https://doi.org/10.1093/comjnl/bxac146

How Not To Design An Efficient FHE-Friendly Block Cipher: Seljuk

Tomer Ashur[1], Mohammad Mahzoun[1] and Dilara Toprakhisar[2]

[1]*Mathematics and Computer Science, Coding Theory and Cryptology, TU Eindhoven, 5612 AZ Eindhoven, the Netherlands*
[2]*imec-COSIC, KU Leuven, 3001 Leuven, Belgium*
**Corresponding author: m.mahzoun@tue.nl*

With the rapid increase in the practical applications of secure computation protocols, increasingly more research is focused on the efficiency of the symmetric-key primitives underlying them. Whereas traditional block ciphers have evolved to be efficient with respect to certain performance metrics, secure computation protocols call for a different efficiency metric: *arithmetic complexity*. Arithmetic complexity is viewed through the number and layout of nonlinear operations in the circuit implemented by the protocol. Symmetric-key algorithms that are optimized for this metric are said to be algebraic ciphers. It has been shown that recently proposed algebraic ciphers are greatly efficient in ZK and MPC protocols. However, there has not been many algebraic ciphers proposed targeting Fully Homomorphic Encryption (FHE). In this paper, we evaluate the behavior of *Vision* when implemented as a circuit in an FHE protocol. To this end, we present a state-of-the-art comparison of *AES* and Vision implemented using HElib. Counterintuitively, *Vision* does not deliver a better performance than AES in this setting. Then, by attempting to improve a bottleneck of the FHE implementation evaluating Vision we present a new cipher: *Seljuk*. Despite the improvement with respect to *Vision*, *Seljuk* does not deliver the expected performance.

Keywords: FHE; Algebraic Ciphers

Received 26 February 2022; Revised 7 August 2022; Editorial Decision 4 September 2022
Handling editor: Chris Mitchell

1. INTRODUCTION

Traditional block ciphers are built with carefully chosen linear and nonlinear layers to resist well-studied attacks. Besides being secure, traditional block ciphers are designed to be efficient in their hardware and software implementations. Depending on the target application domain, their design optimizes the running time, gate count or memory/power consumption. For instance, while an Internet of things device calls for lower memory/power consumption and gate count, a high speed router calls for lower latency. Different efficiency metrics come into consideration when the target application domain is a secure computation protocol. *Multi-Party Computation* (MPC), *Zero-Knowledge* (ZK) *proofs* and *Fully Homomorphic Encryption* (FHE) are examples of such advanced cryptographic protocols that are described via algebraic operations. These operations can be translated into arithmetic computations and vice versa. Converting computations into a sequence of algebraic operations over a finite field is called *arithmetization* and it was first applied to cryptographic protocols by Lund *et al*. [1].

Consider the following scenario in which a secure computation protocol employs a block cipher: a client sends its data encrypted under an FHE scheme to a cloud server that operates on encrypted data. However, depending on the complexity of the function performed by the server, the scheme's parameter set might result in a drastic increase in the size of the freshly encrypted ciphertext. Consequently, this increase would add unwanted overhead to the communication. One solution to this problem is *transciphering*, which means all the private data sent by a client can be encrypted using a block cipher. Then, the server decrypts homomorphically, and consequentially they are able to operate on encrypted data without additional overhead to the communication [2].

The increase in the popularity of secure computation protocols gave rise to new designs known as algebraic ciphers such as *Mimc* [3], *Poseidon* [4], *Vision* and *Rescue* [5]. Unlike traditional block ciphers, the design of these algorithms is driven by arithmetic complexity improving the efficiency of the protocol employing them. Therefore, the relevant attacks and security of these algorithms are also different.

As the design of algebraic ciphers is an evolving research area, there are several design strategies introduced as a framework. The *Marvellous* design strategy [5] and the *Hades* design strategy [6] are examples of such design strategies. The ciphers that are proposed following these design strategies are shown to be efficient in ZK and MPC applications. Although numerous algebraic ciphers were proposed for ZK and MPC applications, there are not many algebraic ciphers proposed in the context of FHE. Yet, FHE is an effective tool to remove privacy barriers obstructing data sharing. Therefore, designing an FHE-friendly algebraic cipher still stands as a research area that needs to be improved.

In this work we present an exploratory study of the expected behaviors of algebraic ciphers when implemented as a circuit in an FHE protocol. For that aim, we adequately compare *AES* and *Vision* [5]. Even though *Vision* is more efficient in ZK and MPC protocols, we show that it performs slower than *AES* in an FHE setting. Then, using the outcome of the comparative study we identify a bottleneck in *Vision* when implemented as a circuit in an FHE setting. Finally, we suggest a novel FHE-optimized cipher *Seljuk* which is still slower than *AES*.

This paper is structured as follows: in Section 2 we recall *AES*, the *Marvellous* design strategy, the ciphers designed following it and Brakerski–Gentry–Vaikuntanathan (BGV)-based FHE. In Section 3 we present the findings of the exploratory study that compares *AES* and *Vision*. Next, we motivate the decisions taken in designing *Seljuk* and give the specification of the cipher in Section 4. Finally, in Section 5 we give a conclusion.

2. PRELIMINARIES

In this section, we mention some related previous work, and some prior knowledge that is required for this work.

2.1. AES

AES [7] is an SP (Substitution–Permutation) network with a fixed block size of 128 bits and key size of 128, 192 or 256 bits. The three variants of *AES* are referred to as *AES*-128, *AES*-192 and *AES*-256. *AES*-128 has $n = 10$ rounds, *AES*-192 has $n = 12$ rounds and *AES*-256 has $n = 256$ rounds. Let us denote the input by $B_0B_1 \dots B_{15}$, where each $B_i = b_{8i}b_{8i+1} \dots b_{8i+7}$ is a sequence of 8 bits. These bytes can be placed in a 4×4 matrix as follows:

$$B = \begin{bmatrix} B_0 & B_4 & B_8 & B_{12} \\ B_1 & B_5 & B_9 & B_{13} \\ B_2 & B_6 & B_{10} & B_{14} \\ B_3 & B_7 & B_{11} & B_{15} \end{bmatrix}$$

Each round of *AES* has the following operations:

- SubBytes: SubBytes is a carefully constructed S-box layer. There are 16 S-boxes with 8-bit input and 8-bit output.
- ShiftRows: Row i is moved to the left by i byte positions.
- MixColumns: Each column of the matrix is mixed together by multiplying with a matrix M.
- AddRoundKeys: If we denote the current state with P and the round key with K, AddRoundKeys create the state $Q = P \oplus K$.

The encryption of *AES* is depicted in Fig. 1.

2.2. The Marvellous Design Strategy

The *Marvellous* design strategy [5] introduces a set of decisions to be taken when designing a secure and efficient algebraic cipher. The state of a *Marvellous* design is an element in the vector space $\mathbb{F}_q^\ell$, with q either a power of 2 or a prime number and $\ell > 1$. A *Marvellous* design is an SP network that repeatedly applies its round function to its state for N iterations. Figure 2 depicts a schematic description of the encryption operation of a *Marvellous* design. A plaintext and a master key are the inputs to the first round. Each round consists of two steps and each step employs three layers: S-box, linear and subkey injection. The subkeys used in subkey injection are derived from the master key through a key schedule algorithm.

The S-box layer of a *Marvellous* round applies an S-box to each of the ℓ state elements. Each S-box consists of a power map $g : x^\alpha$ and is possibly followed by an invertible affine transformation. The motivation behind employing a power map S-boxes is their well-studied cryptanalytic properties [8]. The two steps of a *Marvellous* round employ different S-boxes in terms of their degrees. Let the S-box employed in the first step be denoted by θ_0, and the S-box employed in the second step be denoted by θ_1. θ_0 is chosen such that it has a high degree when the encryption is performed and a low degree when the decryption is performed. θ_1 is chosen such that it serves the opposite goal: it has a low degree when the encryption is performed and a high degree when the decryption is performed. This construction provides a high degree in both encryption and decryption and consequently results in the same cost for both.

The linear layer diffuses local properties to the entire state. This is realized by multiplying the *Marvellous* state vector

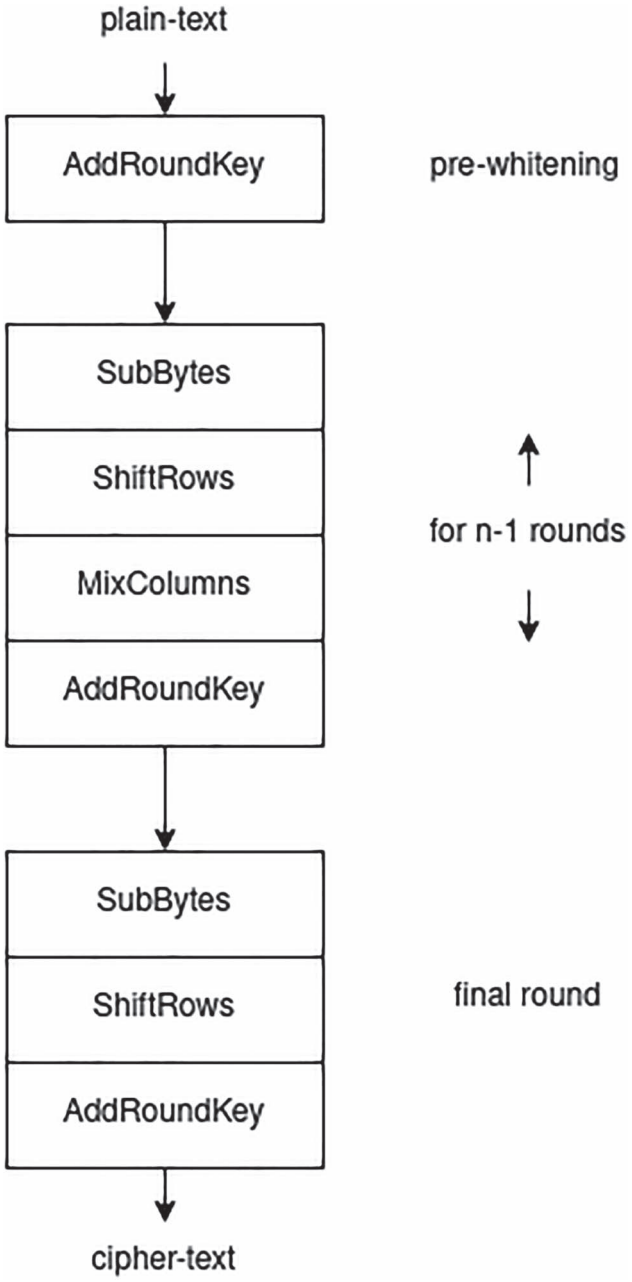

FIGURE 1. The encryption operation of *AES*.

by a maximum distance separable (MDS) matrix. The authors [5] offer to use $\ell \times 2\ell$ Vandermonde matrices using powers of an $\mathbb{F}_q$ primitive element. To obtain the MDS matrix, the Vandermonde matrix is echelon reduced and the $\ell \times \ell$ identity matrix is removed.

The key schedule algorithm of a *Marvellous* design is indeed the iteratively applied encryption round function. In order to generate the subkeys, the round function takes the master key instead of the plaintext input and takes additional round constants instead of the subkeys injected. The round constants are chosen such that they do not belong to a subfield of $\mathbb{F}_q$, nor are rotational invariant. The intermediate state after the round constant injection is provided as a subkey.

The number of rounds in a *Marvellous* round is set to be

$$2 \cdot \max(r_0, r_1, 5),$$

where r_0 is set to be the maximum number of rounds that can be attacked by differential and linear cryptanalysis, higher order differentials and interpolation attacks; r_1 is said to be the instance-specific number of rounds that can be attacked by a GrÃ¶bner basis attack. Five is the sanity factor that protects the cipher against redundant optimization attempts weakening it. As a result, each *Marvellous* instance is set with a minimum of 10 rounds.

2.2.1. *Vision*

Vision is a *Marvellous* family operating on binary fields with its native field $\mathbb{F}_{2^n}$. Most aspects of *Vision* are directly derived from the *Marvellous* design strategy. The *Vision*-specific design decisions are limited to S-box layer which consists of an inversion (with 0 mapped to 0) followed by an affine transformation. A schematic description of a single round (two steps) of *Vision* can be found in Fig. 3. It is constructed by first choosing a fourth-degree $\mathbb{F}_2$-linearized affine polynomial $B(x)$. Then,

$$\theta_1 : \mathbb{F}_{2^n} \mapsto \mathbb{F}_{2^n} : x \mapsto B(x^{-1}),$$

and

$$\theta_0 : \mathbb{F}_{2^n} \mapsto \mathbb{F}_{2^n} : x \mapsto B^{-1}(x^{-1}).$$

2.3. Fully Homomorphic Encryption

FHE is an advanced cryptographic protocol that allows users to evaluate any circuit on encrypted data without first decrypting it. FHE is an effective solution to securely outsourcing computations. However, depending on the size of the computation, the data might be drastically expanded when encrypted under an FHE algorithm. In this case, recalling the example given in Section 1, *transciphering* combining FHE and symmetric encryption allows an efficient encrypted data communication and computation outsourcing.

2.3.1. *BGV Scheme*

BGV is a leveled FHE scheme proposed by Brakerski, Gentry and Vaikuntanathan [9]. Leveled FHE is more restricted than FHE in that the depth of circuits it can evaluate is bounded by the parameters of the scheme. BGV uses *modulus-switching* introduced by Brakerski and Vaikuntanathan [10] to keep the noise under a threshold. Modulus switching is proposed in [10] to be applied once to obtain a ciphertext with less noise. However, it is iteratively applied in BGV to keep the noise under a certain threshold.

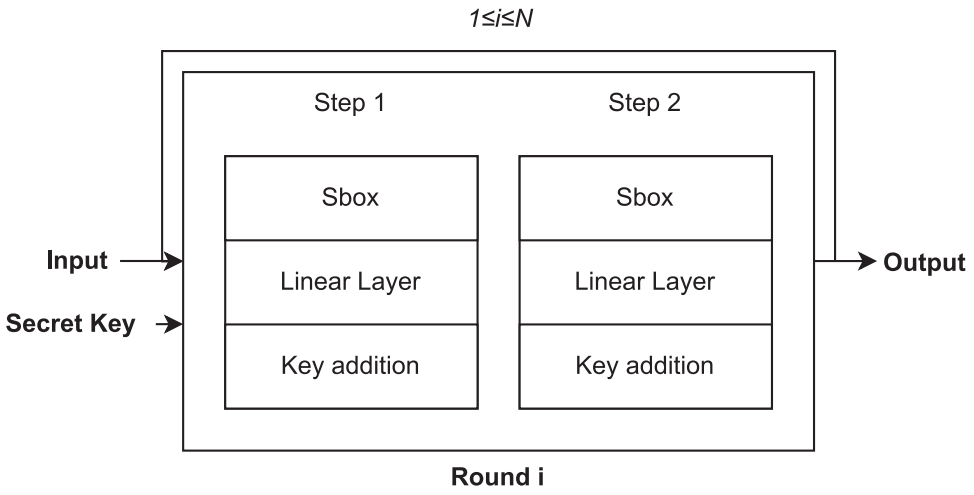

FIGURE 2. The encryption operation of *Vision* and *Rescue*.

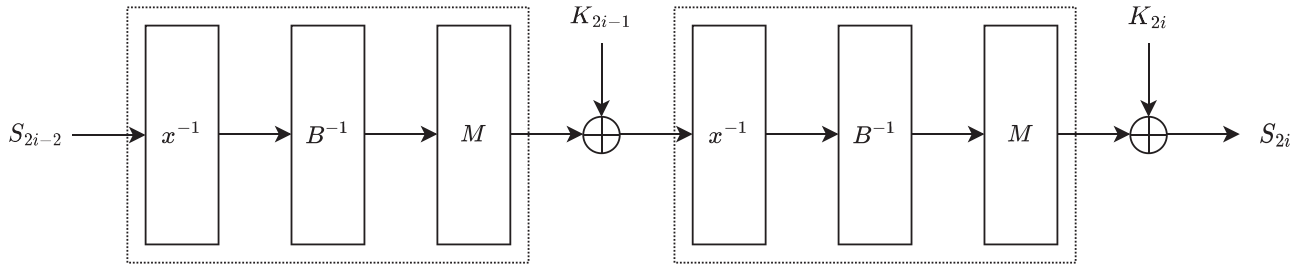

FIGURE 3. One round (two steps) of *Vision*.

In this work, we use a BGV variant proposed by Gentry, Halevi and Smart [11], where both ciphertexts and secret keys are represented as vectors over a polynomial ring $\mathbb{A}$, and the plaintext space is all polynomials over $\mathbb{A}_p$ for $p \geq 2$. Additionally, at any point during the homomorphic evaluation, there are *current integer modulus q* and *current secret key s* that evolve as the homomorphic operations are applied. Decryption is done by taking the inner product of the ciphertext c and the current secret key s over $\mathbb{A}_q$. Then the result is reduced modulo p:

$$a \leftarrow [\ [\ \underbrace{\langle c, s\rangle \mod \Phi_m(X)]_q}_{\text{noise}}\ ]_p. \tag{1}$$

`Addition`, `multiplication` and `automorphism` are used to evaluate circuits and therefore alter the data encrypted under these ciphertexts. `Key − switching` and `modulus − switching` are used to control the complexity of the evaluation and therefore do not affect the underlying data.

Packed ciphertexts. This FHE scheme allows performing operations on packed ciphertexts. Smart and Vercauteren [12] proposed using the Chinese Remainder Theorem to represent the plaintext space $\mathbb{A}_p$ as a vector of plaintext slots. This applies when $\Phi_m(X)$ factors modulo p into l irreducible polynomials such that $\Phi_m(X) = \prod_{j=1}^{l} F_j(X) \bmod p$. Then, a plaintext polynomial $a(X) \in \mathbb{A}_p$ can be represented as encoding l different plaintext polynomials with $a_j = a \bmod F_j$. `Addition` and `multiplication` operations are then performed slot-wise. However, this is not the case for `automorphism`. If i is a power of two, then the transformation $a \mapsto a^{(i)}$ can be realized for each slot separately, and this transformation is called a `Frobenius automorphism`. Conversely, if i is not a power of two, then the transformation acts as a shift operation between the different slot elements.

3. COMPARING AES AND VISION IN A BGV-LIKE CRYPTOSYSTEM

In this section, we present the comparative analysis of *AES* and *Vision* in a BGV-like cryptosystem [11] for 128-bit security in terms of latency (i.e. the time it takes the encryption function to finish).

For our benchmarks, we use an existing implementation of a leveled homomorphic encryption that can evaluate the *AES* circuit presented by Gentry *et al.* [7] built on top of the HElib library. The implementation is based on a variant of BGV cryptosystem [9]. Additionally, we describe working

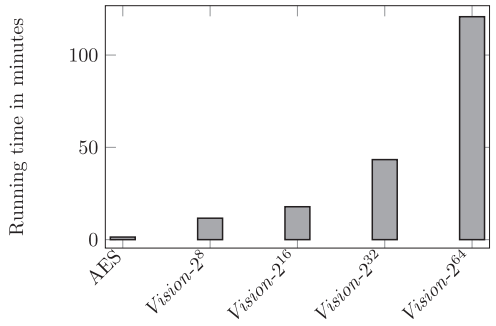

FIGURE 4. Running time comparison of *AES* and *Vision* for 128-bit state.

implementations of a leveled homomorphic encryption that can evaluate a *Vision* circuit built on top of the HElib library.

All our benchmarks are performed in an environment that runs Ubuntu Server 18.04 LTS with 3 TB RAM and 4 x Intel(R) Xeon(R) Gold 6136 CPU @ 3.00GHz.

Benchmarking with a 128-bit state. For a fixed state size, we can choose the dimension of the state for *Vision*. We use instances of *Vision* operating on $\mathbb{F}_{2^8}$, $\mathbb{F}_{2^{16}}$, $\mathbb{F}_{2^{32}}$ and $\mathbb{F}2^{64}$. Figure 4 compares the running times of *AES* and *Vision* instances for a 128-bit throughput. *AES* performs 88% faster than the most efficient instance of *Vision* ($\mathbb{F}_{2^8}^{16}$). The reason *Vision* is slower than *AES* is that it requires a deeper circuit which in turn requires a larger cyclotomic polynomial, $\Phi(m)$, to evaluate. Therefore, apart from requiring more primitive operations (i.e. multiplications, additions and automorphisms), the running time of each primitive operation is longer due to the larger $\Phi(m)$.

Benchmarking with larger state sizes. We reproduced our benchmarks for state sizes larger than 128 bits using *Vision* instances operating on the same fields. The motivation for this is that by increasing the number of state elements we can increase the throughput while keeping the running times of the round operations constant. Figure 5 illustrates the difference between running times and Fig. 6 illustrates the ratio of the increase in the running times of *AES* and *Vision* for increasing throughput (i.e. 256, 512, 1024 and 2048 bit states). The sharp drop in the running time of the *Vision* instance for $\mathbb{F}_{2^{64}}^2$ is due to the decrease in its number of rounds.

Even though the increase in the ratio of the running times of *Vision* is significantly less than the increase in the ratio of the state sizes, *AES* performs 45% faster than the most efficient *Vision* instance ($\mathbb{F}_{2^{16}}^{128}$) yielding 2048-bit throughput. To identify the root cause of this poorer performance of *Vision*, we determined that the operations substantially increase the latency. To this end, individual running times of the operations for a 128-bit state are isolated. Figure 7 depicts these running times of each individual operation in a *Vision* round for $\mathbb{F}_{2^8}^{16}$, $\mathbb{F}_{2^{16}}^{8}$, $\mathbb{F}_{2^{32}}^{4}$ and $\mathbb{F}_{2^{64}}^{2}$.

This comparative analysis concludes that the computation of the inversion and the dense affine polynomial are the most expensive operations for larger degrees of the field extension. Even though *Vision* achieves a compact algebraic description in ZK and MPC, it has a poor performance in FHE. This is because *Vision* heavily depends on ZK and MPC specific computations that cannot be easily computed in FHE (e.g. masked operations in MPC). For instance, inversion is efficiently computed in MPC by means of masking and offloading the heavy operations to the offline phase. However, in FHE this being unavailable; the number of operations required to compute inversion increases as the degree of the field extension increases. Therefore, *different efficiency metrics are involved in FHE applications such as circuit depth and the number of multiplications, Frobenius automorphisms and rotations*.

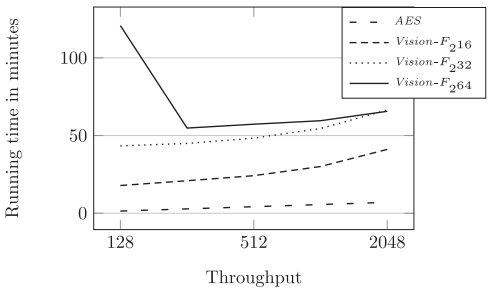

FIGURE 5. Comparison of the running times of *AES* and *Vision* for states that are larger than 128 bits.

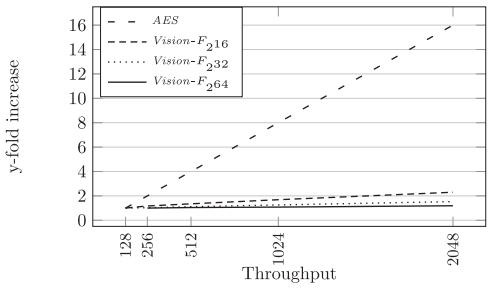

FIGURE 6. Fold increases in the running times of *AES* and *Vision* for states that are larger than 128 bits.

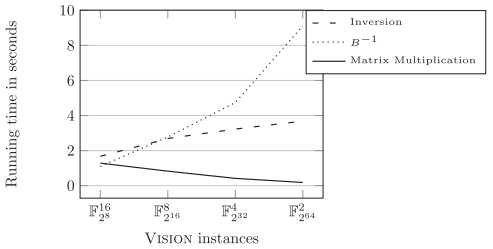

FIGURE 7. Running time comparison for each *Vision* round operation for different *Vision* instances. We see that the running time of B^{-1} grows exponentially, that of inversion linearly and that of the matrix multiplication decreases linearly.

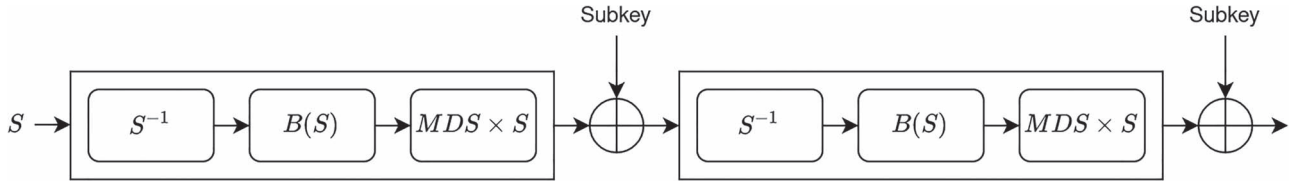

FIGURE 8. A *Seljuk* round function.

4. SELJUK

Our findings in Section 3 suggest that in order to reduce the performance difference between *AES* and *Vision*, one can instantiate a novel *Marvellous* cipher optimizing for the number of operations and circuit depth. However, it is important to ensure that the improvements do not jeopardize security. In this section, we present a new *Marvellous* cipher *Seljuk*.[1]

4.1. Rationale

The increase in the degree of the field extension most notably increases time complexity of B^{-1}. Therefore, in *Seljuk*, we focused on improving the affine polynomial B^{-1} used in even steps by decreasing the number of monomials in the polynomial. Improving B^{-1} does not decrease the circuit depth, but the number of operations to compute the polynomial. Therefore, we hypothesized that there would be a significant improvement in the latency.

4.2. Cipher Description

Seljuk operates on binary fields with its native field $\mathbb{F}_{2^n}$. Most aspects of *Seljuk* are directly derived from the *Marvellous* design strategy. For the sake of completeness, the state is an element of $\mathbb{F}_{2^n}^{\ell}$. Unlike *Vision*, the two steps of a *Seljuk* round are identical and rather than an affine polynomial, a nonlinear polynomial is used in the S-box. To construct the S-box π, a sparse eighth-degree polynomial ($B(X)$) over $\mathbb{F}_{2^n}$ is chosen to decrease the number of operations (i.e. only one Frobenius automorphism):

$$B(x) = x^8 + x \qquad \in \mathbb{F}_{2^n}[x].$$

Then,

$$\pi : \mathbb{F}_{2^n} \mapsto \mathbb{F}_{2^n} : x \mapsto B(x^{-1}).$$

The linear layer follows the *Marvellous* rationale in Section 2.2. Figure 8 depicts a schematic description of the *Seljuk* round function. To generate the ciphertext from a given plaintext, the round function is iterated N times. A key injection with a subkey derived from the master key takes place before the first round, between every two steps, and after the last round. Pseudo-code of *Seljuk* is listed in Algorithm 1.

Algorithm 1: *Seljuk*

Input : Plaintext P, subkeys K_s for $0 \leq s \leq 2N$
Output: $Seljuk(K, P)$
$S_0 = P + K_0$
for $j \leftarrow 1$ ***to*** N **do**
  for $i \leftarrow 1$ ***to*** ℓ **do**
    $Inter_j[i] = (S_{j-1}[i])^{-1}$
    $Inter_j[i] = B(Inter_j[i])$
  for $i \leftarrow 1$ ***to*** ℓ **do**
    $S_j[i] = \sum_{k=1}^{\ell} M[i,k] Inter_j[k] + K_{2j-1}[i]$
  for $i \leftarrow 1$ ***to*** ℓ **do**
    $Inter_j[i] = (S_j[i])^{-1}$
    $Inter_j[i] = B(Inter_j[i])$
  for $i \leftarrow 1$ ***to*** ℓ **do**
    $S_j[i] = \sum_{k=1}^{\ell} M[i,k] Inter_j[k] + K_{2j}[i]$
return S_N

Key Schedule. The key scheduling algorithm follows the *Marvellous* rationale. In order to derive the subkeys, the *Seljuk* round function is applied iteratively. The master key is fed through the plaintext interface, and the round constants are injected as the subkeys. Then, the intermediate state after the round constant injection is provided as a subkey. Round constants are generated by the means of the exact same method used in the *Marvellous* design.

4.3. Benchmarks

Table 1 shows that there is still a difference between *Seljuk* and *AES*. For a 128-bit state, *AES* performs 89% faster than the most efficient *Seljuk* instance (i.e. *Seljuk*-$\mathbb{F}_{2^{16}}^{8}$). Notice that *Seljuk* has a poorer performance than *Vision* when compared with *AES* for 128-bit state. The reason is that due to the degree of $B(X)$, there is no *Seljuk*-$\mathbb{F}_{2^8}^{16}$ instance which would have been more efficient than *Vision*-$\mathbb{F}_{2^8}^{16}$. However, as evident from Fig. 9, *Seljuk* achieves a big improvement especially for the higher

[1] The warlord that gives the name of our cipher was the eponymous founder of the Seljuk dynasty. Similarly, the cipher *Seljuk* constitutes the *Seljuk* dynasty of algebraic ciphers. *Seljuk* was the son of Tuqaq, also known as *Iron Bow* due to his skills. *Seljuk* provides the transition from the *Marvellous* family (*Iron Man*) to *Seljuk* dynasty.

TABLE 1. Running times of *AES* and *Seljuk* **in minutes** for 128-bit state. Security (bits) is provided by HElib, and m is chosen such that the underlying structure allows computations for the respective state.

AES

State	m	Security (bits)	No. rounds	Running time (min)
$\mathbb{F}_{2^8}^{16}$	53 261	141.924	10	1.4

Seljuk

State	m	Security (bits)	No. rounds	Running time (min)
$\mathbb{F}_{2^{16}}^{8}$	116 885	132.476	10	13.07
$\mathbb{F}_{2^{32}}^{4}$	164 737	206.157	10	27.58
$\mathbb{F}_{2^{64}}^{2}$	164 737	142.462	13	50.94

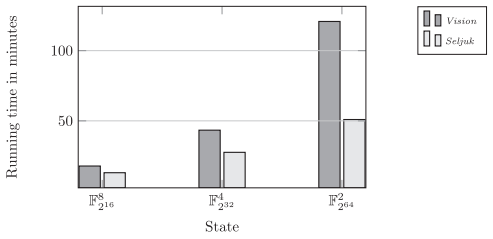

FIGURE 9. Running time comparison of Vision and *Seljuk* for fixed state size of 128 bits.

degrees of the field extension. For example, *Seljuk* performs 58% faster than Vision for $\mathbb{F}_{2^{64}}^{2}$.

5. CONCLUSION

We compared the efficiency of *Vision* and *AES* in the FHE setting. Surprisingly, we observed that despite the fact that *Vision* is more efficient than *AES* in ZK systems and MPC, this is not the case for FHE. Our analysis shows that *AES* is 45% faster than *Vision* in the FHE setting. Thereafter, we analyzed the steps in *Vision* and designed a new block cipher, *Seljuk*, based on the *Marvellous* design strategy. The goal of *Seljuk* is to replace *AES* in the FHE setting. Analyzing the performance of *Seljuk* it can be concluded that for large fields, it is faster than *Vision*. However, it still performs slower than *Vision* and *AES* in the FHE setting. We note that due to its poor performance, we did not evaluate *Seljuk*'s security and we do not recommend it for use.

DATA AVAILABILITY STATEMENT

The data underlying this article are available in the article.

ACKNOWLEDGEMENTS

Tomer Ashur is an FWO post-doctoral fellow under Grant Number 12ZH420N. This work was supported by CyberSecurity Research Flanders with reference number VR20192203. In addition, this work was partially supported by the Research Council KU Leuven, C16/18/004 through the IF/C1 on New Block Cipher Structures and by the Flemish Government through FWO Project Locklock G0D3819N.

REFERENCES

[1] Lund, C., Fortnow, L., Karloff, H.J. and Nisan, N. (1990) Algebraic Methods for Interactive Proof Systems. In *31st Annual Symposium on Foundations of Computer Science* (Vol. ISt. Louis, Missouri, USA, 22-24 October), pp. 2–10. IEEE Computer Society, Washington, DC.

[2] Naehrig, M., Lauter, K.E. and Vaikuntanathan, V. (2011) Can homomorphic encryption be practical? In Cachin, C., Ristenpart, T. (eds) *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW*Chicago, IL, USA, 21 October, pp. 113–124. ACM, New York City.

[3] Albrecht, M.R., Grassi, L., Rechberger, C., Roy, A. and Tiessen, T. (2016) MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity. *IACR Cryptol. ePrint Arch.*, 2016, 492.

[4] Grassi, L., Kales, D., Khovratovich, D., Roy, A., Rechberger, C. and Schofnegger, M. (2019) Starkad and Poseidon: New Hash Functions for Zero Knowledge Proof Systems. *IACR Cryptol. ePrint Arch.*, 2019, 458.

[5] Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S. and Szepieniec, A. (2020) Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols. *IACR Trans. Symmetric Cryptol.*, 2020, 1–45.

[6] Grassi, L., Lüftenegger, R., Rechberger, C., Rotaru, D. and Schofnegger, M. (2020) On a Generalization of Substitution-Permutation Networks: The HADES Design Strategy. In

Canteaut, A., Ishai, Y. (eds) *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part II*Zagreb, Croatia, 10-14 MayLecture Notes in Computer Science (Vol. 12106), pp. 674–704. Springer, Berlin.

[7] Gentry, C., Halevi, S. and Smart, N.P. (2012) Homomorphic Evaluation of the AES Circuit. *IACR Cryptol. ePrint Arch.*, 2012, 99.

[8] Nyberg, K. (1993) Differentially Uniform Mappings for Cryptography. In Helleseth, T. (ed) *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, 23–27 May* Lecture Notes in Computer Science (Vol. 765), pp. 55–64. Springer, Berlin.

[9] Brakerski, Z., Gentry, C. and Vaikuntanathan, V. (2012) (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In Goldwasser, S. (ed) *Innovations in Theoretical Computer Science*Cambridge, MA, USA, 8-10 January, pp. 309–325. ACM, New York City.

[10] Brakerski, Z. and Vaikuntanathan, V. (2011) Efficient Fully Homomorphic Encryption from (Standard) LWE. In Ostrovsky, R. (ed) *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS*Palm Springs, CA, USA, 22-25 October, pp. 97–106. IEEE Computer Society, Washington, DC, United States.

[11] Gentry, C., Halevi, S. and Smart, N.P. (2012) Fully Homomorphic Encryption with Polylog Overhead. In Pointcheval, D., Johansson, T. (eds) *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings*Cambridge, UK, 15-19 AprilLecture Notes in Computer Science (Vol. 7237), pp. 465–482. Springer, Berlin.

[12] Smart, N.P. and Vercauteren, F. (2010) Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. In Nguyen, P.Q., Pointcheval, D. (eds) *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Proceedings*Paris, France, 26-28 MayLecture Notes in Computer Science (Vol. 6056), pp. 420–443. Springer, Berlin.