Towards Efficient LPN-Based Symmetric Encryption

Sonia Bogos[1], Dario Korolija[2], Thomas Locher[3], and Serge Vaudenay[4]

[1] ELCA, Lausanne, Switzerland
sonia.duc@elca.ch
[2] ETH, Zurich, Switzerland
dario.korolija@inf.ethz.ch
[3] ABB Research, Switzerland
thomas.locher@ch.abb.com
[4] EPFL, Lausanne, Switzerland
serge.vaudenay@epfl.ch

Abstract. Due to the rapidly growing number of devices that need to communicate securely, there is still significant interest in the development of efficient encryption schemes. It is important to maintain a portfolio of different constructions in order to enable a quick transition if a novel attack breaks a construction currently in use. A promising approach is to construct encryption schemes based on the learning parity with noise (LPN) problem as these schemes can typically be implemented fairly efficiently using mainly "exclusive or" (XOR) operations. Most LPN-based schemes in the literature are asymmetric, and there is no practical evaluation of any LPN-based symmetric encryption scheme.
In this paper, we propose a novel LPN-based symmetric encryption scheme that is more efficient than related schemes. Apart from analyzing our scheme theoretically, we provide the first practical evaluation of a symmetric LPN-based scheme, including a study of its performance in terms of attainable throughput depending on the selected parameters. As the encryption scheme lends itself to an implementation in hardware, we further evaluate it on a low-end SoC FPGA. The measurement results attest that our encryption scheme achieves high performance rates in terms of throughput on such hardware, providing evidence that symmetric encryption schemes based on hard learning problems may be constructed that can compete with state-of-the-art encryption schemes.

Keywords: Symmetric encryption · learning parity with noise · LPN · FPGA implementation

1 Introduction

There has been a substantial amount of work in recent years on the development of encryption schemes whose security relies on the hardness of solving a difficult *learning problem*. In fact, there is a strong connection between cryptography and learning problems as shown in the seminal work by Impagliazzo and Levin

who quite generally proved that cryptography is only possible if and only if efficient learning is not [20]. A well studied example of such a learning problem is *learning parity with noise* (LPN). In short, it is the problem of identifying an unknown binary vector given only noisy scalar products of this vector and other vectors chosen uniformly at random. Since computations are carried out in $\mathbb{Z}_2$, the scalar product is a single bit. The scalar products are noisy in the sense that the resulting bit is flipped with a certain probability.[5]

Building encryption schemes based on the LPN problem is appealing because they are expected to achieve decent throughput measured in the number of bytes processed per second. In recent years, several encryption schemes based on LPN have been proposed; however, barring a few exceptions, the focus has been primarily on asymmetric cryptography (see §6 for details on related work). All proposed schemes have in common that they require fresh, cryptographically strong random bits for the encryption of each piece of plaintext. This requirement can be disadvantageous for multiple reasons: First, the generation of secure random bits itself may be computationally expensive. Second, low-power embedded devices often have a limited number of entropy sources, which makes it challenging to produce random bits [17]. Moreover, extracting randomness from sources with low entropy incurs a significant computational overhead [29]. Finally, if each invocation of the encryption process requires fresh randomness, this additional randomness must either be appended to the encrypted data or it is embedded in it. In the former case, the space complexity (or bandwidth requirement) increases and in the latter case, we get a more complex decryption routine and thus a lower throughput for decryption.

We propose a *synchronous stream cipher*, called *Firekite*, which uses an LPN-based pseudo-random number generator (PRNG) with a simple structure [3] to generate its keystream. The PRNG used in Firekite differs from the PRNG introduced by Blum et al. [3] in that it handles the noise vector differently and uses a different noise distribution. Firekite further uses a Ring-LPN hardness assumption to reduce the key size. Unlike other proposed encryption schemes based on the LPN problem, our scheme only requires a source of cryptographically strong random bits for key generation but not for encryption. We further discuss how to use our scheme in practice by proposing concrete sets of parameters to instantiate it. As an additional contribution, the attainable throughput based on our software implementation is measured for various parameter sets. While the potential for efficient LPN-based cryptography has been noted before, this is the first work that measures the actual performance of such an encryption scheme. Moreover, as our scheme can greatly benefit from dedicated hardware, we implemented and evaluated it on a low-power field-programmable gate array (FPGA). To the best of our knowledge, this is also the first time any LPN-based encryption scheme is tested and evaluated on an FPGA.

Although it is unlikely that current symmetric encryption schemes such as AES, which is nowadays employed ubiquitously, will be broken in the near future,

[5] Note that the problem is not difficult to solve without any noise, using Gaussian elimination.

we believe that a portfolio of cryptographic constructions must be at our disposal for multiple reasons. First, fallback solutions are required in case one construction is broken. Second, encryption is used in devices with different capabilities and constraints in a multitude of scenarios, which entails that different schemes are appropriate depending on the use case. LPN-based cryptography may prove to be a sensible approach for specific use cases.

The paper is structured as follows. Background on LPN is provided in §2. Our encryption scheme is introduced in §3 and analyzed in terms of security and performance in §4 and §5, respectively. A summary of related work is provided in §6, and §7 concludes the paper.

2 Background

LPN is a well studied problem in cryptography and machine learning. It is appealing in cryptography as it is a strong candidate for post-quantum cryptography: while efficient algorithms for quantum computers have been found to solve the factorization and the discrete logarithm problem [28], which are the foundation of asymmetric encryption schemes used in practice, no efficient quantum algorithm that solves the LPN problem is known. Moreover, the LPN problem is a promising candidate because only simple operations such as "exclusive or" (XOR) and scalar products are required, which can be implemented efficiently.

Informally, the LPN problem is asking to solve a noisy binary system of equations. We will now provide the formal definition, which uses an LPN *oracle*. Let $x \xleftarrow{U} \mathcal{X}$ denote the event that x is drawn uniformly at random from the domain $\mathcal{X}$.

Definition 1 (LPN Oracle). *Let $s \xleftarrow{U} \mathbb{Z}_2^n$ and Ber_τ be the Bernoulli distribution with a noise parameter $\tau \in]0, \frac{1}{2}[$. Let $D_{s,\tau}$ further denote the distribution defined as*

$$\{(u, c) \mid u \xleftarrow{U} \mathbb{Z}_2^n, c = u^T s + d, d \leftarrow \mathsf{Ber}_\tau\} \in \mathbb{Z}_2^{n+1}.$$

An LPN oracle $\mathcal{O}_{s,\tau}^{\mathsf{LPN}}$ outputs independent random samples according to $D_{s,\tau}$.

Given this definition, we are now in the position to define the LPN problem and the notion of an LPN solving algorithm.

Definition 2 (Search LPN). *Given access to an LPN oracle $\mathcal{O}_{s,\tau}^{\mathsf{LPN}}$, find the vector s. Let $n' \leq n$. We say that an algorithm $\mathcal{M}$ (q, t, μ, θ, n')-solves the search LPN problem, where the secret has size n and the noise parameter is τ, if*

$$\Pr[\mathcal{M}^{\mathcal{O}_{s,\tau}^{\mathsf{LPN}}}(1^n) = (s_1 \dots s_{n'}) \mid s \xleftarrow{U} \mathbb{Z}_2^n] \geq \theta,$$

$\mathcal{M}$ runs in time t, uses memory μ, and queries the LPN oracle at most q times.

In the decisional LPN problem, the objective is to distinguish between output from the LPN oracle and uniformly distributed random vectors of size $n + 1$.

It has been proven that the search LPN and the decisional LPN problem are equivalent [3, 22].

Looking at the history of solving LPN, the first specific LPN solving algorithm is BKW [4], which recovers the LPN secret bit by bit. An important improvement was presented by Levieil and Fouque [25] whose algorithm uses the Walsh-Hadamard transform, which makes it possible to recover several bits of the secret at once and requires fewer initial queries. The use of covering codes, introduced by Guo et al. [16], further improves the performance of LPN solving algorithms [5, 32]. An analysis of the best LPN solving algorithms for a wide range of parameters can be found in the work of Bogos and Vaudenay [5].

All these algorithms are characterized by the fact that their time complexity is sub-exponential, $2^{O\left(\frac{n}{\log(n)}\right)}$, and they require a sub-exponential number of queries, $2^{O\left(\frac{n}{\log(n)}\right)}$, when the noise parameter τ is constant.[6]

Trading off the time complexity in favor of the number of queries, Lyubashevsky [26] uses BKW as a black box and adds a processing phase at the beginning of the LPN solving algorithm. With this modification, the LPN problem can be solved requiring only a polynomial number of queries ($n^{1+\varepsilon}$ for a constant $\varepsilon > 0$) at the expense of an increased time complexity of $2^{O\left(\frac{n}{\log(\log(n))}\right)}$.

3 PRNG and Firekite Construction

We first provide an overview of the LPN-based PRNG construction in §3.1. It is important to note that the basic construction has been proposed before [3]. However, the level of detail provided in §3.1 should help the reader to better understand how our PRNG differs from the general construction. Our PRNG is introduced formally and in detail in §3.2. Subsequently, our encryption scheme *Firekite*, which is based on this PRNG, is presented in §3.3.

3.1 Overview

The challenge of the LPN problem, as defined in §2, is to distinguish between a source providing either random bit vectors of length $n+1$ or vectors containing n random bits plus a single bit that is the noisy scalar product of these n bits and a secret vector s of length n. This definition can naturally be extended to matrices and vectors: the noisy scalar product of a secret $m \times n$-matrix M and a random vector v, i.e., $M^T v + e$, where e denotes a sparse n-bit noise vector, is hard to distinguish from a random n-bit vector. Note that addition is carried out in $\mathbb{Z}_2$, i.e., addition corresponds to computing the XOR of the inputs.

The PRNG construction exploits the fact that noisy matrix-vector products are indistinguishable from random vectors. In order to solve the problem that LPN-based constructions require a source of randomness, we proceed as follows:

[6] A simple guessing strategy has a complexity of $O\left(n^3 e^{n^{1-\delta}}\right)$ using $O\left(n e^{n^{1-\delta}}\right)$ queries if $\tau = n^{-\delta}$.

$$\left(\begin{array}{c} \\ M \\ \\ \end{array}\right)^T \cdot \left(\begin{array}{c} \\ v \\ \\ \end{array}\right) + \left(\begin{array}{c} \\ e \\ \\ \end{array}\right) = \left(\begin{array}{c} g \\ \hline v' \\ \hline c_{e'} \end{array}\right)$$

Fig. 1: The output of the noisy matrix-vector product is split into three components g, v', and $c_{e'}$. The vector g is the output of the PRNG, whereas v' and $c_{e'}$ are used iteratively to compute the next noisy matrix-vector product.

rather than having a separate mechanism, the output of a noisy matrix-vector product is used iteratively as a source of randomness for the next noisy matrix-vector multiplication. Concretely, given a secret matrix M and secret initial vector v and e, the vector $M^T v + e$ can be used to generate the input for the next iteration.

Obviously, simply iterating this process alone does not yield a PRNG because there is no output. This problem is addressed as follows. If $n \gg m$, the noisy matrix-vector product of length n can be split into three pieces: m bits are used as the next vector v', some bits are interpreted as a compact encoding $c_{e'}$ of the next noise vector e', and the remaining bits, denoted by g, constitute the output.

While the length of the noise vector is n, a concise representation of the noise vector e' is possible because e' is sparse. Formally, $\mathcal{H}(e')$ bits are needed to encode e', where $\mathcal{H}(e')$ denotes the entropy of the bit string e' [3]. This process is depicted in Figure 1.

3.2 PRNG

Having a basic understanding of the general PRNG construction, we proceed by giving a formal specification of our PRNG. The *state* of the PRNG is the pair (M, w), where M is a binary $m \times n$-matrix, for some integer parameters m and n, where $n \gg m$ and n is a power of 2, and w is a vector of length $m + k \cdot \log(n) < n$ for some integer parameter k.[7] The matrix M is called the *secret key*, which never changes. Unlike the secret M, the vector w, which is kept secret as well, is updated during the execution of the PRNG. Appropriate choices for the parameters m, n, and k are discussed in §4.3.

Let $\|$ denote concatenation of vectors, i.e., for two vectors v and v' of lengths ℓ and ℓ', respectively, $v\|v'$ denotes the vector of length $\ell + \ell'$ for which

$$(v\|v')[i] = \begin{cases} v[i] & \text{if } i < \ell \\ v'[i-\ell] & \text{otherwise} \end{cases}$$

for all $i \in \{0, \ldots, \ell + \ell' - 1\}$. We define $w = v\|c_e$, where v and c_e are vectors of length m and $k \cdot \log(n)$, respectively. The vector c_e is to be understood as the

[7] Note that $\log(\cdot)$ always denotes logarithm base 2 throughout this paper.

Algorithm 1 PRNG with state (M, w)

1: Parse $v\|i_1\|i_2\|\ldots\|i_k \coloneqq w$
2: Set $e \coloneqq \bigvee_{j=1}^{k} b_{i_j}$
3: Compute $g\|w' \coloneqq M^T v + e$
4: $w \coloneqq w'$
5: **return** g

concise formulation of a sparse vector of length n. Let b_i be the unit vector of length n where only the bit at position i is set to 1 (and all other bits are 0). If $c_e = i_1\|i_2\|\ldots\|i_k$, the noise vector e is defined as $e = \bigvee_{j=1}^{k} b_{i_j}$, where i_j denotes the binary representation of a non-negative integer using $\log(n)$ bits. In other words, c_e encodes the positions in e where the bit is set to 1.[8] As an illustrative example, consider the case when $n = 16$, $k = 3$, and $c_e = (1001\|0100\|1100)$, i.e., the bits at indices 9, 4, and 12 are to be set (reading from left to right). Thus, the decoded noise vector is $e = (0001001000010000)$ (with the lowest-order bit on the right). It is important to note that it is possible that the same index occurs more than once, i.e., $i_j = i_{j'}$, for some $j, j' \in \{1, \ldots, k\}$. Consequently, the number of bits set in e is upper bounded by k.

We are now in the position to describe the PRNG algorithm. In the first step, vector v is set to the first m bits in w, and the remaining $k \cdot \log(n)$ bits of w are interpreted as the concise formulation $c_e = i_1\|i_2\|\ldots\|i_k$ of the noise vector e. Next, the noise vector is set to the decoded form of c_e, i.e., $e \coloneqq \bigvee_{j=1}^{k} b_{i_j}$, where each index i_j is extracted from $c_e = i_1\|i_2\|\ldots\|i_k$. In the main step, the n-bit vector $M^T v + e$ is computed, which is interpreted as the concatenation of vectors g and w' of lengths $n - (m + k \cdot \log(n))$ and $m + k \cdot \log(n)$, respectively. Finally, the internal state (M, w) is updated to (M, w') and the output of the PRNG is simply the vector g.

From the description of the algorithm it follows that the algorithm can be implemented using only XOR operations, except for the decoding of the noise vector. The steps of the PRNG algorithm are summarized in Algorithm 1.

3.3 Firekite

The PRNG described in §3.2 can theoretically be used as the basis of several cryptographic constructions. Firekite is a synchronous stream cipher that uses this PRNG, initialized with the secret state (M, w), to produce the keystream directly. Formally, the encryption of a data item d of length $n - (m + k \cdot \log(n))$ is PRNG() + d, where PRNG() is to be understood as the invocation of the PRNG returning the next random vector g of length $n - (m + k \cdot \log(n))$. Thus, the plaintext vectors are processed sequentially, and the output g of the PRNG depends on the internal vector w, which is updated for each invocation of the PRNG. As for any synchronous stream cipher, a ciphertext can be decrypted by simply applying Algorithm 1 again on the ciphertext to obtain the plaintext.

[8] Intuitively, the length of c_e is an approximation of the entropy $\mathcal{H}(e)$ of e.

As we will see, when setting the parameters m, n, and k to appropriate values, the matrix M becomes quite large. For the sake of a low memory footprint and efficient key distribution and management, it is preferable to have short keys. This problem can be addressed by moving from the LPN problem to a variant of the Ring-LPN problem [18]: Consider the ring R of all polynomials in X over $\mathbb{Z}_2$ with binary coefficients reduced modulo $X^b - 1$ for a suitable parameter $b > n$, i.e., a parameter b for which it holds that $(X^b - 1)/(X - 1)$ is irreducible. Let $q_1 \xleftarrow{U} \mathbb{Z}_2^b$ and $q_i \coloneqq X^{i-1}q_1$ for $i \in \{2, \ldots, b\}$. The $b \times b$-matrix Q consists of the rows $q_1, \ldots, q_b$. In other words, the i^{th} row of Q can be constructed by rotating the first row to the left by $i - 1$ positions. The Ring-LPN conjecture states that the problem remains hard when using the matrix Q in place of a fully random matrix, subject to the constraint that the polynomial is irreducible. Thus, the key used in Firekite is the random b-bit vector q_1. It is worth noting that an attacker might obtain the parity of the secret due to the factor $X - 1$ but no more information about the secret is revealed.

As described in §3.1, we require that $m \ll n$ and for n to be a power of 2. Thus, the matrix Q from the Ring-LPN instance cannot be used directly. Instead, given desired parameters n and m and the b-bit key, the matrix M is derived from the key by generating the first m rows of Q and dropping the last $b - n$ columns.

The security implications of this transformation and details on the security of Firekite in general are provided in §4. Moreover, for the Ring-LPN construction, we show in §4.3 that the key size is reduced from mn to $n + c$, where c is a small constant, for suggested parameter sets.

As mentioned before, the internal state w has to be kept secret. If an attacker can control the initialization of w as part of a chosen-ciphertext attack, the attacker can mount a key recovery attack to obtain the secret matrix M. The initial vector w can be derived from a public m-bit nonce N using a standard technique: Let $C_0 = c\|c+1\|\cdots\|c+k-1$ be a vector of length $k\log(n)$, where $c \coloneqq n - m - k\log(n)$, and let $w_0 \coloneqq N\|C_0$. Given the secret key M and the vector w_0 derived from the nonce N, we compute

1. $v\|i_1\|\ldots\|i_k \coloneqq w_{\ell-1}$
2. $e \coloneqq \bigvee_{j=1}^{k} b_{i_j}$
3. $g\|w_\ell \coloneqq M^T v + e$

iteratively for $\ell \in \{1, \ldots, r\}$ for some constant r (defined below) and define $w \coloneqq w_\ell$. Thus, the procedure essentially consists of r successive executions of Algorithm 1, discarding the vector g in each iteration.

We set the value of r as follows: In each iteration, the noise vector contributes $k\frac{m}{n}$ bits on average to vector v because the length of v is m and the noise bits are spread uniformly across n bits. The number of possibilities to choose $k\frac{m}{n}$ positions in vector v (with replacement) is $m^{k\frac{m}{n}}$. During r trials, we assume that $r-1$ of the corresponding noise vectors look random, which implies that the total number of combinations is $m^{k\frac{m}{n}(r-1)}$. Thus, the number of combinations

exceeds $(2^m)^2$ by setting $r \coloneqq 1 + \left\lceil \frac{2n}{k \log(m)} \right\rceil$. This heuristic argument suggests that v is fully random for this choice of r.[9]

This nonce-based variant can also be utilized to turn Firekite into a non-sequential stream cipher by providing a new nonce after a certain number of invocations of the PRNG. It is worth pointing out, however, that the number of invocations must be significantly larger than r to ensure that the cost of processing the nonce does not introduce a substantial overhead.

4 Security Analysis

In order to solve Ring-LPN based on irreducible polynomials, we can apply the same algorithms that solve LPN. It is intuitive that instances where the matrix M is also secret, which is the case in Firekite, require more effort from an attacker compared to the traditional LPN or Ring-LPN instances that are instantiated with the same parameters. On the other hand, utilizing only the top m rows of the Ring-LPN matrix Q as described in §3.2 to generate matrix M can only have a negative impact on security. As there are no known techniques to break Ring-LPN instances for irreducible polynomials faster than standard LPN instances, we conjecture that using a secret matrix M derived from the b-bit key Q_1 does not result in a substantially reduced security compared to using a fully random $m \times n$-matrix M. Therefore, we do not distinguish between these two constructions in the following and simply consider a secret matrix M.

However, we must analyze the implication of generating the noise vector as described in §3.2. While the PRNG proposed in prior art [3] is based on an LPN instance where the noise vector has a Hamming weight of k, the noise vector has a Hamming weight of *at most* k in our case. In this section, we first show that an LPN instance based on our distribution of noise bits is still hard.

In order to be able to study the performance of Firekite in practice, we need to instantiate it with secure parameters. To this end, we transform the problem of breaking our scheme into the LPN problem in §4.2. This transformation allows us to find concrete parameters for our scheme based on the best known attacks against the well-studied LPN problem. In §4.3, we then discuss how to derive parameters for practical use and provide exemplary parameters.

4.1 Security Reduction for Noise Distribution

A well-known result about PRNGs states that it suffices to show the pseudo-randomness of a single application of the PRNG (e.g., §3.3.2 in [15]). Hence, we need to prove that it is hard to distinguish the output $M^T v + e$ from bits chosen uniformly at random. We will now prove that we get an LPN variant that is hard when using the noise distribution of Firekite as opposed to setting each bit in the vector e independently according to the noise parameter $\tau \in]0, \frac{1}{2}[$ as defined in Definition 1.

[9] Note that the square in $(2^m)^2$ is a safety margin.

Let N_τ denote the Hamming weight of the noise vector, i.e., the number of bits that are set to 1 in the n-bit noise vector. In the standard LPN setting, it holds that $\mathbb{E}\left[N_\tau^{\mathsf{LPN}}\right] = \tau n$. In Firekite (FK), the Hamming weight of the noise vector corresponds to the number of distinct elements when picking k out of n elements uniformly at random with replacement. Hence, we have that

$$\mathbb{E}\left[N_\tau^{FK}\right] = n \cdot \left(1 - \left(1 - \frac{1}{n}\right)^k\right). \tag{1}$$

Using the inequality

$$(1+x)^r \leq 1 + \frac{rx}{1-(r-1)x} \tag{2}$$

for $x \in (-1, \frac{1}{r-1}]$ and $r > 1$, we obtain that

$$\mathbb{E}\left[N_\tau^{FK}\right] \overset{(1)}{=} n \cdot \left(1 - \left(1 - \frac{1}{n}\right)^k\right) \overset{(2)}{\geq} \frac{n \cdot k}{n + k - 1}. \tag{3}$$

Furthermore, observing that $\frac{k}{n} < \frac{1}{2}$,[10] we obtain that

$$k > \mathbb{E}\left[N_\tau^{FK}\right] \overset{(3)}{>} \frac{2}{3}k. \tag{4}$$

We assume for the sake of contradiction that an LPN instance with the Firekite noise distribution can be broken, i.e., solved efficiently. Given a standard LPN instance where the size of the secret is n and the noise parameter is τ, we set k such that $\tau n \leq k$, e.g., by setting $k \coloneqq \frac{3}{2}\tau n$.

Since N_τ^{LPN} follows a binomial distribution, we have that $\Pr[N_\tau^{\mathsf{LPN}} = \lfloor \mathbb{E}\left[N_\tau^{\mathsf{LPN}}\right] \rfloor] \in \Omega(1/n)$. For the chosen parameters, it is thus likely that the number of noise bits set to 1 is less than k. If we assume that LPN with our noise distribution can be solved efficiently, then LPN with noise parameter τ can be solved efficiently too: the noise of any given LPN instance could come from our noise distribution with probability at least $\Omega(1/n)$. Hence it follows that the attacker can solve LPN with a complexity that is $O(n)$ times larger than the complexity needed to break LPN with the Firekite noise distribution.

Thus, we can use essentially the same proof as for a constant Hamming weight [3] to show that the Firekite noise distribution is secure under the assumption that LPN is a hard problem.

4.2 Transformation to LPN Problem

In the standard LPN problem, the goal is to reconstruct a secret vector s given pairs of the form (u, c), where u is a random vector and $c = u^T s + e$, with

[10] While this inequality is true for any LPN instance, the ratio k/n is much smaller for parameters we propose in §4.3. Thus, tighter bounds on $\mathbb{E}\left[N_\tau^{FK}\right]$ can be derived for recommended parameters.

$e \in \{0,1\}$, is a noisy scalar product of u and s. The basic problem underlying Firekite restricts the input provided to an attacker. Specifically, the attacker sees only (parts of) $c = M^T v + e$, i.e., both M and v are kept secret. We show in this section that Firekite is based on a problem that can be transformed into the LPN problem.

Let $H = I \| X$ be a parity-check matrix of dimensions $(n-m) \times n$, where I is the $(n-m) \times (n-m)$-identity matrix and X is a $(n-m) \times m$-matrix such that $HM^T = 0$. Furthermore, let M_1 and M_2 denote the matrices comprising the first $(n-m)$ and last m columns of M, respectively. Thus, it holds that $M = M_1 \| M_2$ and $X = M_1^T (M_2^T)^{-1}$. Since $HM^T = 0$, we have that

$$(HM^T)v = H(M^T v) = H(c+e) = (I\|X)c + (I\|X)e = 0. \tag{5}$$

Let $\bar{c}$ and $\bar{e}$ denote the subvectors consisting of the last m elements of c and e, respectively. For the j^{th} component c_j of c it holds that

$$c_j \overset{(5)}{=} x_j^T \bar{c} + e_j + x_j^T \bar{e}, \tag{6}$$

where x_j is the j^{th} row of X, for any $j \in \{1, \ldots, n-m\}$. By defining $\eta_j \coloneqq e_j + x_j^T \bar{e}$, we get

$$c_j \overset{(6)}{=} x_j^T \bar{c} + \eta_j.$$

Since c_j and $\bar{c}$ are known, x_j is a secret, and η_j is a noise bit, this corresponds to a standard LPN problem with the goal of recovering x_j for all $j \in \{1, \ldots n-m\}$. The noise bit η_j consists of noise bit e_j plus $\frac{m}{2}$ additional noise bits in expectation because the expected number of bits set in x_j is $\frac{m}{2}$. Once all vectors $x_1, \ldots, x_{n-m}$ (and thus X) are recovered, M can be recovered as well. Hence, the problem can be transformed into an equivalent LPN problem with higher noise.

While this transformation merely shows that the problem underlying Firekite is *at most* as hard as LPN, we believe the inverse to be true as well, i.e., we conjecture that the two problems are equivalent. As the above transformation is the best available method to attack the problem, we use it to derive secure parameters based on the most efficient known attacks against the standard LPN problem in the next section.

4.3 Parameters

The parameters n, m, and k must be carefully chosen to maximize security and performance while keeping the size of the key and internal state small. In order to determine a level of security for a specific set of parameters, we use the following approach. Since the expected number of noise bits set in e is $n\left(1 - \left(\frac{n-1}{n}\right)^k\right)$ according to Equation (1), we define the probability that any specific bit in e is set as $\tau \coloneqq 1 - \left(\frac{n-1}{n}\right)^k$. Recall, however, that the noise bits are not set according to a binomial distribution.[11] Since the noise η_j is the combination of $\frac{m}{2}+1$ noise

[11] In particular, the variance is substantially lower.

Parameters			Properties			
m	n	k	Key Size (b)	α	r	Sec. Level
216	1024	16	1061	0.63	18	82.76
216	2048	32	2053	0.72	18	82.76
216	4096	54	4099	0.79	21	80.69
216	8192	112	8219	0.80	20	82.60
216	16,384	216	16,421	0.80	21	80.68
224	32,768	416	32,771	0.80	22	80.66
224	65,536	834	65,539	0.79	22	80.82

Table 1: Parameters for 80-bit security and the resulting key sizes (corresponding to parameter b), α and r values, and computed security levels.

terms in expectation, the *bias* is approximately $(1-2\tau)^{\frac{m}{2}+1}$, which is small for a reasonably large m. Given that the bias is low, the most efficient method to solve the LPN problem is the algorithm by Levieil and Fouque [25]. This algorithm can be seen as a Gaussian elimination algorithm performed on blocks of bits (instead of single bits) where at the end the Walsh-Hadamard transform is applied to retrieve a block of the secret. Since τ approximately determines the ratio of k over n, suitable parameters can easily be computed based on their algorithm. Although the execution of the algorithm by Levieil and Fouque solely retrieves a single vector x_j, we use the resulting computational complexity as a bound to derive the entire matrix M. This is a conservative estimate as it may be possible to retrieve all $n-m$ with the same amortized complexity.

The fact that the choice of parameters has a direct impact on performance must also be taken into account. Naturally, the larger the vector g is in relation to n, the more bits are used directly for encryption and fewer invocations of Algorithm 1 are needed. Let

$$\alpha = \alpha(m,n,k) \coloneqq \frac{n-(m+k\log n)}{n}$$

denote the fraction of the output of Algorithm 1 that is used for encryption. A simple model of the amortized cost of Algorithm 1 in terms of the number of instructions that need to be executed *per bit* is

$$I(m,n,k) \coloneqq \frac{m}{2p}\frac{1}{\alpha(m,n,k)} + N(n,k), \tag{7}$$

where p is the number of bits for which an XOR operation can be computed in a single instruction and $N(n,k)$ captures the amortized per-bit overhead to compute and apply the noise vector. The factor $\frac{m}{2p}$ is due to the fact that in expectation $m/2$ bits in vector v are set and p bits are processed together. The term $N(n,k)$ is roughly proportional to k/n and thus only marginally increases the amortized cost. Profiling of our implementation confirms that this model is fairly accurate in that 95% of the computation is spent on the matrix-vector

Parameters			Properties			
m	n	k	Key Size (b)	α	r	Sec. Level
352	1024	16	1061	0.50	17	129.07
352	2048	32	2053	0.66	17	129.07
352	4096	58	4099	0.74	18	128.95
352	8192	120	8219	0.77	18	128.99
352	16,384	228	16,421	0.78	18	128.93
352	32,768	456	32,771	0.78	18	128.93
352	65,536	906	65,539	0.77	19	128.92

Table 2: Parameters for 128-bit security and the resulting key sizes (corresponding to parameter b), α and r values, and computed security levels.

multiplication, which corresponds to the first term in Equation (7). Evidently, m contributes directly to the amortized cost and must be minimized. However, a reduction of m must be compensated with an increase of the number k of noise bits in order to keep the same level of security. Since every increment of k requires an additional $\log(n)$ bits, α becomes smaller, which negatively affects the amortized cost according to Equation (7). In practice, the search space can be restricted quite well because an efficient implementation imposes additional constraints, e.g., all vectors should fit into an integer number of bytes.

Table 1 and Table 2 provide parameters for 80-bit and 128-bit security, respectively, for an increasing length n. As mentioned before, the values in the table are derived analytically, based on the algorithm by Levieil and Fouque. A first observation is that the security level remains basically the same when scaling n and k in the same manner while keeping m constant. This behavior is expected because a constant ratio of n and k implies that the probability that any bit is flipped remains constant as well. The effect of scaling the parameter n will be examined more closely in §5.1. The tables further show that it is possible to have keys lengths, which correspond to the length b of Q_1 as defined in §3.2, that are comparable to the lengths of keys in standard asymmetric encryption schemes. However, it is impossible to have key lengths similar to the lengths of standard symmetric encryption schemes because m must be in the order of hundreds of bits, and it must hold that n is significantly larger than m to ensure that α is not too small. Moreover, we see that increasing n only leads to an increase of α up to a certain point because more bits are needed to encode a single noise bit index as n grows larger. The tables also provide the number r of rounds needed to initialize vector w for the nonce-based variant for each set of parameters. We see that r only varies slightly for the proposed sets of parameters. Finally, it is important to note that increasing the security level does not affect the key size. Still, the increase of m invariably leads to a larger computational cost. Hence it follows that the right choice of parameters highly depends on the requirements in terms of performance and memory constraints. A general recommendation would be to use $n = 4096$ (i.e., $b = 4099$) and m and k as given in the two tables for either 80-bit or 128-bit security. These parameters achieve a decent

trade-off between performance and space. As we will see in §5.2, the memory requirements for these parameters are small enough for use on a standard low-end FPGA. For certain architectures with plenty of memory and wide buses, the last row in the two tables, i.e., the parameters for $n = 65,536$ (i.e., $b = 65,539$) might be preferable for performance reasons. The actual performance for specific parameter ranges is investigated in the subsequent section, which will provide further justification for our choice of recommended parameters.

5 Performance Evaluation

The primary objective is to evaluate the performance in terms of throughput, which is the number of bytes that can be encrypted or decrypted per second. As discussed in §4, the choice of parameters crucially affects not only the attained level of security but also the performance. Therefore, performance is evaluated for a range of parameters corresponding to different security levels.

The procedure to obtain the desired measurement results is the same for all experiments: Random input data is allocated in memory, which is then encrypted and the time required for this encryption is measured. This process is repeated 20 times and all measured times are recorded. The reported throughput is simply the ratio of the input size and the *median* of all recorded times in seconds required to process the input. Note that we solely report the median value because the variance is so small that the differences would hardly be visible in the figures. Similarly, the input size was varied from tens of kilobytes up to one gigabyte without any significant impact on performance on all considered platforms, suggesting that the measured throughput reflects the throughput that would be observed in real-world applications.

The impact of the parameters on performance is analyzed in §5.1 using our software implementation. We further explore the potential of parallelization and evaluate performance gains in a multi-threaded environment. It is important to note that this implementation is neither tested nor analyzed sufficiently for practical use. In particular, it might be susceptible to side-channel attacks because the implementation uses direct memory access and executes XOR operations conditional on the bits in the secret state w. While an optimized, well-tested implementation might yield somewhat different numbers, we believe that the evaluation results capture the relative performance with respect to parameterization reasonably well. Since Firekite is better suited to be run on dedicated hardware, we further implemented it on a low-power FPGA. The evaluation results on this platform are presented in §5.2. Unlike the software implementation, the constant-time FPGA implementation is significantly better protected against side-channel attacks.

Naturally, a base of comparison is needed to put the performance numbers into perspective. We chose to compare our software and hardware implementation against the *Advanced Encryption Standard* (AES) [8] due to its ubiquity and high level of efficiency. This comparison is meant to illustrate how close an LPN-based scheme can come to a state-of-the-art symmetric encryption scheme

in terms of performance on the given hardware. A thorough comparison against multiple state-of-the-art stream ciphers on different hardware platforms is beyond the scope of this work.

5.1 Performance on a Desktop Computer

The experiments to analyze the impact of the parameters on performance were conducted on a quad-core Intel Core i5-4570 at 3.2 GHz with 8GB of DDR3 memory (at 1.6 GHz). Our Firekite implementation is written in C++ and consists of roughly 2000 lines of code. It is compiled using the optimization flags `-O3` and `-funroll-loops` for most classes. The additional compilation flag `-mavx2` is added for the core classes that perform XOR operations in order to make use of *Advanced Vector Extensions* (AVX),[12] which add several SIMD instructions that operate on 256 bit inputs. Thus, an XOR operation can be applied to $p = 256$ bits per cycle. Recall that the amortized number of instructions to encrypt a single bit is roughly proportional to $1/p$ according to Equation (7), i.e., any non-trivial increase of p leads to a substantial improvement of throughput.

The level of security is raised primarily by increasing the number m of n-bit vectors. Equation (7) states that the computational effort grows linearly with m, which implies that a greater level of security results in a lower throughput. In order to test this hypothesis, values for m and k have been chosen that maximize throughput while achieving a security level of $80, 90, \ldots, 150$ for the two recommended values for n, i.e., $n = 4096$ and $n = 65,536$. As discussed in §4.3, the algorithm by Levieil and Fouque is used to determine the security level of a specific set of parameters m, n, and k based on the transformation to the standard LPN problem (see §4.2).

The measured throughput for the chosen parameter sets is given in Figure 2. It is evident from this figure that the hypothesis is true in that the performance degrades when increasing the security level. While the rate of degradation slightly decreases for larger levels of security, the simplified model that assumes a linear relationship between security level and throughput is fairly accurate.

An interesting observation is that there is a substantial gap in the attained throughput for $n = 4096$ and $n = 65,536$. A plausible explanation for this gap is that a larger n is likely to result in fewer cache misses. The effect of increasing n is studied in a second experiment. Specifically, all valid values for n in the range from $n = 2^{10}$ to $n = 2^{18}$ are tested. The parameters m and k have been set to values that maximize throughput for the two security levels 80 and 128. The result of this experiment is depicted in Figure 3.

The figure shows that the vector length n considerably affects performance. When n is small, there are frequent cache misses, leading to a low throughput. The rate at which performance improves slows down when reaching $n = 4096$. Thus, this value for n is a good choice when memory is limited. However, there is

[12] Note that the flag `-mavx` can be used instead, in which case the `PXOR` instruction is used in place of `VPXORS`, resulting in 256-bit operations but with fewer execution ports.

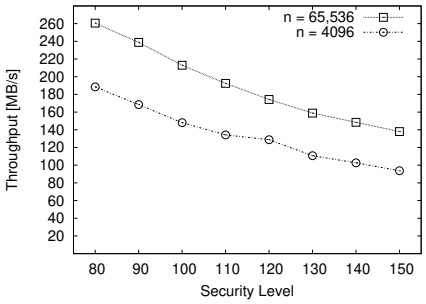

Fig. 2: The change in throughput is shown for $n = 4096$ and $n = 65,536$ when increasing the security level from 80 to 150 in increments of 10.

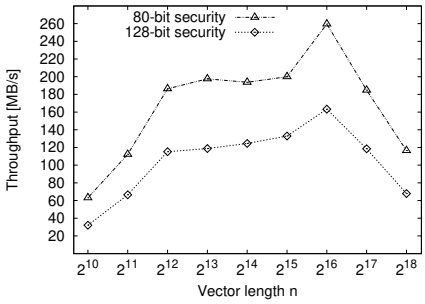

Fig. 3: The effect on the throughput when increasing the vector length from $n = 2^{10}$ to $n = 2^{18}$ is shown for the security levels 80 and 128.

a substantial improvement when increasing n from 2^{15} to 2^{16}. This improvement is due to the fact that 16 is a power of 2, which enables multiple optimizations: First, the noise vector can be constructed efficiently as 16 bits can efficiently be read sequentially. Moreover, when setting m and k to values so that αn is divisible by 256, *aligned memory access* is possible for efficient use of the AVX instructions. On the given test machine, throughput dropped significantly when increasing n further for two reasons. First, the optimizations for $n = 2^{16}$ are not possible for these vector lengths. Second, if the vectors become too long, cache misses become more frequent and parts of the vectors need to be loaded repeatedly. In fact, the drop is so steep that the throughput for $n = 2^{17}$ is lower than for $n = 2^{12}$. Naturally, results may vary depending on the given hardware architecture. In particular, the peak may occur for a different value of n.

Having discussed how the parameters affect performance in terms of throughput, we proceed to analyze the potential for parallelization. It is easy to see that the computation of $M^T v + e$ can be parallelized well. The basic principle is to partition M into t matrices $M_1, \ldots, M_t$ of dimension $\frac{m}{t} \times n$ and assigning each partition to one of t threads. Additionally, vector v and c_e are also partitioned into smaller vectors roughly of size m/t and $k \log(n)/t$, respectively. Each thread $i \in \{1, \ldots, t\}$ then computes $M_i^T v_i$ and e_i, which requires a fraction of $1/t$ of the entire computational effort. Subsequently, the t matrix-vector products are added together and the *logical or* of all t noise vectors is computed. Finally, the resulting noise vector is added to the computed matrix-vector product.

In reality, this process is slightly more complex because there are several constraints that must be respected when partitioning M, v, and e. Obviously, m/t may not be an integer number, therefore it must be guaranteed that the partitioning uniquely assigns each bit in v to a thread. The splitting of c_e is more involved because care has to be taken that each partition consists of a multiple of $\log(n)$ bits as these many bits encode a single index in the noise vector.

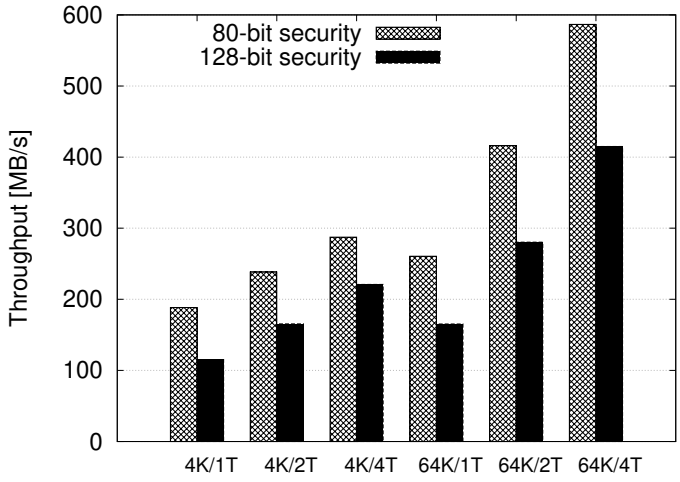

Fig. 4: The effect of using multiple threads on throughput is shown. One, two, and four threads (1T, 2T, 4T) are used for vector lengths $n = 4096$ (4K) and $n = 65,536$ (64K). The results are provided for security levels 80 and 128.

In our next experiment, one, two, and four threads (denoted by 1T, 2T, and 4T, respectively) are used to process the provided input and the processing time is measured. Figure 4 summarizes the results for both recommended vector lengths, $n = 4096$ (4K) and $n = 65,536$ (64K), and security levels 80 and 128. The measurement results indicate that spreading the computational task across multiple threads indeed leads to a higher throughput. The improvement is more substantial for vectors of larger size and for higher security levels, i.e., when parameters n and m are larger. This is due to the fact that increasing these parameters results in more work that can be partitioned among the threads. As an example, throughput increases by merely 26% (21%) when using two threads instead of one (four threads instead of two) for 80-bit security and a vector length n of 4096. By contrast, for $n = 65,536$ and 128-bit security, the throughput improves by 70% and 48% when increasing the number of threads from one to two and from two to four, respectively, resulting in an overall speed-up factor of approximately 2.5. While a multi-threaded execution evidently improves performance, the overhead to synchronize the threads and the effort to merge the partial results from all threads limits the potential of parallelization.

Finally, Table 3 compares the performance numbers against the performance of the AES implementation of OpenSSL.

Hardware acceleration was disabled for some experiments to show the huge effect of having hardware support in the form of the `AES_NI` instruction. Firekite only reaches a similar performance level when hardware acceleration is disabled, $n = 65,536$, and when using multiple threads. While Firekite makes use of AVX/AVX2 instructions, we conjecture that support for operands the size of

Algorithm	Mode	HWA	Throughput [MB/s]
AES-128	CBC	✓	738
AES-128	CTR	✓	2864
AES-128	CBC	×	357
AES-128	CTR	×	258
Firekite (128-bit)	4K/1T	×	115
Firekite (128-bit)	4K/4T	×	221
Firekite (128-bit)	64K/1T	×	165
Firekite (128-bit)	64K/4T	×	415

Table 3: The throughput of AES-128 in CBC and CTR mode with and without hardware acceleration (HWA) are listed, as well as throughput of Firekite for the configurations 4K/1T, 4K/4T, 64K/1T, and 64K/4T for 128-bit security.

n, e.g., $n = 4096$, would be required to become competitive. Naturally, Firekite would further greatly benefit from hardware support for the decoding of the noise vector. Thus, we conclude that hardware support is a general requirement for high performance.

5.2 Performance on an FPGA

Both AES and Firekite have been implemented for execution on a *Cyclone V* FPGA, which is a low-cost and low-power system on a chip with a dual-core ARM Cortex-A9 MPCore processor at 925 MHz.[13] It offers 41,910 adaptive logic modules, 166,036 registers, and 553 RAM blocks. Even though 1GB of external RAM is available and accessible through a dedicated controller, only on-chip RAM is used in our implementation for performance reasons as the access latency for on-chip RAM is lower. Encryption modules were added to the system with a softcore *NIOS II* CPU instantiated in the FPGA fabric. Specifically, version `f` of the CPU is used, which is characterized by high-speed pipelined data paths and available on-board data and instruction caches. Each of the encryption modules contains read and write direct memory access (DMA) units to minimize memory access latencies. The frequency of the clock supplied to all units is 50 MHz.

The goal for the implementation of both AES and Firekite is to utilize the available resources to the largest extent possible in order to maximize performance in terms of the number of bytes that are encrypted per cycle. A custom implementation of AES with the S-box implemented as a lookup table is used in our experiments. We distinguish between single port (SP) and dual port (DP) memory access: for SP memory access it is only possible to read from memory or write to memory but not in the same cycle, whereas DP memory access uses two ports to enable reading and writing at the same time. Consequently, the implementation for SP and DP memory access differ substantially, notably in that

[13] https://www.verical.com/datasheet/intel-fpga-5CSXFC6D6F31C6-N-5759991.pdf.

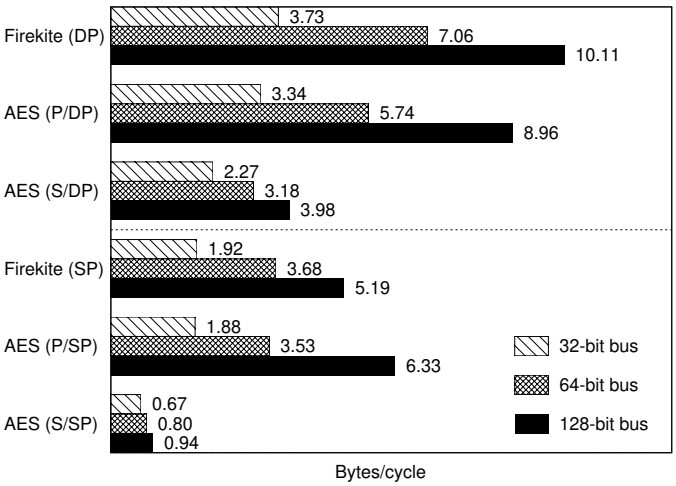

Fig. 5: The number of encrypted bytes/cycle is shown for AES and Firekite using DP (top) and SP (bottom) memory access, for a bus size of 32, 64, and 128 bit. Results are provided for both sequential (S) and parallelized (P) AES executions.

only the DP implementation is *pipelined*, i.e., data is scheduled for encryption (or decryption) as soon as it is fetched from memory.

For AES, we consider a parallelizable mode (CTR) and a non-parallelizable mode (CBC). In non-parallelizable modes, batches of data must be encrypted sequentially; moreover, the encryption of the next batch can only be started after it has been fetched from memory. However, it is important to note that both modes are pipelined for DP memory access. For the more complex version with DP memory access, four stages can be executed in parallel on this FPGA, whereas it is possible to execute 32 stages in parallel for the simpler version with a single-port DMA component. Thus, the sole advantage of the parallelizable modes is that more data can be fetched in parallel, resulting in larger bursts, fewer memory accesses, and consequently higher memory performance. We therefore expect parallelizable modes to perform slightly better, in particular when coupled with dual port memory access.

The parameters that are used for the Firekite implementation are $m = 512$, $n = 4096$, and $k = 64$, which corresponds to a security level of 183. These parameters have been chosen because parameters that are powers of 2 simplify the design. The key advantage in comparison to AES is that all computations can, in theory, be carried out in parallel. The top-level diagram is similar to the diagram for AES and is omitted. The implementation uses numerous registers on the datapath. In order to save space, the rows of matrix M are constructed when needed as described in §3.2. The largest amount of space in the FPGA fabric is consumed by the computational blocks that are used to perform the XOR operations and decode the noise vectors. Since the resources on this FPGA do

not suffice to perform all operations of Algorithm 1 in one cycle, the computation is executed in 32 cycles. In addition to an SP DMA version, a version with DP memory access was developed as well, which introduces extra registers to enable the parallelization of all operations.

As mentioned before, performance is measured in terms of the number of bytes that are encrypted per cycle. The numbers are derived by encrypting a payload of 48KB and dividing 49,152 by the number of used cycles. Figure 5 summarizes the results for AES, with and without parallelization, and Firekite for SP and DP memory access. Furthermore, the effect of using different bus sizes is presented as well.

The results are encouraging as Firekite encrypts more bytes per cycles when using either DP or SP DMA components, except for SP memory access and a bus size of 128 bits. The throughput of Firekite is higher by a factor of 1.64 to 2.54 (depending on the bus size) for DP memory access, and even 2.87 to 5.5 times larger for SP memory access. As far as resource consumption is concerned, Firekite uses more registers than AES as expected (roughly 20-25K vs. 4-11K). However, the versions of AES using parallelization (both DP and SP memory access) actually use slightly more combinational logic elements (34K vs. 35-36K).

These numbers naturally do not imply that Firekite generally performs better. First of all, it requires significantly more registers and would clearly not perform well when constrained to a small number of registers. Second, the results might be quite different on a different platform. Finally, there are numerous light-weight encryption schemes that would reach a considerably higher throughput given the same resources. Nonetheless, the results demonstrate that an LPN-based encryption scheme can reach decent performance levels, which means that such schemes can potentially become viable alternatives to state-of-the-art symmetric encryption schemes for specific architectures in the future.

6 Related Work

One main application of LPN and variants of LPN is authentication, and a plethora of LPN-based authentication protocols have been proposed: HB [19], HB^+ [21], HB^{++} [6], $\text{HB}^\#$ [14], AUTH [24], and Lapin [18] among others [7, 27]. Several encryption schemes also base their security on the hardness of LPN [1, 9–13, 23, 31]. Constructions of *pseudo-random number generators* [2, 3] and *pseudo-random functions* [30] based on LPN have been presented as well.

Alekhnovich [1] proposed two constructions for public-key encryption schemes that encrypt a given plaintext bit for bit. Improvements were introduced by Damgård et al. [9] and by Döttling et al. [11]. A more efficient scheme building on top of the work of Döttling et al. was presented by Kiltz et al. [23]. HELEN [12] is another encryption scheme that bases its security on LPN and the decisional minimum distance problem. More recently, Yu and Zhang [31] illustrated how LPN can be used in a tag-based encryption scheme.

The work that is most closely related to ours presents the symmetric encryption scheme LPN-C [13]. The secret key in their scheme is a random matrix M.

It uses an error correcting code $\mathcal{C}$ with generator matrix G to encrypt a plaintext vector d: the ciphertext is (v, y) for a random vector v and $y \coloneqq M^T v + e + G \cdot d$, where e is a noise vector whose bits are sampled from a Bernoulli distribution. In order to decrypt (v, y), $y + M^T v = e + G \cdot d$ is computed and then d is recovered by running the decoding algorithm of $\mathcal{C}$.

Clearly, there are similarities between LPN-C and Firekite: both schemes use a random matrix M as the secret key and the computation of a ciphertext includes a term of the form $M^T v + e$. However, the two encryption schemes are quite different in several respects. First of all, LPN-C uses an error-correcting code $\mathcal{C}$, which is not required for Firekite. Another differentiating factor is the distribution of the bits in the noise vectors. It is a binomial distribution in the case of LPN-C, whereas the distribution for Firekite has a lower variance, and the hamming weight of the noise vector is upper bounded by k. Since the error-correcting code can only successfully recover the plaintext if the number of noise bits does not exceed a given threshold, it is possible that decryption fails with a certain (small) probability. Alternatively, the authors suggest to truncate the binomial distribution to ensure that the Hamming weight of noise vectors does not exceed the correction capacity of $\mathcal{C}$. However, this modification can have a negative impact on the security of the scheme. By contrast, there are no decryption failures for Firekite. What is more, unlike Firekite, LPN-C must generate fresh random numbers for each messages.

While the work introducing LPN-C does not contain any measurement results, it is evident from the specification that LPN-C is unlikely to reach the same level of performance as Firekite for recommended parameters. The lower performance is partly due to the use of an error-correcting code, which may incur a substantial overhead. More importantly, LPN-C requires a much larger m, e.g., $m = 512$ for a security level of 80, because the vector v is made public. According to current knowledge, state-of-the-art attacks can exploit this additional information to recover the secret key more efficiently.

7 Conclusion

We introduced a novel LPN-based synchronous stream cipher, called Firekite, that has a simple structure and is parallelizable, particularly when given hardware support. This is the first work that presents performance numbers of any LPN-based scheme by benchmarking both a software and a hardware implementation. Moreover, it is the first LPN-based scheme that achieves decent throughput numbers on dedicated hardware, albeit at the cost of higher resource usage than state-of-the-art symmetric encryption schemes.

We hope that these results stimulate interest and trigger more research in this direction in order to further explore the potential of practical encryption based on LPN, which may lead to the development of viable alternatives to commonly used symmetric encryption schemes.

References

1. Alekhnovich, M.: More on Average Case vs Approximation Complexity. In: Proc. 44th Symposium on Foundations of Computer Science (FOCS). pp. 298–307 (2003)
2. Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast Cryptographic Primitives and Circular-Secure Encryption Based on Hard Learning Problems. In: Proc. 29th Annual International Cryptology Conference (CRYPTO). pp. 595–618 (2009)
3. Blum, A., Furst, M., Kearns, M., Lipton, R.J.: Cryptographic Primitives Based on Hard Learning Problems. In: Proc. 13th Annual International Cryptology Conference (CRYPTO). pp. 278–291 (1993)
4. Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant Learning, the Parity Problem, and the Statistical Query Model. In: Proc. 32nd Annual ACM Symposium on Theory of Computing (STOC). pp. 435–440 (2000)
5. Bogos, S., Vaudenay, S.: Optimization of LPN Solving Algorithms. In: Proc. 22nd International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT). pp. 703–728 (2016)
6. Bringer, J., Chabanne, H., Dottax, E.: HB^{++}: a Lightweight Authentication Protocol Secure against Some Attacks. In: Proc. 2nd International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing (SecPerU). pp. 28–33 (2006)
7. Cash, D., Kiltz, E., Tessaro, S.: Two-Round Man-in-the-Middle Security from LPN. In: Proc. 13th International Conference on Theory of Cryptography (TCC). pp. 225–248 (2016)
8. Daemen, J., Rijmen, V.: The Design of Rijndael: AES – The Advanced Encryption Standard. Springer Science & Business Media (2013)
9. Damgård, I., Park, S.: Is Public-Key Encryption Based on LPN Practical? IACR Cryptology ePrint Archive (2012)
10. Döttling, N.: Low Noise LPN: KDM Secure Public Key Encryption and Sample Amplification. In: Public Key Cryptography. Lecture Notes in Computer Science, vol. 9020, pp. 604–626. Springer (2015)
11. Döttling, N., Müller-Quade, J., Nascimento, A.C.A.: IND-CCA Secure Cryptography Based on a Variant of the LPN Problem. In: Proc. 18th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT). pp. 485–503 (2012)
12. Duc, A., Vaudenay, S.: HELEN: A Public-Key Cryptosystem Based on the LPN and the Decisional Minimal Distance Problems. In: Proc. 6th International Conference on Cryptology in Africa (AFRICACRYPT). pp. 107–126 (2013)
13. Gilbert, H., Robshaw, M.J., Seurin, Y.: How to Encrypt with the LPN Problem. In: Proc. 35th International Colloquium on Automata, Languages, and Programming (ICALP). pp. 679–690 (2008)
14. Gilbert, H., Robshaw, M.J.B., Seurin, Y.: $HB^{\#}$: Increasing the Security and Efficiency of HB^{+}. In: Proc. 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT). pp. 361–378 (2008)
15. Goldreich, O.: Foundations of Cryptography: Volume 1. Basic Tools. Cambridge University Press (2007)
16. Guo, Q., Johansson, T., Löndahl, C.: Solving LPN Using Covering Codes. In: Proc. 20th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT). pp. 1–20 (2014)
17. Heninger, N., Durumeric, Z., Wustrow, E., Halderman, J.A.: Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices. In: Proc. 21st USENIX Security Symposium. pp. 35–35 (2012)

18. Heyse, S., Kiltz, E., Lyubashevsky, V., Paar, C., Pietrzak, K.: Lapin: An Efficient Authentication Protocol Based on Ring-LPN. In: Proc. 19th International Workshop on Fast Software Encryption (FSE). pp. 346–365 (2012)
19. Hopper, N.J., Blum, M.: Secure Human Identification Protocols. In: Proc. 7th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT). pp. 52–66 (2001)
20. Impagliazzo, R., Levin, L.A.: No Better Ways to Generate Hard NP Instances than Picking Uniformly at Random. In: Proc. 31st Annual Symposium on Foundations of Computer Science (FOCS). pp. 812–821 (1990)
21. Juels, A., Weis, S.A., et al.: Authenticating Pervasive Devices with Human Protocols. In: Proc. 25th Annual International Cryptology Conference (CRYPTO). pp. 293–308 (2005)
22. Katz, J., Shin, J.S., Smith, A.: Parallel and Concurrent Security of the HB and HB^+ Protocols. Journal of Cryptology **23**(3), 402–421 (2010)
23. Kiltz, E., Masny, D., Pietrzak, K.: Simple Chosen-Ciphertext Security from Low-Noise LPN. In: Proc. 17th International Conference on Practice and Theory in Public-Key Cryptography (PKC). pp. 1–18 (2014)
24. Kiltz, E., Pietrzak, K., Cash, D., Jain, A., Venturi, D.: Efficient Authentication from Hard Learning Problems. In: Proc. 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT). pp. 7–26 (2011)
25. Levieil, É., Fouque, P.A.: An Improved LPN Algorithm. In: Proc. 5th International Conference on Security and Cryptography for Networks (SCN). pp. 348–359 (2006)
26. Lyubashevsky, V.: The Parity Problem in the Presence of Noise, Decoding Random Linear Codes, and the Subset Sum Problem. In: Proc. 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX). pp. 378–389 (2005)
27. Lyubashevsky, V., Masny, D.: Man-in-the-Middle Secure Authentication Schemes from LPN and Weak PRFs. In: Proc. 33rd Annual Cryptology Conference (CRYPTO). pp. 308–325 (2013)
28. Shor, P.W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM Journal on Computing **26**(5), 1484–1509 (1997)
29. Trevisan, L.: Extractors and Pseudorandom Generators. Journal of the ACM **48**(4), 860–879 (2001)
30. Yu, Y., Steinberger, J.P.: Pseudorandom Functions in Almost Constant Depth from Low-Noise LPN. In: Proc. 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT). pp. 154–183 (2016)
31. Yu, Y., Zhang, J.: Cryptography with Auxiliary Input and Trapdoor from Constant-Noise LPN. In: Proc. 36th Annual International Cryptology Conference (CRYPTO). pp. 214–243 (2016)
32. Zhang, B., Jiao, L., Wang, M.: Faster Algorithms for Solving LPN. In: Proc. 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT). pp. 168–195 (2016)