solving a birthday paradox problem with a specific target weight.

We then consider the secret matrix as the generator for a code and by carefully determining which positions in the above combinations are free of errors, we describe a key-recovery attack with a slightly higher complexity than that of the distinguishing attack.

As examples, we apply the distinguishing attacks on the Firekite cipher with specific parameters for 80-bit and 128-bit to better understand Firekite's security. In particular, we launch both, a distinguishing attack and a key recovery attack on parameters $n = 16384, m = 216, k = 216$ with complexity $2^{68.87}$ and $2^{69.97}$. As there are many choices of parameter sets for each security level, the complexity numbers vary a bit depending on selected parameter sets.

1.2 Related work

Due to their difficulty, either proved or conjectured, LPN and Ring-LPN have made their ways into many cryptographic constructions such as human identification protocols which were firstly introduced by Hopper and Blum [HB01], later modified and improved to HB$^+$

and HB$^{\#}$ [JW05, KS06, GRS08a]. Recent LPN-based encryption schemes that can be found are HELEN by Duc and Vaudenay [DV13], or LPN-C by Gilbert et al. [GRS08b]. Using the RING-LPN variant, Heyse et al. [HKL+12] proposed an efficient two-round identification protocol in constrained environments, called Lapin. LPN also proves useful in other applications, e.g., message authentication codes (MACs) [KPV+17, DKPW12], pseudo-random generators [ACPS09, BFKL93], or CCA-secure public-key encryption schemes [YZ16].

Since its introduction, LPN has drawn a plethora of LPN-solving studies with different approaches. It is natural to see LPN as a decoding problem; hence a generic decoding technique applies. Attacks on LPN can be categorized as *Information-set decoding* (ISD) or BKW-type algorithms and they prove advantageous in different scenarios, namely, low noise-rate and constant noise. ISD was first introduced by Prange [Pra62], and further improved by Leon[Leo88], Lee and Brickell [LB88] and Stern [Ste93]. Recently, several methods have been proposed to gain better attacks, to name a few, *Ball-collision technique* by Bernstein et al. [BLP11], *representation technique* by Becker, Joux, May, Meurer [BJMM12], or the state-of-the-art algorithm [BV15]. On the other hand, BKW began with the foundation laid by Blum, Kalai, and Wasserman [BKW03]. Besides the improvement by Levieil and Fouque who used Walsh-Hadamard transform to recover several bits of the secret vector, using a limited number of queries, notable advancements can be found such as covering codes by Guo et al. [GJL14], or on the use of the *Generalized birthday attack* (GBA) [Wag02] as in [Kir11].

GBA is one of the most pertinent generic attacks in cryptology, in particular, analyzing the security of an LPN-based cryptographic scheme. There have been many notable works related to the generalized birthday problem. Our study is inspired by the seminal works of Wagner [Wag02] and May et al.'s approximate k-list algorithm [BM17].

1.3 Organization

The paper is organized as follows. Section 2 presents preliminary and background knowledge regarding the LPN problem and its variants such as RING-LPN. A brief review of the LPN-based Firekite PRNG, and how it gave rise to the Firekite synchronous stream cipher follows. We then describe our idea, and formally analyze our attack for Firekite in Section 3. In Section 4, we attack different parameters proposed for Firekite and verify our approach by a simulation with smaller parameters. We describe our key-recovery attack in Section 5 and discussions on how to improve Firekite finally concludes our work.

2 Background

Whereas the LPN problem usually finds its cryptographic applications in the public-key domain, we will be interested in its application in symmetric cryptography. In particular, we have seen constructions of a few synchronous stream ciphers [GRS08b, BKLV21] based on LPN.

A synchronous stream cipher is a symmetric cipher, in which a stream of pseudorandom bits is generated independently of the plaintext and ciphertext messages, and then bitwise XOR-ed to the plaintext, to encrypt, or to the ciphertext, in order to decrypt. Cryptanalytic attacks either aim at distinguishing the output of the pseudorandom bit generator from random, recovery of its state, or key recovery. As known plaintext for a segment of ciphertext implies knowledge of keystream for the same segment, a known plaintext attack of a synchronous stream cipher assumes that a large part of keystream is available to an attacker which is only limited by the maximum number of keystream bits allowed to be output for a same key. Distinguishing attacks on the (known) keystream are relevant to the security of stream ciphers as well: Depending on the nonranomness detected, some

information on the plaintext may be leaked. For some stream ciphers, a distinguishing feature can even be elaborated to a key recovery attack, as is the case for the distinguishing property we shall derive for Firekite.

2.1 The LPN problem

LPN is an important problem in cryptography. It appears as one of main problems on which we base post-quantum cryptography. Due to the existence of fast algorithms for quantum computers that can solve the factorization and the discrete logarithm problems [Sho99], the LPN problem (and the related LWE problem) including its different versions are of great interest. No fast quantum algorithm that solves the LPN problem is known. Although current omnipresent symmetric encryption schemes such as AES will likely not be rendered obsolete in the near future, studies in post-quantum cryptography, namely aforementioned works, are of absolute necessity. We need post-quantum cryptographic primitives to have *efficiency, confidence,* and *usability* [Ber09].

LPN-based cryptographic constructions are also appealing, since only simple operations such as bit-wise addition (XOR) and scalar products are used. This can give rise to efficient algorithms or protocols.

The LPN problem can informally be described as the problem of solving a noisy binary system of equations. We formally define it below.

Let Ber_η be the Bernoulli distribution and $X \sim \mathsf{Ber}_\eta$ be a random variable where $X = \{0, 1\}$. Then $\mathbf{Pr}[X = 1] = \eta$, $\mathbf{Pr}[X = 0] = 1 - \eta$. Denote by $\mathbf{x} \xleftarrow{U} \{0,1\}^m$ the event that a vector $\mathbf{x}$ is drawn uniformly from $\{0,1\}^m$.

Definition 1. (LPN Oracle). Let $\mathbf{x} \xleftarrow{U} \{0,1\}^m$ and $\eta \in (0, \frac{1}{2})$. An LPN *oracle* Π_{LPN} for $\mathbf{x}$ and η returns pairs of the form

$$\left(\mathbf{g} \xleftarrow{U} \{0,1\}^m, \langle \mathbf{x}, \mathbf{g} \rangle + e\right),$$

where $e \leftarrow \mathsf{Ber}_\eta$, and $\langle \mathbf{x}, \mathbf{g} \rangle$ denotes the scalar product of vectors $\mathbf{x}$ and $\mathbf{g}$.

Definition 2. (LPN problem). Given an LPN oracle Π_{LPN} with parameters m and η. The (m, η)-LPN problem is finding the secret vector $\mathbf{x}$. An algorithm $\mathfrak{A}_{\text{LPN}}(T, N, \delta)$ asking for at most N oracle queries, using time at most T solves (m, η)-LPN if

$$\mathbf{Pr}\left[\mathfrak{A}_{\text{LPN}}(T, N, \delta) = \mathbf{x} : \mathbf{x} \xleftarrow{U} \{0,1\}^m\right] \geq \delta.$$

The definition above is known as the search version of the LPN problem. In the decisional version of the LPN problem, the objective is to distinguish Π_{LPN} from a source giving uniformly random bits. The search and decisional versions are proved to be computationally equivalent [KSS10].

We briefly look at a subclass of LPN problems called Ring-LPN which proves to be useful in general and specifically used in the Firekite PRNG. Let f be a polynomial over $\mathbb{Z}_2$ and $R = \mathbb{Z}_2[x]/(f)$ denote the quotient ring. Hence R consists of all polynomials over $\mathbb{Z}_2$ of degree less than that of f. We say $r \leftarrow \mathsf{Ber}_\eta^R$ if the coefficients of the ring element $r \in R$ are assigned independently following the distribution Ber_η. If r is drawn uniformly from R, we write $r \xleftarrow{U} R$. The Ring-LPN problem can be defined similarly to the standard LPN problem.

Definition 3. (Ring-LPN oracle). Let $s \xleftarrow{U} R$ and $\eta \in (0, \frac{1}{2})$. A Ring-LPN oracle $\Pi_{\text{Ring-LPN}}$ for s and η returns pairs of the form

$$(r \xleftarrow{U} R, r \cdot s + e),$$

where $e \leftarrow \mathsf{Ber}_\eta^R$.

Definition 4. (RING-LPN problem). Given a RING-LPN oracle $\Pi_{\text{RING-LPN}}$ with parameters η and a polynomial ring R. The RING-LPN problem is finding the secret polynomial $s \in R$. An algorithm $\mathfrak{A}_{\text{RING-LPN}}(T, N, \delta)$ asking for at most N oracle queries, using time at most T solves the RING-LPN problem if

$$\mathbf{Pr}\left[\mathfrak{A}_{\text{RING-LPN}}(T, N, \delta) = s\right] \geq \delta.$$

It is worth pointing out the essential difference between LPN and RING-LPN. If we query the LPN oracle N times, then we can collect an $m \times N$ matrix $\mathbf{G} = \left(\mathbf{g}_1^{\mathrm{T}} \cdots \mathbf{g}_N^{\mathrm{T}}\right)$ and each column is generated independently. In the case of RING-LPN, only one polynomial r is generated uniformly random in R. If we consider a polynomial as its coefficient vector, only the first column $\mathbf{r}$ is drawn uniformly random. The other columns are obtained via shifting $\mathbf{r}$. While the LPN problem has been shown to be $\mathcal{NP}$-hard in the worst case [BMVT78], the hardness of RING-LPN is not known. However, there is a reduction from RING-LPN to LPN and the assumption is that RING-LPN is also hard.

2.2 Firekite's PRNG and Firekite construction

We recall that the decisional version of the LPN assumption can be interpreted as one can not efficiently distinguish an LPN oracle from a source providing random bit vectors of length m. Naturally, it can be extended into stating that distinguishing a noisy product of an $m \times n$ matrix M and a secret vector $\mathbf{v}$, i.e., $\mathbf{vM} + \mathbf{e}$ from a random n-bit vector in $\mathbb{Z}_2$, where $\mathbf{e}$ is a sparse n-bit noise vector is hard. As an example, LPN-C further used a $[k, n]$ error correcting code $\mathcal{C}$ with a generator matrix $\mathbf{G}$ to encode a plaintext $\mathbf{x}$ to a ciphertext $\mathbf{c}$ through

$$\mathbf{c} = \mathbf{xG} + \mathbf{vM} + \mathbf{e}.$$

However, this construction inherently asks the source to produce random $\mathbf{v}$ and $\mathbf{e}$ for encrypting a single plaintext. The Firekite PRNG circumvents this problem by extracting both $\mathbf{v}$ and $\mathbf{e}$ from the noisy product and feeding them iteratively into the next encryption invocations. Out of n bits, one can spare $m + k \cdot \log n$ bits to initialize the next round of Firekite[2]. Let $||$ denote the usual concatenation of vectors. We write

$$\mathbf{vM} + \mathbf{e} = (\mathbf{g}||\mathbf{v}'||\mathbf{c}_e).$$

Assuming $n \gg m$, one can split the noisy product into three components. Obviously, m bits are used for producing the next vector $\mathbf{v}$. Since $\mathbf{e}$ is only required to be a sparse n-bit vector, we can have a compact representation of the next noise vector, called $\mathbf{c}_e$. Then, the remaining bits, forming $\mathbf{g}$, are the PRNG's output. We are now in the position to describe the Firekite PRNG formally.

Let m, n, and k be some integer parameters, where $n \gg m$ and n is a power of 2. A secret key $\mathbf{M}$ is a binary matrix of size $m \times n$, and $\mathbf{w}$ is a vector of length $m + k \log n < n$. Together they form a pair $(\mathbf{M}, \mathbf{w})$, the *state* of the PRNG. We define $\mathbf{w} = \mathbf{v}||\mathbf{c}_e$, where $\mathbf{v}$ and $\mathbf{c}_e$ are of length m and $k \log n$, respectively. As stated above, $\mathbf{M}$ is fixed and $\mathbf{w}$ is updated for every iteration. It is straightforward to assign $\mathbf{v} = \mathbf{v}'$. To get the next error vector $\mathbf{e}$, we further parse $\mathbf{c}_e = \mathbf{c}_1||\mathbf{c}_2||...||\mathbf{c}_k$ where $\mathbf{c}_i$ is of length $\log n$. Hence, each $\mathbf{c}_i$ can be seen as the binary presentation of a non-negative integer less than n. Therefore, $\mathbf{c}_e$ encodes an n-bit error vector of weight at most k. In particular, let $\mathbf{b}_{\mathbf{c}_j}$ be the unit vector of length n, where the bit at the position represented by $\mathbf{c}_j$ is 1. Then the error vector $\mathbf{e}$ is defined as $\mathbf{e} = \bigvee_{j=1}^{k} \mathbf{b}_{\mathbf{c}_j}$. The execution of the Firekite PRNG is described by Algorithm 1.

At each iteration, the PRNG's input is its state $(\mathbf{M}, \mathbf{w})$, where the first m bits and the remaining $k \log n$ bits of $\mathbf{w}$ are set to be $\mathbf{v}$ and $\mathbf{c}_e$ respectively. Then the error vector

[2]Throughout the paper, log(.) denotes logarithm to base 2.

Algorithm 1: Firekite PRNG

Input: An $m \times n$ secret matrix M and a nonce $\mathbf{w}$, $r > 0$: randomization rounds.

while $r \neq 0$ **do**
  Parse: $\mathbf{w} = \mathbf{v}||\mathbf{c}_1||\mathbf{c}_2||...||\mathbf{c}_k$;
  $\mathbf{e} \leftarrow \bigvee_{j=1}^{k} \mathbf{b}_{\mathbf{c}_j}$;
  $\mathbf{g}||\mathbf{w}' := \mathbf{vM} + \mathbf{e}$; `// Randomization`
  $\mathbf{w} \leftarrow \mathbf{w}'$;
  $r \leftarrow r - 1$;
while 1 **do**
  Parse: $\mathbf{w} = \mathbf{v}||\mathbf{c}_1||\mathbf{c}_2||...||\mathbf{c}_k$;
  $\mathbf{e} \leftarrow \bigvee_{j=1}^{k} \mathbf{b}_{\mathbf{c}_j}$;
  $\mathbf{g}||\mathbf{w}' := \mathbf{vM} + \mathbf{e}$; `// Generating keystream`
  $\mathbf{w} \leftarrow \mathbf{w}'$;
  Return: $\mathbf{g}$

$\mathbf{e}$ is derived from its concise representation $\mathbf{c}_e$ and the noisy n-bit product is computed as $\mathbf{vM} + \mathbf{e}$. This vector is again parsed into $\mathbf{g}$ and $\mathbf{w}'$ of length $d = n - m - k \log n$ and $m + k \log n$, respectively. The internal state is then updated to $(\mathbf{M}, \mathbf{w}')$ and $\mathbf{g}$ is the output of the PRNG. The number r of randomization rounds is needed to guarantee that $\mathbf{v}$ is fully random when Firekite outputs its keystream [BKLV21].

Firekite is a synchronous stream cipher that makes use of this PRNG to produce the d-bit keystream $\mathbf{g}$ directly. Therefore, for each invocation, d-bit data of a plaintext is encrypted, and the next output of Firekite depends on the updated internal state. The designers pointed out that, for practicality, the parameters m, n, and k need to be large which in turn makes the secret key $\mathbf{M}$ big. In order to solve this problem, they proposed the following technique, which turns the LPN instance into a Ring-LPN instance. Consider $R = \mathbb{Z}_2[x]/(X^b - 1)$, i.e, the polynomial ring with binary coefficients reduced modulo $X^b - 1$ such that $(X^b - 1)/(X - 1)$ is irreducible. It is well known that every polynomial in R can be represented by its coefficient vectors in $\mathbb{Z}_2^b$. Pick $\mathbf{q}_1 \xleftarrow{U} \mathbb{Z}_2^b$ and define $\mathbf{q}_i = X^{i-1}\mathbf{q}_1, i = 1, ...b$, meaning that we shift the entries in the coefficient vectors $\mathbf{q}_1$ by $i - 1$ times. Hence, we can construct a $b \times b$ matrix $\mathbf{Q}$ by shifting the first column to the left consecutively $b - 1$ times. The secret matrix $\mathbf{M}$ is obtained by generating the first m rows, then dropping the last $b - n$ columns of $\mathbf{Q}$. Therefore, the secret key of Firekite PRNG is, in fact, the random b-bit vector $\mathbf{q}_1$ rather than an $m \times n$ matrix $\mathbf{M}$. The designers conjectured that using such $\mathbf{M}$ does not substantially reduce the security compared to a fully random matrix $\mathbf{M}$.

Table 1: Firekite's parameters for 80 and 128-bit security with the properties in terms of key size b, relative throughput α, and number of initilization rounds r.

Parameters			Properties			
m	n	k	Key size (b)	α	r	Security level
216	1024	16	1061	0.63	18	82.76
216	2048	32	2053	0.72	18	82.76
216	16,384	216	16,421	0.80	21	80.68
352	2048	32	4099	0.74	18	129.07
352	8192	120	8219	0.77	18	128.99
352	16,384	228	16,421	0.78	18	128.93

Table 1 shows a few sets of suggested parameters for Firekite that corresponds to 80 and 128 security bit levels. Other proposed parameter sets can be found in [BKLV21].

To derive an estimation of the concrete security of Firekite, one faces two problems: First, the noise vectors from Firekite has weight at most k and the noise distribution is not binomial, as opposed to a standard LPN instance. Second, an adversary only sees a part of the noisy product. Therefore, it is necessary to prove that using Firekite noise distribution for an LPN variant is still hard, and the underlying problem of solving Firekite is as hard as LPN.

The first question is solved as follows. Let $\Delta(\mathbf{e})$ denote the Hamming weight of a n-bit vectors and e_j is the j-th bit of $\mathbf{e}$. If $\mathbf{e}$ comes from Firekite, then $\mathbf{Pr}[e_j = 0] = \left(\frac{n-1}{n}\right)^k$. Therefore, the expected Hamming weight of Firekite noise is

$$\mathbb{E}[\Delta^{\text{Firekite}}(\mathbf{e})] = n\left(1 - \left(\frac{n-1}{n}\right)^k\right),$$

and one can show that

$$\frac{2}{3}k < \mathbb{E}[\Delta^{\text{Firekite}}(\mathbf{e})] < k.$$

In a standard LPN problem with parameters η and m, $\mathbb{E}[\Delta^{\text{LPN}}(\mathbf{e})] = \eta m$ and $\mathbf{Pr}[\Delta^{\text{LPN}}(\mathbf{e}) = \lfloor \mathbb{E}[\Delta^{\text{LPN}}(\mathbf{e})] \rfloor] \in \Omega(1/n)$. Therefore, given such an LPN instance, we set k such that $\eta m \leq k$, e.g., $k := \frac{3}{2}\eta m$. Then the noise of this LPN instance could come from the Firekite noise distribution with probability at least $\Omega(1/n)$. In other words, if the LPN instance with the Firekite noise distribution can be broken efficiently, any standard LPN instance can also be broken with $O(n)$ more work.

As for the underlying problem of solving Firekite, Firekite's designers were able to show that it is *at most* as hard as the LPN problem, and they also conjectured that the reverse is also true [BKLV21]. Using this transformation to attack Firekite with the most efficient LPN-solving algorithm, namely the one by Levieil and Fouque[LF06], they were able to derive the concrete proposed parameters for the different security levels.

2.3 The problem of observing noisy codewords from an unknown code

The task of recovering *partially* the secret matrix $\mathbf{M}$ (by observing vectors $\mathbf{g}_i$) can be seen as identifying an unknown code by observing noisy codewords. The problem often arises in different contexts [MGB12], especially in analyzing cryptosystems where encryption involves error-correcting codes and the transmission is carried over a noisy channel (e.g., a binary symmetric channel). General approaches consist of three steps: First, arranging noisy codewords as rows of a matrix, then running the Gaussian elimination, and finally from the non-echelon part finding sums of vectors that are candidates to construct dual codewords (i.e, parity-check equations). Instead of looking at only columns that sum to 0, Sicot, Houcke and Barbier argued that sparse sums of columns can also be candidates for being dual codewords [BSH06, SHB09]. Therefore, the last step can be reduced to an instance of the well known *close neighbors search problem*. Beside the *projection method* proposed by Cluzeau and Finiasz [CF09] which aimed to find sparse sums of p columns (with complexity of order $\Omega(n^{p/2})$ when p is even) that are equal in some positions using birthday paradox and hashtables, there have been many improvements and extensive studies to the close neighbors search problem recently. In particular, one being the *Dubiner method* which later was applied by Carrier and Tillich in their generalized approach [CT19]. Their algorithm only performed a *partial* Gaussian elimination for the second steps. The argument is that the Gaussian elimination increases the noise by combining noisy codewords, hence it is more likely to obtain sparse sums in the early stage

of the Gaussian elimination and minimizing the dual codewords that might have been undetected by Sicot-Houcke-Barbier algorithm [CT19]. Moreover it also allowed them to find dual codewords of much larger weight (compared to the full Gaussian elimination) with reasonable complexities.

In practice, the recovery of an unknown code by observing noisy codewords concerns useful families of codes, such as cyclic codes, convolutional codes, turbo codes, or the ubiquitous LDPC which is important as finding low-weight dual codewords is essential in determining communication components such as unknown interleaver [BSH06, Tix15] or reconstructing other families of codes.

In the next section we introduce a new method, namely finding a small number of noisy codewords summing to the zero codeword through a generalized birthday type of algorithm.

3 The proposed distinguishing algorithm

In this section, we aim to give a brief description of the idea used in our distinguishing attacks on Firekite. We firstly observe that the secret key matrix $\mathbf{M}$ is fixed throughout the rounds of Firekite; hence, the keystream output by Firekite PRNG is subjected to accumulating non-randomness. Let us look at the Firekite PRNG, fulfilling

$$\mathbf{vM} + \mathbf{e} = (\mathbf{g}||\mathbf{v}'||\mathbf{c}_e),$$

where $\mathbf{v}'$ and $\mathbf{c}_e$ are used in the next iteration by assigning $\mathbf{v}' = \mathbf{v}$ and $\mathbf{e} = \bigvee_{j=1}^{k} \mathbf{b}_{\mathbf{c}_j}$, and $\mathbf{g}$ is the PRNG's output. In the initial part of the attack, we concentrate on assuming the knowledge of

$$\mathbf{g} = \mathbf{vM}' + \mathbf{e}',$$

where $\mathbf{g}$ is a known d-bit vector, $\mathbf{M}'$ is now considered as an $m \times d$ secret binary matrix (obtained from the first d columns of the original $\mathbf{M}$ matrix) and $\mathbf{e}'$ is a secret d-bit noise vector, being the first d positions of $\mathbf{e}$. It is known that $\Delta(\mathbf{e}) \leq k$ (which is small); hence, the weight of $\mathbf{e}'$ is also small. The expected weight of $\mathbf{e}'$, denoted by $\hat{k}$, can be computed from k, n and d, since it is assumed that the ones in $\mathbf{e}$ are uniformly distributed among all d positions.

In a synchronous stream cipher attack, we assume that an adversary has access to a long output stream, which means access to a large number of d-bit vectors $\mathbf{g}$. The set of these vectors is written as $\{\mathbf{g}_i, i = 1, \ldots, S\}$, where now $\mathbf{g}_i = \mathbf{v}_i\mathbf{M}' + \mathbf{e}'_i$ and for some S to be addressed in the following subsections. We first sketch the ideas behind our distinguishing attack, i.e., given an aforementioned set of vectors, decide whether they originate from Firekite or if they are random vectors.

Our goal is to find a subset of $\mathbf{g}_{i_j}$ vectors, $j = 1, ..., l$, such that the corresponding $\sum_{j=1}^{l} \mathbf{v}_{i_j} = \mathbf{0}$, i.e., we find a set of noisy codewords such that the underlying information vectors sum to zero. If l vectors $\mathbf{v}_{i_j}$, $j = 1, \ldots, l$, sum to zero, then the sum of the corresponding $\mathbf{g}_{i_j}$'s is expected to be of weight around $l \cdot \hat{k}$ with nonzero contributions coming only from the errors $\mathbf{e}'_{i_j}$. Indeed, we then have

$$\sum_{j=1}^{l} \mathbf{g}_{i_j} = \sum_{j=1}^{l} \mathbf{v}_{i_j}\mathbf{M}' + \sum_{j=1}^{l} \mathbf{e}'_{i_j} = \sum_{j=1}^{l} \mathbf{e}'_{i_j}.$$

Therefore, when l is not too large, e.g. $l = 4$ or $l = 8$, the expected weight in $\sum_{j=1}^{l} \mathbf{g}_{i_j}$ will be low if $\sum_{j=1}^{l} \mathbf{v}_{i_j} = 0$. Since d is much larger than $l \cdot \hat{k}$, such a weight is very unlikely if the vectors $\mathbf{g}_{i_j}$ are random vectors. In the Firekite PRNG, such a collision of vectors of length m (i.e, with probability proportional to 2^{-m}) guarantees a low weight vector of

length d. It is only intuitive to deduce that we can detect such occurrences more frequently than what is expected in the random case.

3.1 A basic algorithm for finding noisy codewords summing to the zero codeword

Recall that we want to find l different $\mathbf{g}_{i_j}$ vectors, $j = 1, \cdots, l$, such that the associated unknown vectors $\mathbf{v}_{i_j}$ sum to zero. Our approach is built from ideas from the generalized birthday attack [Wag02] and the BKW algorithm [BKW03]. A different but related approach is also May et al.'s *Match-and-Filter* algorithm [BM17].

In a simplified description following [Wag02], we set up l ($l = 2^t$ is a power of 2) lists of size 2^c filled by $\mathbf{g}_i$ vectors. We then combine the lists pairwise, resulting in a new list containing vectors created as a sum of two vectors, one from each initial list, such that some c predetermined positions are all zero. The expected number of vectors in the new list is 2^c. After the first step we have $l/2$ lists. We then perform the same procedure again, reducing another c positions to zero until one single list remains, i.e, after t steps. In the remaining list, we will finally examine whether there are vectors $\sum_{j=1}^{l} \mathbf{g}_{i_j}$ that are candidates to satisfy $\sum_{j=1}^{l} \mathbf{v}_{i_j} = 0$. In fact, they are quite easily detected, since if this is the case then $\sum_{j=1}^{l} \mathbf{g}_{i_j} = \sum_{j=1}^{l} \mathbf{e'}_{i_j}$, which has very low weight.

As in the BKW algorithm framework, one may use the same list for $\mathbf{g}_i$ vectors, providing we slightly increase the list size to roughly $3 \cdot 2^c$ [LF06]. Starting with a list $L^{(0)}$, we can write up a sequence of updated lists $L^{(0)} \to L^{(1)} \to L^{(2)} \cdots \to L^{(t)}$, where in each step we reduce another c positions. This means that $L^{(i)}$ have vectors where the first $i \cdot c$ positions are all zero. We formally describe this approach in Algorithm 2.

Algorithm 2: t-step Distinguisher

Input: A list $L^{(0)}$, with $\mathbf{g}_i$ vectors, $i = 1, ..., 3 \cdot 2^c$ ($|L^{(0)}| = 3 \cdot 2^c$), parameters $t = 2^l$, k, c_ω.

for $i = 1, ..., t$ **do**
  $L^{(i)} = \text{COMBINE}(L^{(i-1)})$; `// Combine list, cancelling c bits.`
minweight $= k \cdot l$;
for $\mathbf{g}' : \mathbf{L^{(t)}}$; `// Filtering low weight sums.`
do
  if $\mathsf{HammingWt}(\mathbf{g}') \leq$ minweight **then**
    minweight $= \mathsf{HammingWt}(\mathbf{g}')$
if minweight $\leq c_\omega$ **then**
  Return: Firekite
else
  Return: Random

Figure 1 and Figure 2 is a visualization of the COMBINE step for the $(i-1)$-th list $L^{(i-1)}$ and the filtering for the last list, respectively.

We need to consider complexity and memory of the algorithm. Let this computational complexity measured in simple operations be denoted C and the used memory in bits be denoted Mem. Its main parts are the $L^{(i)} = \text{COMBINE}(L^{(i-1)})$ steps in the loop. We assume that the vectors in the list $L^{(i-1)}$ are organized in a hash table. We have that the first $(i-1) \cdot c$ positions are all zero in all vectors in $L^{(i-1)}$, and they are again sorted in different buckets in the hash table according to the value of the next c positions, i.e., position $(i-1) \cdot c$ to $i \cdot c - 1$, for $i = 1, ..., t$. The COMBINE step now creates new vectors for the new list $L^{(i)}$ by adding together all possible pairs that are stored in the same

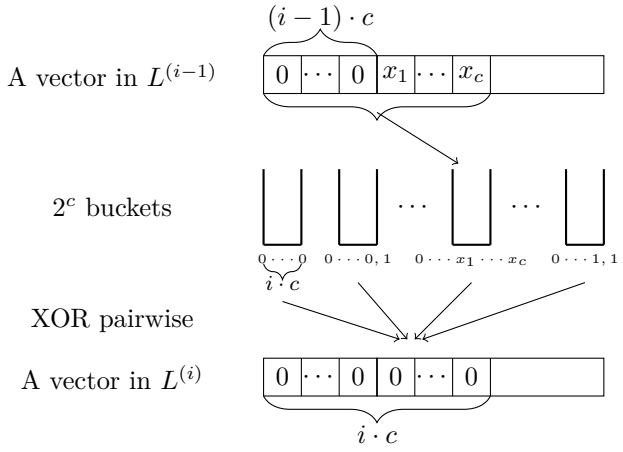


Figure 1: COMBINE for $L^{(i-1)}$.

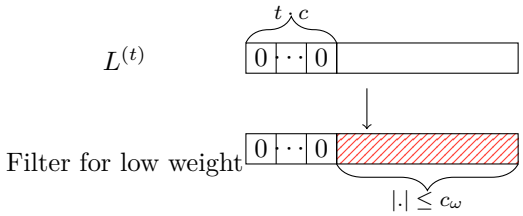


Figure 2: Filter $L^{(t)}$ with c_ω.

bucket. This will cancel out another c positions so that vectors in $L^{(i)}$ start with $i \cdot c$ zeros. New vectors are created until the list $L^{(i)}$ has cardinality $3 \cdot 2^c$ and the sorting procedure is repeated for the next iteration [3]. The complexity of one COMBINE step is then $3 \cdot 2^c$ bit-wise additions of vectors of length at most d and storing the result in memory. We adopt Firekite's designers' notation by letting p be the word-length of a bit-wise addition operation, i.e, the number of bits for which an XOR operation can be computed [4]. We write the cost of one d-bit XOR operation as $(1 + \lfloor d/p \rfloor)$. This procedure is repeated t times in Algorithm 2. The final check for low weight vectors actually does not need to go through all buckets, but only those with a low weight. This cost is then much smaller than the previous steps and can be disregarded. The complexity can thus be estimated as

$$C = t \cdot (3 \cdot 2^c) \cdot (1 + \lfloor d/p \rfloor).$$

The required memory Mem is the storage of two lists, altogether at most $\text{Mem} = 2 \cdot 3 \cdot 2^c \cdot d$ in bits.

Clearly, there is a possibility to reduce memory and complexity by not storing the full length d vector but only indices to the original vectors instead. This can become useful especially when d grows large and significantly adds to the complexity and memory required per step. In the next subsection, we investigate the success probability of the distinguisher.

[3] We might loose some vectors with this condition. In our simulation, we also tested to use all combinations in a bucket, hence resulting in a slightly larger list after each COMBINE step. One obvious improvement is to ignore buckets with fewer than two vectors.

[4] For example, the *Advance Vector Extension* AVX-512 allows XOR to have 512 bits computed per cycle.

3.2 An analysis of the success probability of the proposed algorithm

Since the added noise in the $\sum_{j=1}^{l} \mathbf{g}_{i_j} = \sum_{j=1}^{l} \mathbf{v}_{i_j}\mathbf{M}' + \sum_{j=1}^{l} \mathbf{e}'_{i_j} = \sum_{j=1}^{l} \mathbf{e}'_{i_j}$ expression becomes significant as l grows large, a low-weight sum from Firekite will become hard to distinguish as l grows (from the random case). We hence fix the number of algorithmic steps t to 2 or 3, corresponding to $l = 4$ and $l = 8$ in the proposed algorithm, respectively. Also, a vector formed as $\sum_{j=1}^{l} \mathbf{g}_{i_j} = \sum_{j=1}^{l} \mathbf{v}_{i_j}\mathbf{M}' + \sum_{j=1}^{l} \mathbf{e}'_{i_j} = \sum_{j=1}^{l} \mathbf{e}'_{i_j}$ will be called a *zero sum vector*. Furthermore, considering a sum of error vectors, e.g., $\sum_{j=1}^{l} \mathbf{e}'_{i_j}$, we say that a position is *error free* mod 2 if $\sum_{j=1}^{l} \mathbf{e}'_{i_j}$ is zero in that position; we say that a position is simply *error free* if all $\mathbf{e}'_{i_j}$ are zero in the position. Obviously, it can happens that a *double error* event occurs, i.e., two ones in the same position and $1 + 1 = 0$. We shall look more closely into this particular scenario in later subsections.

3.2.1 The case $l = 4$

Starting with $l = 4$, we are interested in knowing if a zero sum vector can be found in the final list. The expected number of zero sum vectors in the final list is denoted by N. We can expect

$$N = \binom{3 \cdot 2^c}{4} \cdot 2^{-m} \cdot 3 \cdot 2^{-c} \cdot P_{\text{nf}}$$

such zero sum vectors, which can be roughly explained as follows: First, there are $\binom{3\cdot 2^c}{4}$ possible combinations from the initial list. Among all such 4-sums, only a fraction 2^{-m} will correspond to a zero sum in $\mathbf{v}_{ij}, j = 1, \cdots, 4$. Second, there are 3 ways to choose 2 pairs as proceeded in Algorithm 2. We consider two particular pairs $\{\mathbf{g}_{i_1}, \mathbf{g}_{i_2}\}$ and $\{\mathbf{g}_{i_3}, \mathbf{g}_{i_4}\}$ summing to a zero sum, we further condition $\mathbf{g}_{i_1}$and $\mathbf{g}_{i_2}$ to cancel in the first c bits with probability 2^{-c} (the other pair automatically follows). Finally, we assume that $\mathbf{e}'_{i_1} + \mathbf{e}'_{i_2} + \mathbf{e}'_{i_3} + \mathbf{e}'_{i_4}$ is zero in the first $2c$ positions, i.e, error free mod 2. This probability is denoted P_{nf} (noise-free).

P_{nf} can be bounded by the probability that the first $2c$ positions are error free. For each $\mathbf{e}'_{i_1}$, there are at most k bits set, uniformly distributed among n positions[5], so the probability of one error vector, being error free in the first $2c$ position, is roughly $((n - 2c)/n)^k$. Therefore, we have

$$P_{\text{nf}} \geq ((n - 2c)/n)^{4k}.$$

Lemma 1. *When $l = 4$, we expect to have*

$$N > \binom{3 \cdot 2^c}{4} \cdot 2^{-m-c} \cdot 3 \cdot ((n - 2c)/n)^{4k}$$

zero sum vectors in the final list in Algorithm 2.

Then if we pick c_w to be slightly larger than $4 \cdot \hat{k}$, it is very likely that we will be successful, assuming that the expected number of zero sum vectors in the final list is more than one.

3.2.2 The case $l = 8$

Next, we investigate $l = 8$. Similar to the previous case, we can have an expression of the expected zero sum vector N as:

[5]The expected weight of $\mathbf{e}'$ from Firekite is smaller than k, but we can compute probabilities from error assignment in $\mathbf{e}$.

$$N = \binom{3 \cdot 2^c}{8} \cdot 2^{-m} \cdot 105 \cdot 2^{-4c} \cdot P_{\text{nf}}.$$

The explanation is again as follows: The number of different sums of 8 vectors that can be constructed is $\binom{3 \cdot 2^c}{8}$. Among them, we expect a fraction of 2^{-m} summing to the zero case. There are $7 \cdot 5 \cdot 3 = 105$ ways to form 4 pairs of 8 vectors. Consider the particular pairing $\{\mathbf{g}_{i_1}, \mathbf{g}_{i_2}\}$, $\{\mathbf{g}_{i_3}, \mathbf{g}_{i_4}\}$, $\{\mathbf{g}_{i_5}, \mathbf{g}_{i_6}\}$, and $\{\mathbf{g}_{i_7}, \mathbf{g}_{i_8}\}$. A sum constructed from this pairing will be in the final list of Algorithm 2 if $\mathbf{g}_{i_1} + \mathbf{g}_{i_2}$, $\mathbf{g}_{i_3} + \mathbf{g}_{i_4}$ and $\mathbf{g}_{i_5} + \mathbf{g}_{i_6}$ are all zero in the first c positions. Then $\mathbf{g}_{i_7} + \mathbf{g}_{i_8}$ has to be zero in the first c positions. The probability of this event for each choice of fixed indices is 2^{-3c}. Similarly to the 4-sum, now $(\mathbf{g}_{i_1} + \mathbf{g}_{i_2}) + (\mathbf{g}_{i_3} + \mathbf{g}_{i_4})$ must sum to zero in the next c positions, with probability 2^{-c}. Finally, we also need the sum of error vectors to be error free mod 2 in $3c$ positions and the probability of this event is again denoted by P_{nf}.

As before, P_{nf} can be bounded by the probability that no errors occur in the first $3c$ positions. The probability of such a distribution for a single error vector is then roughly $((n-3c)/n)^k$ and for all eight of them we have

$$P_{\text{nf}} \geq ((n-3c)/n)^{8k}.$$

Lemma 2. *When $l = 8$, we expect to have*

$$N > \binom{3 \cdot 2^c}{8} \cdot 2^{-m-4c} \cdot 105 \cdot ((n-3c)/n)^{8k}$$

zero sum vectors in the final list in Algorithm 2.

For $l = 8$ there are more errors in general, meaning that P_{nf} is much smaller compared to $l = 4$. This gives a stronger motivation for examining other error patterns such as the double errors canceling out. In particular, the sums from our algorithm can have $1 + 1 = 0$ in the first $3c$ bits. More specifically, if two error vectors have a one in the same position, their combination still survives the Combine step in Algorithm 2. For some parameters proposed by Firekite's designers, certain double error events are even more likely than having no error at all in the first $3c$ positions and thus should not be neglected. Since the error vectors are sparse (e.g., $k = 16 \ll n = 1024$), if a double error occurs at a position, it most likely happens only **once**, i.e, coming from one pair of $\mathbf{g}_{i_j}$'s (or equivalently, $\mathbf{e}'_{i_j}$'s). Having four ones in the same position is exceedingly rare. Therefore, we can safely say that we only have non-repeating double errors. Let us look at the simple case where errors from Firekite have exactly k bits set and we consider the estimation below for P_{nf} as reasonable.[6]

Assume we have $\epsilon \leq k$ double errors, and the probability is denoted by P_ϵ. Then P_ϵ is equal the sum of all possible error patterns/combinations of g_i vectors, provided they result in ϵ collisions. One writes

$$\epsilon = \sum_{i,j>i} \epsilon_{ij},$$

where ϵ_{ij} denotes the number of double errors between $\mathbf{g}_i$ and $\mathbf{g}_j$. The total number of errors in $3 \cdot c$ positions of $\mathbf{g}_i$ is $\epsilon_i = \sum_j \epsilon_{ij}$. Hence

$$P_\epsilon = \sum_{\{(\epsilon_{ij})\}} P_{\epsilon,\{(\epsilon_{ij})\}}.$$

where $\{(\epsilon_{ij})\}$ is an eligible error colliding pattern of $\mathbf{g}_i$ vectors with the corresponding probability $P_{\epsilon,\{(\epsilon_{ij})\}}$.

[6] The expected weight of errors from Firekite can be smaller than k. Hence using binomial expressions, while not entirely correct, gives a good approximation.

Lemma 3. *Let $\mathbf{g}_i, i = 1, \ldots, 8$ be binary vectors. Then, the noise-free probability of $\sum_{i=1}^{8} \mathbf{g}_i$ in the first $3 \cdot c$ bits is:*

$$P_{\text{nf}} = \sum_{\epsilon=0,\ldots,k} P_\epsilon = \sum_{\substack{\epsilon=0,\ldots,k \\ \{(\epsilon_{ij})\}}} P_{\epsilon,\{(\epsilon_{ij})\}}$$

where

$$P_{\epsilon,\{(\epsilon_{ij})\}} \approx \prod_{i=1}^{8} \binom{k}{\epsilon_i} \left(\frac{3c}{n}\right)^{\epsilon_i} \left(\frac{n-3c}{n}\right)^{k-\epsilon_i} \frac{\binom{\epsilon_1}{\epsilon_{1i}}\binom{\epsilon_2-\epsilon_{12}}{\epsilon_{2i}} \cdots \binom{3c-\epsilon_{1i}-\ldots-\epsilon_{i-1i}}{\epsilon_i-\epsilon_{1i}-\ldots\epsilon_{i-1i}}\binom{\epsilon_{i-1}-\epsilon_{1i-1}-\ldots\epsilon_{i-2i-1}}{\epsilon_{i-1i}}}{\binom{3c}{\epsilon_i}}.$$

Proof. Given ϵ double errors and a fixed error colliding pattern $\{(\epsilon_{ij})\}$. Without loss of generality, we further assume that $\epsilon_i \geq \epsilon_j$ for $i < j$, i.e., $\mathbf{g}_1$ has the most errors in the first $3c$ bits. The probability of $\mathbf{g}_i$ having ϵ_i errors in the first $3c$ bits is:

$$\binom{k}{\epsilon_i} \left(\frac{3c}{n}\right)^{\epsilon_i} \left(\frac{n-3c}{n}\right)^{k-\epsilon_i}$$

We also requires $\mathbf{g}_2$ to have ϵ_{12} colliding positions out of ϵ_2. This probability is

$$\frac{\binom{\epsilon_1}{\epsilon_{12}}\binom{3c-\epsilon_{12}}{\epsilon_2-\epsilon_{12}}}{\binom{3c}{\epsilon_2}}.$$

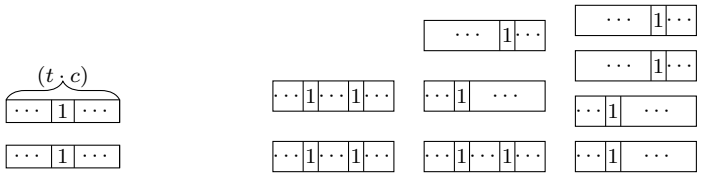
Similarly, for vector $\mathbf{g}_3$, the colliding probability is

$$\frac{\binom{\epsilon_1}{\epsilon_{13}}\binom{\epsilon_2-\epsilon_{12}}{\epsilon_{23}}\binom{3c-\epsilon_{13}-\epsilon_{23}}{\epsilon_3-\epsilon_{13}-\epsilon_{23}}}{\binom{3c}{\epsilon_3}}.$$

Generalizing for $\mathbf{g}_i$ and the lemma follows. □

However, it is not practical to take into account all possible double error events. For instance, if the expected numbers of 1's in the first $t \cdot c$ positions for each error vector is small, e.g., fewer than 2, multiple double errors occur with vanishing probability. It also becomes increasingly hard to deduce all possible colliding patterns and the probabilities for such occurrences are insignificant to, say one or two double error events; hence they do not contribute substantially to our estimate. Moreover, an improvement in P_{nf} suggests that we need less input for Algorithm 2. A reasonable approximation suffices for us to deduce the necessary initial list size $|L^{(0)}|$ so that the expected number of zero sums is $N > 1$.

To illustrate this argument, we consider the case where we have the relative Hamming weight of the error vectors $\mathbf{e}_i$'s in the first $3 \cdot c$ positions to be slightly larger than 2 ($l = 8$). Hence, we focus only on the scenarios of up to 2 double errors in Figure 3.



(a) 1 double error **(b)** 2 double errors

Figure 3: Illustration for the colliding patterns

3.2.3 The random case

In the previous subsections, we saw that if we choose parameters accordingly, we can expect to have zero sum vectors in the final list with non-trivial probability. We now need to check whether low weight sums can stem from random vectors. In other words, the zero sums must be easily distinguished from those coming from the random case.

Assume Algorithm 2 outputs "Firekite" for the random case. This means that it has found a vector in the final list of weight at most c_ω. It is thus of interest to derive the likelihood of such a vector in the random case. Recall $\Delta(\mathbf{g})$ as the Hamming weight of a binary vector $\mathbf{g}$, and let $\mathbf{g}_{[i]}$ be the i-bit truncated $\mathbf{g}$ (first i positions). A vector in the final list will have the first $t \cdot c$ positions all zero, but the remaining positions $d - t \cdot c$ positions are just formed by XOR-ing l random bit values; thus, they are independent and uniformly distributed on $\{0, 1\}$. The probability of such a vector having Hamming weight at most c_ω is

$$\mathbf{Pr}\left[\Delta(\mathbf{g}) \leq c_\omega : \mathbf{g} \in \mathbb{Z}_2^d, \mathbf{g}_{[t\cdot c]} = 0\right] = \frac{\sum_{i=0}^{c_\omega} \binom{d-t\cdot c}{i}}{2^{d-t\cdot c}},$$

and the expected number of vectors of weight at most c_ω, denoted by N_{random}, in the final list is

$$N_{\text{random}} = 3 \cdot 2^c \cdot \frac{\sum_{i=0}^{c_\omega} \binom{d-t\cdot c}{i}}{2^{d-t\cdot c}}.$$

Information theoretically, we have an approximation[7] as

$$N_{\text{random}} = 3{\cdot}2^c{\cdot}\frac{\sum_{i=0}^{c_\omega} \binom{d-t\cdot c}{i}}{2^{d-t\cdot c}} \approx \sum_{i=0}^{c_\omega} 2^{-(1-H(i/(d-t\cdot c)))(d-t\cdot c)+c} \approx 2^{-(1-H(c_\omega/(d-t\cdot c)))(d-t\cdot c)+c},$$

where $H(p) = -p\log(p) - (1-p)\log(1-p)$ with $p \in (0, 1)$. Therefore, if there exists a low weight sum in the final list $L^{(t)}$ and N_{random} is vanishingly small, we have shown that the Firekite's output vectors are indeed not random.

4 Results for the distinguisher

In this section we give the results for our distinguishing attack as described in Section 3 when it is applied on the suggested parameter sets for Firekite.

4.1 Theoretical complexity estimation for the proposed parameters of Firekite

We investigate the results for the proposed parameters case $n = 1024, m = 216, d = 648, k = 16$, where the claimed security level is 82. We justify our choices of the crucial parameter c as follows. One can find the inspiration from Wagner l-tree algorithm [Wag02] in Algorithm 2, namely, by consecutively canceling out c bits. Wagner argued that one needs lists of size $O(2^{\frac{m}{1+\log l}})$ to have a solution in the exact *l-list birthday problem*. In our algorithm, obviously we need a bit more[8], i.e, $O(2^{\frac{m}{1+\log l}+a})$ where a depends on P_{nf}. Note that P_{nf} remains relatively the same if $c \approx \frac{m}{1+\log l}$. Therefore, we initially set $c = \frac{m}{1+\log l}$, then raising until we get $N > 1$. Finally, we verify $N > 1$ again with P_{nf} estimated by said c.

[7] We use $2^n\binom{n}{i} \approx 2^{(1-H(i/n))n}$. The final approximation is due to overwhelming contribution of $2^{-(1-H(c_\omega/(d-t\cdot c)))(d-t\cdot c)+c}$.

[8] Wagner showed that one can find $\alpha^{1+\log l}$ more solutions at the expense of α times more work, provided $\alpha \leq 2^{m/(\log l\cdot(1+\log l))}$ [Wag02].

1. For $l = 4$ we derive the following: We pick $c = 76$, $c_w = \lceil 4 \cdot \hat{k} \rceil = 41$, where $\hat{k} = \frac{k \cdot d}{n}$. Let P_0 denote the probability of the first $2 \cdot c$ position being error free. By simply setting $P_{\text{nf}} \approx P_0 = \left(\frac{n - 2 \cdot c}{n}\right)^{4k} \approx 2^{-14.83}$, we get $N > 1.42$ and

$$N_{\text{random}} \approx 2^{-215.76}.$$

2. For $l = 8$ we derive the following: Picking $c = 62$ and similarly, $c_\omega = \lceil l \cdot k \cdot \frac{d}{n} \rceil = 81$, we have N_{random} very close to zero.

   As an example, we approximate P_{nf} by the sum of probabilities of no double errors P_0, one double errors P_1, and two double errors P_2 ($\epsilon = 0, 1, 2$). As discussed that many double errors are improbable, we focus on the most likely cases.

   - If there is no double error, $P_0 = \left(\frac{n - 3c}{n}\right)^{8k} \approx 2^{-37.017}$.
   - Assume there is one double error occurring. For the colliding pair of vectors, the probability of having exactly one 1 in the same position $\frac{k^2}{3c}\left(\frac{3c}{n}\right)^2\left(\frac{n-3c}{n}\right)^{2(k-1)}$, and there are $\binom{8}{2}$ ways to select a pair/colliding pattern $\{(\epsilon_{ij})\}$, hence

$$P_1 \approx \binom{8}{2}\frac{k^2}{3c}\left(\frac{3c}{n}\right)^2\left(\frac{n-3c}{n}\right)^{2(k-1)}\left(\frac{n-3c}{n}\right)^{6k} \approx 2^{-36.092}.$$

   - If there are two double errors, then there are two cases: the double errors happen in one pair or two pairs (note that a vector in the first pair can appear in the second pair). Let P_{21} and P_{22} denoted such events, respectively, then

$$P_2 \approx P_{21} + P_{22}.$$

   where

$$P_{21} \approx \binom{8}{2}\frac{\binom{k}{2}^2}{\binom{3c}{2}}\left(\frac{3c}{n}\right)^4\left(\frac{n-3c}{n}\right)^{2(k-2)}\left(\frac{n-3c}{n}\right)^{6k}.$$

   and

$$P_{22} \approx \binom{8}{3}\binom{k}{2}\left(\frac{3c}{n}\right)^2\left(\frac{n-3c}{n}\right)^{k-2} \cdot 2 \cdot \left[\frac{k}{3c}\left(\frac{3c}{n}\right)\left(\frac{n-3c}{n}\right)^{(k-1)}\right]^2\left(\frac{n-3c}{n}\right)^{5k} + \binom{8}{2}\binom{6}{2}\left(\frac{k^2}{3c}\right)^2\left(\frac{3c}{n}\right)^4\left(\frac{n-3c}{n}\right)^{4(k-1)}\left(\frac{n-3c}{n}\right)^{4k}.$$

   Therefore, $P_2 \approx 2^{-35.86}$, and $P_{\text{nf}} > P_0 + P_1 + P_2 \approx 2^{-34.65} \approx 4P_0$ which gives $N > 2.7$. The failure probability for this attack when $c_\omega = 81$ can be indicated by

$$N_{\text{random}} \approx 2^{-90.23}.$$

Table 2 shows our attack's complexity and the corresponding N_{random} for a few sets of parameters suggested by the Firekite's designers. Recall that the theoretical complexity is

$$C = t \cdot (3 \cdot 2^c) \cdot (1 + \lfloor d/p \rfloor).$$

In their implementation, beside several optimization flags, they also use a compilation flag `-mavx2` that allows XOR operations to apply on 256 bits per cycle. Therefore, in our complexity estimates, we set $p = 256$.

Table 2: Our distinguishing attack complexity corresponding to a few selected sets of parameters for 80-bit and 128-bit security of Firekite's stream cipher.

Parameters				Attacks(log)		N_{random}(log)	
m	n	k	Security	4-sum	8-sum	4-sum	8-sum
216	1024	16	82.76	80.17	66.755	-215.76	-90.23
216	2048	32	82.76	81.17	67.755	-765.79	-465.74
216	16384	216	80.68	83.28	68.87	-9011.62	-6541.71
352	2048	32	129.07	130.16	106.75	-541.407	-275.94
352	4096	58	128.95	129.16	105.75	-1742.23	-1150.69
352	16384	228	128.93	131.257	107.84	-8510.19	-6023.39

With 4-sum attacks and for small parameters of 80-bit secured Firekite, we can only refine Firekite's designer measurements marginally. However, our 8-sum distinguisher manages to break Firekite for all parameters, except for the smallest 128-bit secure instance, which is $n = 1024, m = 352,$ and $k = 16$. In particular, we can find a zero sum with the cost $2^{107.75}$ but $N_{\text{random}} \approx 83$. Therefore, we were unable to claim that the Firekite's output is not randomly distributed as the low weight sums found could easily come from random vectors. The explanation is that $d = n - m - k \log n$ is not so large compared to $8 \cdot \hat{k}$ in this case; hence, it is impossible to distinguish from the case of random vectors $\mathbf{g}_i$. In general, 8-sum attack performs slightly better when the parameters n and k grows (with the same factor, as suggested by Firekite's designers). This is owing to the fact that d grows bigger while m remains relatively unchanged; hence we have even smaller failure probability and bigger error free probability P_{nf}. In fact, we need a smaller initial list ($3 \cdot 2^{60}$ compared to $3 \cdot 2^{62}$) when attacking Firekite instance with $n = 16384, m = 216, k = 216$.

These theoretical results above can be improved; larger Firekite parameters make the double error events more probable. For instance, attacking the parameters $n = 16384, m = 352, k = 228$ with 8-sum distinguisher, we find that two double errors (P_2) are twice as likely as no error (P_0). Therefore, P_{nf} should be better approximated by taking, e.g., P_3 and P_4 into consideration.

4.2 Simulation results for smaller parameters

We verify our approach and formulas by performing simulations. As a toy example, we set up a mini version of Firekite with small parameters and run Algorithm 2. Our parameters are $m = 52, n = 256, k = 4, b = 269$, and $r = 15$. Recall that b is the secret key's length used to generate the first row of Firekite's secret matrix $\mathbf{M}$ such that $(X^b - 1)/(X - 1)$ is irreducible in $\mathbb{Z}_2[x]$ and r is the number of randomization rounds before Firekite generates its actual output.

- For the 4-sum distinguisher, the filter weight is $c_\omega = 11$. The parameter c is chosen so that
$$N = \binom{3 \cdot 2^c}{4} \cdot 2^{-m} \cdot 3 \cdot 2^{-c} \cdot P_{\text{nf}} > 1.$$
One has $P_0 = \left(\frac{n - 2c}{n}\right)^{4k} \approx 2^{-14.83}$. If we choose $c = 18$, i.e $|L^{(0)}| = 3 \cdot 2^{18}$, there are, on average, less than 1 bit set of the error vectors in the first $2c$ bits. One can safely assume $P_{\text{nf}} \approx P_0$, and it gives $N \approx 3.6$. The simulation returns $1 \sim 3$ low weight vectors consistently. The discrepancy can be explained as follows: The assumption that we can keep the list's size $|L^{(i)}| = 3.2^c$ is often violated as there are **more** vectors after every COMBINE step owing to vectors being not evenly distributed among buckets. Therefore, "good" combinations that present in zero sums might be

discarded by chance. Keeping all combinations from COMBINE, we obtain more low weight sums after filtering with c_ω (obviously, at the cost of higher complexity) and the simulation is more consistent with the theoretical estimate. We now look at the probability of 4 random vectors summing to such sums:

$$\mathbf{Pr}\left[\Delta(\mathbf{g}) \leq c_\omega = 11 : \mathbf{g} \in L^{(2)}, \mathbf{g}_{[2\cdot 18]} = 0\right] = \frac{\sum_{i=0}^{c_\omega} \binom{d-2\cdot c}{i}}{2^{d-2\cdot c}} \approx 2^{-83.757},$$

therefore,

$$N_{\text{random}} \approx 2^{-64.172}.$$

- For the 8-sum distinguisher, the filter weight is chosen to be $c_\omega = 21$. Again, c must fulfill

$$N = \binom{3\cdot 2^c}{8} \cdot 2^{-m-4c} \cdot 105 \cdot P_{\text{nf}} > 1,$$

where $P_{\text{nf}} \approx P_0 + P_1 + P_2 \approx 2^{-7.67}$. Setting $c = 14$, meaning $|L^{(0)}| = 3 \cdot 2^{14}$, suffices and gives $N > 1.35$. It needs to be clarified that in the 8-sum attack's implementation, the effect of keeping $|L^{(i)}| = 3 \cdot 2^{14}$ is more visible. In particular, we might discard all "good combinations" when N is very close to 1. We adapt by allowing $|L^{(1)}|$ and $|L^{(2)}|$ to be at most $2 \cdot |L^{(0)}|$, then directly filter combinations from $|L^{(2)}|$ with c_ω. Therefore, the complexity is slightly higher than the theoretical estimate provided in the previous section. We suppose said negative impact can be mitigated when c is large as the vectors in $|L^{(i)}|$ might be more evenly distributed among buckets. The simulation returns around 1 low weight vector on average.

In the random case, the probability of having a vector having Hamming weight up to c_ω is:

$$\mathbf{Pr}\left[\Delta(\mathbf{g}) \leq c_\omega = 21 : \mathbf{g} \in L^{(3)}, \mathbf{g}_{[3\cdot 14]} = 0\right] = \frac{\sum_{i=0}^{c_\omega} \binom{d-3\cdot c}{i}}{2^{d-3\cdot c}} \approx 2^{-50.162},$$

which yields

$$N_{\text{random}} \approx 2^{-34.577}.$$

5 A key-recovery attack on Firekite

In this section we show a possible way to turn the distinguishing attack into a key-recovery attack with a bit higher complexity. We focus on the recovery of the secret matrix $\mathbf{M}'$. First, we recall that the secret matrix $\mathbf{M}$ in Firekite is constructed by choosing a part of the bigger a $b \times b$ matrix $\mathbf{Q}$ as described in Section 2. We pick $\mathbf{q}_1 \xleftarrow{U} \mathbb{Z}_2^b$ and defined rows in the matrix $\mathbf{Q}$ as $\mathbf{q}_i = X^{i-1}\mathbf{q}_1, i = 1, ...b$, i.e., by shifting the first column to the left consecutively $b-1$ times. The secret matrix $\mathbf{M}$ is obtained by dropping the last $b-n$ columns of $\mathbf{Q}$ and keeping only the first m rows. The secret key is only the random b-bit vector $\mathbf{q}_1$ rather than a $m \times n$ matrix $\mathbf{M}$. Let the unknown bits in $\mathbf{q}_1$ be written as $\mathbf{q}_1 = (k_1, k_2, \ldots, k_b)$.

More specifically, we can now see that if $\mathbf{M} = [m_{ij}]_{m\times n}$ then every entry in the matrix $\mathbf{M}$ corresponds to an unknown key bit, i.e.,

$$m_{ij} = k_{i_{ij}},$$

where i_{ij} is a known index. As $\mathbf{M}'$ is the first part of $\mathbf{M}$, the same holds for $\mathbf{M}'$.

Next, we note that $\mathbf{M}'$ as a generator matrix is spanning a code $\mathcal{C}$. But there are many generator matrices spanning the same code. One particular case is when $\mathbf{M}'$ is transformed to a systematic generator matrix, that is a matrix of the form $\mathbf{M}'' = [\mathbf{I} \mid \mathbf{J}]$, where $\mathbf{I}$

is the $m \times m$ identity matrix. We assume $\mathcal{C} = \{\mathbf{v}\mathbf{M}', \mathbf{v} \in \mathbb{Z}_2^m\} = \{\mathbf{v}[\mathbf{I} \mid \mathbf{J}], \mathbf{v} \in \mathbb{Z}_2^m\}$. Clearly, $\mathbf{M}'' = \mathbf{S}\mathbf{M}'$ for some $m \times m$ unknown matrix $\mathbf{S}$. In this case, entries in $\mathbf{J}$ can no longer be linearly expressed from unknown key bits. Therefore, we consider all entries in $\mathbf{J}$ as unknown. There is an assumption here that the fist m columns of $\mathbf{M}'$ are linearly independent, which is adopted.

The key-recovery attack consists in running the aforementioned distinguisher, finding several zero sum vectors, and then deducing $\mathbf{M}'$. We first show how to derive the secret key from such zero sum vectors if we assume that the first m positions are all error free mod 2. Again, a zero sum vector fulfills

$$\sum_{j=1}^{l} \mathbf{g}_{i_j} = \sum_{j=1}^{l} \mathbf{v}_{i_j}\mathbf{M}' + \sum_{j=1}^{l} \mathbf{e}_{i_j} = \sum_{j=1}^{l} \mathbf{e}_{i_j}.$$

Therefore, finding a zero sum amounts to knowing the corresponding $\sum_{j=1}^{l} \mathbf{e}_{i_j}$. We now consider a single $\mathbf{g}_{i_j}$ vector. Its positions can be split in two parts, namely those for which we know that they are (most likely) error free mod 2 (since the error vector is zero in this position) and those for which we do not have knowledge of, since one of the l involved vectors has an error. Note that the first $t \cdot c$ positions are error free, most other positions are as well, but there will be roughly $l \cdot \hat{k}$ positions where at least one of the eight vectors will have an error.

Assuming that the first m positions are all of the error free type, one can write

$$\mathbf{g}_{i_j} = \mathbf{v}_{i_j}[\mathbf{I} \mid \mathbf{J}] + \mathbf{e}_{i_j}.$$

We further have roughly $d - t \cdot c - l \cdot \hat{k}$ additional positions to be error free. For each such position, we can form a linear equation. Denote by $\mathbf{J}_q$ by the q-th column of $\mathbf{J}$. Assume a position $q > m$ is error free. Then

$$\mathbf{g}_{i_j}(q) = \sum_{i=1}^{m} \mathbf{v}_{i_j}(i)\mathbf{J}_q(i).$$

Here $\mathbf{g}_{i_j}(q)$ denotes position q in vector $\mathbf{g}_{i_j}$, etc. Since $\mathbf{g}_{i_j}(q)$ and $\mathbf{v}_{i_j}$ are known, it gives a linear equation in the unknowns of vector $\mathbf{J}_q$. Collecting many such equations will enable us to derive $\mathbf{J}_q$ and eventually, the full $\mathbf{J}$ matrix. However, this will not work in practice if the probability of double errors as discussed above are not negligible. Hence, we need to consider a more complicated approach where we try to detect columns with double errors.

Assuming now that we have found several zero sum vectors with the 8-sum distinguisher, we put them in a matrix and examine its structure. Let the first seven vectors in the first zero sum vector $\sum_{j=1}^{l} \mathbf{g}_{i_j}$ be denoted $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_7$, the first seven vectors in the next zero sum vector be denoted $\mathbf{g}_8, \mathbf{g}_9, \ldots, \mathbf{g}_{14}$, etc. We now form a matrix $\mathbf{G}$ consisting of P such vectors of the form

$$\mathbf{G} = \begin{pmatrix} \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_7 \\ \mathbf{g}_8 \\ \vdots \\ \mathbf{g}_P \end{pmatrix}.$$

Now we will examine the columns and the related known error vector for the corresponding zero sum vector. The first $t \cdot c$ columns are all without error, from the process. For the remaining columns, we check each one of them to see if $\sum_{j=1}^{l} \mathbf{g}_{i_j}$ is zero in this position in all the zero sum vectors that we use. If this is the case, then the column is

free of direct errors and we keep it. If it is not true, then we discard the column. The probability that a single column will be kept is computed as in Subsection 3.2.2 After this process, we have a new matrix $\mathbf{G}'$ of length $t \cdot c + U$, where U is the number of columns kept in the previous step. If $t \cdot c + U > P > m$ there will be low-weight codewords in the code spanned by $\mathbf{G}'$. Namely, if $P > m$ then we have linear combinations of rows that correspond to summing to a zero codeword plus error terms. But from the previous step we removed all columns for which there is a visible direct single error contribution, so the only error contribution in the code comes from double errors. Each double error will give either a 0 or a 1 as contribution in that position. But double errors are much less common. If we assume that we have D double errors in the columns in $\mathbf{G}'$, then we expect to have codewords in the code spanned by $\mathbf{G}'$ that have weight around $D/2$. Also, because we can form 2^{P-m} different combinations of rows that sum to zero in the underlying code, we will have 2^{P-m} low weight codewords of weight around $D/2$. Finally, every column in $\mathbf{G}'$ can be found to contain a double error or not. If this position is zero in all (or almost all) low-weight codewords, there is no double error. Otherwise, we detected a double error in that position. From this information, it is possible to do a full recovery through additional steps.

One problem is to find enough columns free of direct errors. When the length is small, there will be very few columns free of direct errors and the length of $\mathbf{G}'$ is not enough to have low-weight codewords.

5.1 Explaining the key recovery attack on Firekite with n=16384

Consider the Firekite parameters choice $m = 216$, $n = 16384$ and $k = 216$. The attack works as follows. First we run the 8-sum distinguisher to obtain zero sum vectors. In this case, we need to have slight more than $m/7$, so we choose to find 32 zero sum vectors. Using our previous formulas we choose $c = 60$ generating $N \approx 3.5$ zero sum vectors using the 8-sum distinguisher. Instead of repeating the distinguishing attacks 32 times or, equivalently 2^5 more work, we can instead increase the initial list size to $3 \cdot 2^{61}$ to obtain enough low-weight sum ($N \approx 41$), with an affordable complexity of $C_{\text{distinguishing}} = 2^{69.87}$.

We now consider a matrix $\mathbf{G}$ of dimension $32{\cdot}7$ consisting of the $\mathbf{g}_i$ vectors as its rows and we then remove columns with direct associated errors. In a zero sum vector, there are $216{\cdot}8$ errors inserted, so a positon is error free with probability $(1-1/(16384-61{\cdot}3))^{216{\cdot}8} \approx 0.899$. In our case, we want the position to be error free in all 32 zero sum vectors, which brings the probability to about 0.033. Since $d = 13144$, we can expect to have about 432 columns error free. So we form the matrix $\mathbf{G}'$ which is of dimension $P = 224$ and length $189 + 432 = 621$. There will be $2^{224-216} = 2^8$ codewords in the code spanned by $\mathbf{G}'$ with support corresponding to the double errors.

By computing the likelihood of double errors we find that a column in $\mathbf{G}'$ is error free with probability at most $0.995^{32} = 0.85$. For instance, consider a simple case where there is a non-repeating double error at position j-th of a zero sum, then the probability is

$$1 - \binom{8}{2} \cdot \left(1 - \left(\frac{16383}{16384}\right)^{216}\right)^2 \cdot \left(\frac{16383}{16384}\right)^{216 \cdot 6} \approx 0.995$$

One can expect $621 \cdot 0.15 = 93$ columns to have double errors. In conclusion, the code spanned by $\mathbf{G}'$ will contain 2^8 codewords where the weight is distributed around 47. Finding low-weight codewords in a random binary linear code is a well-known problem that has been studied extensively. One can use ISD algorithms to complete the task. For our example, an improved Stern's ISD algorithm [9] yields the bit-complexity estimate, denoted

[9] The estimate is obtained in a recent work by Andre Esser and Emanuele Bellini [EB21], where they unify ISD-algorithm variants (Prange,Stern,MMT,BJMM) in a *Nearest-Neighbor* framework. They also provided a complexity estimator for independent parameters.

C_{ISD}, to be $2^{44.6}$, which is small compared to the distinguishing step.

A random linear code with dimension 224 and length 621 will have an expected minimum distance of about 100 according to the Varshamov-Gilbert bound, so the low weight codewords would come from the observation above. Finally, generating say 16 such low weight codewords, we look for the positions where all the 16 of these codewords are zero. This would be the case for more than 500 positions and in this way we have identified 500 columns that are completely error free. Using a selection of them as the information set of the code we can now easily recover remaining parts of the code $\mathbf{M}'$. The total complexity is therefore

$$C = C_{\text{distinguishing}} + C_{\text{ISD}} \approx 2^{69.87} + 16 \cdot 2^{44.6} \approx 2^{69.87}.$$

6 Discussion and Conclusions

Having seen how Firekite is vulnerable to our distinguisher, especially the 8-sum distinguishing attack, it is natural to ask how we can make Firekite and other similar ciphers resilient to a generic birthday problem solving algorithm. From the result and performance of our attacks, there are certain approaches one can consider. First, we observed that N_{random}, or in other words, the failure probability inflates when the filtering weight c_ω grows. That is to say, unless c_ω is very small compared to d, it is difficult to distinguish Firekite's zero sum vectors from those that could stem from random vectors $\mathbf{g}_i$. Therefore, instantiating Firekite with larger k can be beneficial. Second, we have discussed that the attack complexity depends on the parameter c which is solely determined by m (if we fix l), the number of rows in $\mathbf{M}$. Therefore, if the security level is close to $m/(1 + \log l)$, our attack becomes infeasible. As a contribution to Firekite's design criteria, we propose a few modifications as follows.

1. For small Firekite's parameters, one can increase k slightly which yields an LPN instance with a higher noise rate; therefore more difficult to solve in general. In our estimate, larger k suggests a drastic decrease in P_{nf} and an increase in N_{random}. Clearly, it is now exceedingly unlikely to have no error in the first $t \cdot c$ bits and d becomes smaller which makes it more difficult to distinguish the zero sum found by Algorithm 2 from those stemming from the random case. As an example, by setting $k = 24$ for the instance $n = 1024, m = 216$, our attack was rendered vain as N_{random} is always larger than 1. This comes at the cost of decreasing the number of bits encrypted per invocation; hence more instructions need to be executed *per bit*. However, larger parameter instances of Firekite are less affected by this "fix" as d becomes large relatively to k. For instance, our 8-sum attack still succeeds with $n = 16384, m = 216$ despite raising k from its original $k = 216$ to $k = 400$. We only need to slightly increase $|L^{(0)}| = 3 \cdot 2^{67}$ and we still obtain a good good failure probability as $N_{\text{random}} \approx 2^{-2835}$. An extreme adjustment such as $k = 600$ gives Firekite resistance to our attack. It is also worth noting that the likelihood of double error events become much more prevalent with this modification and will affect the proposed key recovery attack negatively.

2. Having discussed that the initial list $L^{(0)}$ size requirement depends directly on m, we therefore can adapt Firekite with a bigger m, equivalently using more rows from the matrix $\mathbf{Q}$. It also effectively decreases d, hence a larger failure probability. However, as in Table 1, the estimated security level (by the **LF2** BKW algorithm) should grow correspondingly. Therefore, our attacks still serve as better security measurements for Firekite. Again, a larger matrix $\mathbf{M}$ worsens the Firekite performance in terms of a more costly matrix-vector multiplications and fewer bits encrypted per invocation.

Finally, we may discuss possible future improvements to the proposed attack. We believe that there can be a possibility to gain some small amount in terms of decreased complexity by smaller changes in the distinguishing algorithm. One idea could be to not only allow sums of vectors that sum to zero in c positions, but also those that have weight 1 in these c positions. Still, it would not change the complexity significantly and with modified parameters as suggested above the Firekite should meet the intended security level.

References

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *Annual International Cryptology Conference*, pages 595–618. Springer, 2009.

[Ber09] Daniel J. Bernstein. *Introduction to post-quantum cryptography*, pages 1–14. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[BFKL93] Avrim Blum, Merrick Furst, Michael Kearns, and Richard J Lipton. Cryptographic primitives based on hard learning problems. In *Annual International Cryptology Conference*, pages 278–291. Springer, 1993.

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in 2 n/20: How 1+ 1= 0 improves information set decoding. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 520–536. Springer, 2012.

[BKLV21] Sonia Bogos, Dario Korolija, Thomas Locher, and Serge Vaudenay. Towards efficient lpn-based symmetric encryption. In *International Conference on Applied Cryptography and Network Security*, pages 208–230. Springer, 2021.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM (JACM)*, 50(4):506–519, 2003.

[BLP11] Daniel J Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: ball-collision decoding. In *Annual Cryptology Conference*, pages 743–760. Springer, 2011.

[BM17] Leif Both and Alexander May. The approximate k-list problem. *IACR Transactions on Symmetric Cryptology*, pages 380–397, 2017.

[BMVT78] Elwyn Berlekamp, Robert McEliece, and Henk Van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[BSH06] Johann Barbier, Guillaume Sicot, and Sébastien Houcke. Algebraic approach for the reconstruction of linear and convolutional error correcting codes. *International Journal of Applied Mathematics and Computer Science*, 2(3):113–118, 2006.

[BV15] Sonia Bogos and Serge Vaudenay. How to sequentialize independent parallel attacks? In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 704–731. Springer, 2015.

[CF09] Mathieu Cluzeau and Matthieu Finiasz. Recovering a code's length and synchronization from a noisy intercepted bitstream. In *2009 IEEE International Symposium on Information Theory*, pages 2737–2741. IEEE, 2009.

[CT19] Kevin Carrier and Jean-Pierre Tillich. Identifying an unknown code by partial gaussian elimination. *Designs, Codes and Cryptography*, 87(2):685–713, 2019.

[DKPW12] Yevgeniy Dodis, Eike Kiltz, Krzysztof Pietrzak, and Daniel Wichs. Message authentication, revisited. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 355–374. Springer, 2012.

[DV13] Alexandre Duc and Serge Vaudenay. Helen: A public-key cryptosystem based on the lpn and the decisional minimal distance problems. In *International Conference on Cryptology in Africa*, pages 107–126. Springer, 2013.

[EB21] Andre Esser and Emanuele Bellini. Syndrome decoding estimator. *IACR Cryptol. ePrint Arch.*, 2021:1243, 2021.

[GJL14] Qian Guo, Thomas Johansson, and Carl Löndahl. Solving lpn using covering codes. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 1–20. Springer, 2014.

[GRS08a] Henri Gilbert, Matthew JB Robshaw, and Yannick Seurin. : Increasing the security and efficiency of. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 361–378. Springer, 2008.

[GRS08b] Henri Gilbert, Matthew JB Robshaw, and Yannick Seurin. How to encrypt with the lpn problem. In *International Colloquium on Automata, Languages, and Programming*, pages 679–690. Springer, 2008.

[HB01] Nicholas J Hopper and Manuel Blum. Secure human identification protocols. In *International conference on the theory and application of cryptology and information security*, pages 52–66. Springer, 2001.

[HDWH12] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In *21st {USENIX} Security Symposium ({USENIX} Security 12)*, pages 205–220, 2012.

[HKL+07] Stefan Heyse, Eike Kiltz, Vadim Lyubashesvky, Christof Paar, and Krzysztof Pietrzak. An efficient authentication protocol based on ring-lpn. In *ECRYPT Workshop on Lightweight Cryptography*, volume 2011. Citeseer, 2007.

[HKL+12] Stefan Heyse, Eike Kiltz, Vadim Lyubashevsky, Christof Paar, and Krzysztof Pietrzak. Lapin: An efficient authentication protocol based on ring-lpn. In *International Workshop on Fast Software Encryption*, pages 346–365. Springer, 2012.

[IL90] Russell Impagliazzo and Levin LA. No better ways to generate hard np instances than picking uniformly at random. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 812–821. IEEE, 1990.

[JW05] Ari Juels and Stephen A Weis. Authenticating pervasive devices with human protocols. In *Annual international cryptology conference*, pages 293–308. Springer, 2005.

[Kir11] Paul Kirchner. Improved generalized birthday attack. *IACR Cryptol. ePrint Arch.*, 2011:377, 2011.

[KPV+17] Eike Kiltz, Krzysztof Pietrzak, Daniele Venturi, David Cash, and Abhishek Jain. Efficient authentication from hard learning problems. *Journal of Cryptology*, 30(4):1238–1275, 2017.

[KS06] Jonathan Katz and Ji Sun Shin. Parallel and concurrent security of the hb and hb+ protocols. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 73–87. Springer, 2006.

[KSS10] Jonathan Katz, Ji Sun Shin, and Adam Smith. Parallel and concurrent security of the hb and hb+ protocols. *Journal of cryptology*, 23(3):402–421, 2010.

[LB88] Pil Joong Lee and Ernest F Brickell. An observation on the security of mceliece's public-key cryptosystem. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 275–280. Springer, 1988.

[Leo88] Jeffrey S Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354–1359, 1988.

[LF06] Éric Levieil and Pierre-Alain Fouque. An improved lpn algorithm. In *International conference on security and cryptography for networks*, pages 348–359. Springer, 2006.

[MGB12] Mélanie Marazin, Roland Gautier, and Gilles Burel. Algebraic method for blind recovery of punctured convolutional encoders from an erroneous bitstream. *IET Signal Processing*, 6(2):122–131, 2012.

[Pra62] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

[SHB09] Guillaume Sicot, Sebastien Houcke, and Johann Barbier. Blind detection of interleaver parameters. *Signal Processing*, 89(4):450–462, 2009.

[Sho99] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[Ste93] Jacques Stern. A new identification scheme based on syndrome decoding. In *Annual International Cryptology Conference*, pages 13–21. Springer, 1993.

[Tix15] Audrey Tixier. Blind identification of an unknown interleaved convolutional code. In *2015 IEEE International Symposium on Information Theory (ISIT)*, pages 71–75. IEEE, 2015.

[Wag02] David Wagner. A generalized birthday problem. In *Annual International Cryptology Conference*, pages 288–304. Springer, 2002.

[YZ16] Yu Yu and Jiang Zhang. Cryptography with auxiliary input and trapdoor from constant-noise lpn. In *Annual International Cryptology Conference*, pages 214–243. Springer, 2016.