A Simple Homomorphic Obfuscation Scheme over Automorphisms

Lars Willemsen
Supervisors: Bruno Endres Forlin, Marco Ottavi
Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente
21-09-2023

Homomorphic encryption was long considered the holy grail of public encryption but was widely thought unobtainable. Being first theorized to exist by Rivest, Adleman, and Dertouzos in 1978 after observing the homomorphic properties of RSA, homomorphic encryption refers to a class of encryption methods capable of evaluating arbitrary boolean circuits on ciphertexts as if the operations were applied to the plaintext directly. Since being conceptually conceived the academic community has been hard at work trying to find a bounded, secure implementation. In 2009 Craig Gentry was the first to produce such a model[2][29], albeit not very practical yet. This opened the floodgates and various research departments from prominent tech companies sought to find an all-encapsulating solution, yielding large libraries such as Microsoft SEAL[38] and HElib[25]. Unfortunately, these solutions are largely based on variations of LWE (Learning With Error) due to investing in being quantum secure, making them difficult to apply in real-life situations[18]. Simpler homomorphic schemes do exist but haven't been the target of recent research[41]. The goal of this thesis is to attack the problem from the angle of performance first and foremost. Would it be possible to find a homomorphic encryption scheme that we know abides by strict overhead limitations?

I. Introduction

A Homomorphism is a relation in algebra (abstract algebra) that acts like a structure-preserving map. It abides by the property $f(x * y) = f(x) \times f(y)$, where the binary operator is the one belonging to two different mathematical bodies (e.g., groups, rings, and fields). Even though homomorphic encryption does not entirely play by the same rules, one could think of the homomorphic property between the plaintext and ciphertext. In practice, it manifests itself as the feature allows the evaluation of boolean circuits over the ciphertext as if they were applied to the plaintext directly. Or slightly more formally: $Dec(Eval(C, c)) = C(p)$, where C is a boolean circuit, c is a ciphertext, and p is a plaintext.

Homomorphic encryption has a wide range of applications, but primarily in the field of outsourcing computing. In recent years we have seen a surge in cloud computing solutions for vast amounts of data, but the concept of remotely processing potentially sensitive data comes with a lot of privacy concerns. Homomorphic encryption would allow for cloud computing without giving any insight in the underlying data. It would be applicable in similar situations too: like the processing of medical documents or financial data. Unfortunately not all applications fall within practical feasibility even for the latest generations of FHEs. In the last decades it has been a common trend to construct large scale *edge networks*[15]: networks dedicated to offering content much closer to the end user, offering lower latency and cost. These networks tend to consist of devices designed to offer computation power almost perfectly proportional to their designation.

Most implementations of homomorphic encryption abide by the "C-evaluation structure"[6]. This is a small probabilistic model consisting of a few basic properties and four functions. The four functions in question are:

- The Generation function $Gen : \mathbb{N} \times \mathcal{A} \to \mathcal{K}_p \times \mathcal{K}_s \times \mathcal{K}_e$
- The Encryption function $Enc : \mathcal{K}_p \times \mathcal{P} \to \mathcal{X}$
- The Decryption function $Dec : \mathcal{K}_s \times \mathcal{Z} \to \mathcal{P}$.
- The Evaluation function $Eval : \mathcal{K}_e \times \mathcal{C} \times \mathcal{Z}^* \to \mathcal{Y}$

And a few important definitions. In the following all keys are generated by the generation function:

$$pk, sk, evk \leftarrow Gen(1^\lambda, \alpha)$$

Definition 1. *Correct Decryption*

$$\forall m \in \mathcal{P} : Pr[Dec(sk, Enc(pk, m)) = m] = 1$$

Definition 2. *Correct Evaluation*

$$\forall c_i \in \mathcal{X} :$$
$$Pr[Dec(sk, Eval(evk, C, c_1, ..., c_n)) = C(m_1, ..., m_n)] = 1-\varepsilon(\lambda)$$

Definition 3. *Compactness*

$$\forall c_i \in \mathcal{X}, \exists p \in P[X] :$$
$$dim(Eval(evk, C, c_1, ..., c_n)) \leq p(\lambda)$$

Definition 4. *Correct: combination of definition 1 and 2*

Furthermore, the definition explicitly limits the scope to implementations of the aforementioned functions that grow less fast than exponential run-time (usually sizable polynomials). This constraint does not apply to memory complexity. These definitions are also found but slightly modified in the standardization of the consortium for homomorphic encryption, published in 2018[5].

One of the simplest and earliest candidates for a FHE (Fully Homomorphic Encryption) scheme was described by Regev et al. [3]. In his publication, Regev describes the LWE (Learning With Errors) problem. LWE, or more

specifically RLWE (Ring Learning With Errors), and lattice-based cryptography, such as those relying on GapSVP and SIVP, are currently the standard for PQC (Post Quantum Cryptography). Furthermore, he describes the "bootstrapping" technique, that, once combined with a "squashing" phase forms the basis of Gentry's work in the same year[2], producing the first secure FHE.

LWE-based cryptography is based on an inherent amount of noise that needs to be corrected during decryption. However, as boolean circuits, more specifically its gates, are applied to the noisy ciphertext the noise grows too. Linearly with addition operations, and exponentially with multiplication. In Gentry's paper, this was combated with the bootstrapping technique by doubly encrypting, and then decrypting the inner ciphertext to reduce noise. This technique is described in more detail in IV-B. Unfortunately, this technique leads to a significant increase in run-time complexity. Most RLWE-based schemes tend to fall short in practical use by either being very slow, producing massive ciphertexts (some even exponential in the size of the plaintext as described in [3]), or not even being able to generate keys for large amounts of data altogether. Newer implementations belonging to later generations also suffer issues like only being able to produce approximate results, with as trade-off of being to operate on complex numbers as well [34] [10].

Even though FHEs have come a very long way since their first practical conception in 2009, the current schemes are still a long way off from being considered ready for most everyday use cases. This begs the question whether the desirable properties of homomorphic encryption can be leveraged in a smaller, more relaxed context.

II. Problem description

During this thesis, the goal is to find a simple scheme offering homomorphic properties, preferably one abiding by most requirements for a FHE, that imposes little overhead in terms of memory- and computational complexity. The evaluation function needs to be practically applicable on small scale edge devices. These agents are commonly designed to meet the processing power of the task they are designed for and consequently do not offer much leeway in processing power.

Even though optimally all functions in the evaluation scheme would be fast, the Evaluation function itself is what needs to be within feasible boundaries first and foremost. This applies to both the input key size as well run-time complexity. After all, both the Encryption and Decryption could, on paper, be performed outside the processor's workflow. Achieving state-of-the-art security is not the objective, let alone post-quantum security. Instead it must offer obfuscation at the least amount of overhead we could possibly achieve. Consequently, the current frontier of research on homomorphic schemes is unlikely to offer an answer to this problem, but the path that led up to the current implementations might have left a lot of answers in its wake.

To illustrate some of the issues with current FHE schemes we can look at BKV [17], which is a widely implemented scheme. It is a RLWE based algorithm as many are. In order to make use of its homomorphic properties one needs to convert plaintexts to the coefficients of a bounded degree polynomial. Even though this operation is not necessarily very complicated (Encryption and Decryption are not the bottleneck), key generation and operations applied by Evaluation might be. In 2015 a testing framework was written by researchers from MIT to measure the state of performance of applying straightforward circuits in HElib[4]. In this paper some of the most prominent issues with FHE schemes currently available are laid bare: both key generation and the evaluation function parametrized with a specified boolean circuit have best-fit run-time functions exponential in the depth of the circuit, while also growing aggressively with the size of the plaintext. To reach 128 bits of security keys would grow into hundreds of megabytes for very simple circuits. Needless to say for industrial applications we would be talking about hundreds of gigabytes at least and days to generate the required keys. It is also worth noting that many FHE schemes are currently still in the process of being optimized for domain-specific tasks. For example convolution in CNNs (Convolutional Neural Networks) is notoriously slow using BKV and can be optimized aggressively using synthesized logic as demonstrated in [45]. The point is that the current state of homomorphic encryption, even a few *generations* (IV-A) later, isn't quite suited for all the applications it could find use in.

A. Research goals

In this thesis, we aim to find a homomorphic obfuscation solution that compromises on state-of-the-art security to achieve more "practical feasibility" (V-A). The primary objective is that it needs to be applicable in the context of small embedded devices that suffer memory and/or power constraints. Examples of such devices include IoT agents. This would ultimately result in an implementation compiled for a RISC-V based ISA and run on an embedded device. Rather than delivering an encryption scheme by conventional standards, the aim is to provide provable obfuscation. Some of the questions the thesis wishes to answer:

Is it possible to find a scheme derived from a FHE or SWHE (SomeWhat Homomorphic Encryption) that abides by the following characteristics?:

- Fast encryption and decryption, with key-sizes that scale "reasonably" within the size of the parameters (less than quadratic)
- A modest memory footprint that could be implemented reasonably on small scale devices with limited available memory
- Perfect compactness: the ciphertext does not grow with any subsequent evaluations
- Fast evaluation, scaling no worse than linearly in the circuit depth and plaintext width

- Achieves a non-trivial amount of obfuscation suited for specific use cases that value speed above all else

As the remainder of this paper will try to convey we developed a very simple yet promising model abiding by most of these specifications (which were set far prior to the actual development and ultimate concessions).

III. Preliminaries

Despite its simplistic nature understanding the scheme requires a small amount of knowledge of abstract algebra. For the uninitiated this section serves as a brief gentle introduction. For more information on other involved topics consult the appendix. This section does not constitute a small course in any of these topics; they are a mere introduction and only the ones relevant to this paper.

A. Modular arithmetic

Many people are familiar with the infinite sets such as the integers $\mathbb{Z}$ or the rationals $\mathbb{Q}$. These bodies can be classified according to the properties they abide by. Both of the aforementioned are spaces because they abide by strict "define-able" rules, but, for example the integers form a ring as well.

Even though the mentioned spaces are unbounded, computers cannot work with structures of infinite dimensions. Hence we often resort to modular arithmetic to reduce aforementioned bodies while potentially preserving the properties of their super space. The result is a structure that reduces elements to its congruence class. For example $5{*}3 \ \% \ 4$ reduces to its congruence class $x = 15 - n4 \ for \ x >= 0$, which is equal to 3.

B. Fields, groups and rings

As was briefly mentioned in the previous section mathematicians like to categorize bodies by their properties. A set is the most basic of such structures, with spaces following closely. Once we start to equip it with operations things become more interesting. An operation is a function that transforms zero or more inputs to an output. The most commonly known operations include addition and multiplication, both abiding by the signature $(X, X) \to X$. Both these functions have an arity of 2, and are as such referred to as "binary operations". From this signature it is clear that applying this function yields an output that remains within the same domain as its inputs.

A group, often denoted as $(X, *)$ is a space equipped with "some operation". Take for example the integers and equip it with the addition operator. For this to abide by the structure of a group, it needs to follow the following axioms:

- Closure: $a + b \in \mathbb{Z}$
- Associativity: $(a + b) + c = a + (b + c)$
- Identity element: $\epsilon + a = a$
- Inverses: $a + -a = \epsilon$

It is easy to observe the integers abide by these requirements, and thus form a group. The integers are also an abelian group, which means the operation commutes: $a + b = b + a$

To form a ring we equip the space with two operations: addition and multiplication. For the integers we would denote this as $(\mathbb{Z}, *, +)$. For a space to be considered a ring it needs to abide by the following properties:

- Closure: $a + b \in \mathbb{Z}$ and $a * b \in \mathbb{Z}$
- Associativity: $(a + b) + c = a + (b + c)$ and $(a * b) * c = a * (b * c)$
- Identity elements: $0 + a = a$ and $1 * a = a$, the additive and multiplicative identity respectively
- Additive inverses: $a + -a = 0$
- Additive abelianess: $a + b = b + a$
- Distributive: $a*(b+c) = (a*b)+(a*c)$ and $(c+b)*a = (a * c) + (a * b)$

Sometimes rings are defined without a multiplicative identity, which is often referred to as a "rng". Notice how rings are similar to groups but slightly more lenient than we would expect from the additional operation it is equipped with. Rings do not need to have multiplicative inverses!

Thanks to their lenience rings generalize fields, which are a slightly more demanding structure. They must abide by all the axioms of a ring, but also support multiplicative inverses. So in total:

- Closure: $a + b \in \mathbb{Z}$ and $a * b \in \mathbb{Z}$
- Associativity: $(a + b) + c = a + (b + c)$ and $(a * b) * c = a * (b * c)$
- Identity elements: $0 + a = a$ and $1 * a = a$, the additive and multiplicative identity respectively
- Additive inverses: $a + -a = 0$
- Multiplicative inverses: $a * -a = 1$
- Additive abelianess: $a + b = b + a$
- Distributive: $a*(b+c) = (a*b)+(a*c)$ and $(c+b)*a = (a * c) + (a * b)$

The multiplicative inverses also give rise to being able to define fractions of elements by defining: $a/b = a * b^{-1*}$! Similarly the ring, and consequently the field also support subtraction through $a - b = a + b^{-1+}$.

C. Quotient groups

Quotient groups are groups made up of the cosets of its corresponding subgroup. A quotient group may also be referred to as a factor group. A quotient group over the natural number may looks like $\mathbb{N}/a\mathbb{N}$. The index of a subgroup is often denoted with $[G : H]$ which informally reads as "the number of cosets of H in G".

To form a quotient group the subgroup needs to be normal. This is often denoted with $H \trianglelefteq G$, which, informally, states "H is a normal subgroup of G". For a subgroup to be considered normal all its cosets must be invariant under conjugation by members of its super-group. This is often denoted as

$ghg^{-1} \in H$ for all $g \in G$ and all $h \in H$. In practice this property simply guarantees the abelianess of operations on the cosets, which is why the quotient of a quotient group needs to be normal.

In some cases it might be easier to think of a quotient group as a regular group. For example the group mentioned previously $\mathbb{N}/a\mathbb{N}$ could be seen as a group with modulus a. Note that the modulus corresponds to the index of the quotient in the super-group $\mathbb{N}$. The associated operation on these elements will abide by the axioms mentioned in the previous section.

D. Finite fields

Finite fields combine the axioms in the previous sections. They are bodies that map elements outside their space to the corresponding congruence class, while still abiding by all the aforementioned axioms of a field. Finite fields are often denoted as $GF(x)$, which stands for Galois field.

Galois fields can be constructed fairly easily by taking a quotient constructed from an irreducible element in the same field. For example in the polynomial field of $GF(2)[x]$ the polynomial $x^8 + x^4 + x^3 + x + 1$ is irreducible; it cannot be factored into polynomials of a smaller degree. We can also refer to this concept as the element being prime. This concept carries over neatly into the integers: the common interpretation of prime numbers are irreducible in $\mathbb{Z}$.

So we know that if the quotient is constructed from a prime element the resulting quotient field will again be a field in itself. For example $\mathbb{Z}/p\mathbb{Z}$ is a field if p is prime.

E. Generators and Lagrange's Theorem

Generators are elements in a set that generate a so called generator set. A generator set is a set that is created by propagating an element under an operation as many times as its order in the same set. A set generated by a single element S is often denoted as $\langle S \rangle$.

For example in the group $(\mathbb{Z}^+/4\mathbb{Z}^+, +)$ the element 1 and 3 generate the whole group by repeated propagation under addition:

- $1 = 1 \ mod \ 4 = (3 + 3 + 3) \ mod \ 4$
- $2 = (1 + 1) \ mod \ 4 = (3 + 3) \ mod \ 4$
- $3 = (1 + 1 + 1) \ mod \ 4 = 3 \ mod \ 4$
- $0 = (1 + 1 + 1 + 1) \ mod \ 4 = (3 + 3 + 3 + 3) \ mod \ 4$

Consequently the order of these elements equals the size of the group: 4. The order can be considered equal to the cardinality (size) of generated set. In this case the generated set equals the entire group. Hence $(\mathbb{Z}^+/4\mathbb{Z}^+, +) = \langle 1 \rangle = \langle 4 \rangle$. The elements 2 and 0 generate subgroups of the aforementioned group, but clearly not the entirety of it. Their order is smaller. 0 has an order of 1 while 2 has an order of 2.

We can reason about the order of individual elements by applying Langrange's theorem, which states $|G| = [G : H] * |H|$ if $H < G$. We can clearly see this holds (G is the original group):

- $[G : \langle 0 \rangle] = 4$ and $|\langle 0 \rangle| = 1$, hence $|G| = [G : \langle 0 \rangle] * |\langle 0 \rangle| = 4$
- $[G : \langle 2 \rangle] = 2$ and $|\langle 2 \rangle| = 2$, hence $|G| = [G : \langle 2 \rangle] * |\langle 2 \rangle| = 4$

This theorem can also be interpret as $|G|/|H| = [G : H]$, and since we know the index is an integer this implies the size of the generator group is a divisor of the size of the original group. Equipped with this knowledge we can easily track down generators of a specific order.

F. Homomorphisms and symmetries

Homomorphisms were discussed in earlier sections, but are elaborated upon a little more here. A homomorphism is a relation between two bodies and one of their operations. Informally a homomorphism states that an operation acts very similarly in both bodies under a specific mapping. It is consequently often referred to as a "structure preserving map". A homomorphism is denoted as $f(x \cdot y) = f(x) * f(y)$, where f abides by the signature $A \rightarrow B$. In particular group homomorphisms are a homomorphisms between two groups and their corresponding operation. Morphisms span various mathematical disciplines and generalize the concept. A morphism is a form of an equivalence relation.

Homomorphisms come in various forms. A monomorphism is an injective homomorphism. An isomorphism is a bijective homomorphism. An endomorphism is a homomorphism that maps to the same domain (mathematical structure). If an endomorphism is also a bijection we refer to it as an automorphism. Automorphisms form the symmetries of a group and are essentially "structure preserving permutations".

Automorphisms are classified in two distinct classes: inner- and outer automorphisms. Inner automorphisms are formed by conjugation under an element:

$$\varphi_g(x) : gxg^{-1}$$

Outer automorphisms are defined by taking the inner automorphisms as quotient from all automorphisms:

$$Out(G) = Aut(G)/Inn(G)$$

It is easy to observe that the inner automorphisms in an integer quotient group consist of merely the trivial automorphism and hence the outer automorphisms contain the non-trivial automorphisms.

In this scheme we will exclusively work with outer automorphisms for their obfuscation properties.

IV. Homomorphic encryption in literature

This section serves as an all-encapsulating birds-eye-view of the state and developments in the homomorphic encryption scene. It also draws some connections to the current scheme and how it aims to bring the concepts into practical reality for small devices lacking the processing power of cloud networks.

Research into homomorphic encryption has experienced a surge since Gentry's influential dissertation in 2009. Since then many interpretations, mostly LWE, RLWE or lattice based (not mutually exclusive per say), have been proposed and improved upon over the years. These are now classified in their corresponding generations based on the iterative improvements made to them [42].

A. History

After the conception and publication of RSA in 1977 [36] its authors, the now famous Rivest, Shamir and Adleman, recognized its homomorphic properties over multiplication of two encrypted bundles. Only a year later they were the first recorded instance to theorize the existence of homomorphic encryption as we know it today [37]. In the short publication they describe an algorithm that would act homomorphically within an enclosed "homomorphic domain" on unspecified operations. This definition was later formalized as the operation being over those of a ring [5].

Since its theorized existence more algorithms were classified as being "partially homomorphic" (*partial* referring to the fact they don't enclose both addition and multiplication) such as ElGamal [16] or the Goldwasser-Micali cryptosystem [23], being homomorphic in multiplication and a bitwise difference (XOR) respectively. Not much headway was being made in the development of a scheme similar to described by the authors of RSA however.

That changed three decades later when in 2009 Gentry published his now influential dissertation "Fully Homomorphic Encryption Using Ideal Lattices" [2], being the first person to publish a plausible FHE (Fully Homomorphic Encryption) by modern definitions, based on earlier work by Regev [3]. A year later the scheme was practically demonstrated during EuroCrypt 2011 [19]. This opened the floodgates for a wealth of new research in the field, with lattice based solutions forming the basis for most.

It should come as no surprise a lot of iterative progress has been made since then. This lead the need for a classification system, which was provided by Vinod Vaikuntanathan in the form of incremental *generations* [42]. A total of four generations have been classified since 2009.

B. Implementations

This section is dedicated to the progression of the groundbreaking work by Gentry as well newly introduced algorithms that spawn as the generations pass by.

As mentioned in the previous section IV-A there are currently a total of four generations of FHEs. Not all schemes that are currently at the forefront of the research originate from the first. This section is once again based on V. Vaikuntanathan's classification system [42]. Works that do not abide by the modern definitions of a FHE are not mentioned in this section. For example leveled or partially homomorphic schemes such as for example Bram Cohen's simple public key algorithm [12].

In the first generation we have Gentry's dissertation "Fully Homomorphic Encryption Using Ideal Lattices". It is the flagship work of this generation. To the same generation belongs a paper that shows that Gentry's techniques of *bootstrapping* and *squashing* could be applied outside the domain of ideal lattices [41]. The basic premise is a LWE scheme that operates on the coefficient of the polynomials in (which is the same strategy applied in Rijndael as well this scheme V-D):

$$\mathbb{Z}[x]/f(x)$$

For a *prime* polynomial (irreducible polynomial) in $\mathbb{Z}[x]$ this forms a field, giving the author access to the field operations which can then be used to express a set of *complete* logic gates Γ (complete implies this would allow the author to express any circuit C in the aforementioned gates in Γ). For more details on the working on this scheme consult the next section IV-C. The scheme is an improvement of earlier work by Boneh, Goh and Nissim [8] according to Gentry himself.

As mentioned in the previous section this opened the floodgates to many new and/or iteratively improved schemes. In the second generation we see some divergence, but most schemes are still derived from lattice based cryptography similar to Gentry's. This is also a very productive generation for Gentry himself and alongside Zvkika Brakerski and Vinod Vaikuntanathan FHE is brought much closer to practical reality by weakening some security assumptions and showing the prowess of *leveled* homomorphic schemes that can potentially circumvent the very expensive boostrapping technique. This is known as the **BGV** scheme, named after its authors Brakerski, Gentry and Vaikuntanathan. Later in the same year they demonstrate this scheme by evaluating the AES circuit *without* bootstrapping [20].

The third generation shows more improvements to the relinearization step, which in term aggressively reduces the cost of bootstrapping under certain conditions. These implementations are primarily RLWE based[18]. Gentry demonstrates a "simplified" alternative to previous LWE based schemes that no longer requires an evaluation key (similar to the *public component* in this scheme, see V-D) [21]. This scheme also significantly reduces the overhead of conventional multiplication on the homomorphic domain by changing it to be more akin to matrix multiplication. They refer to the new technique as "the approximate eigenvector method".

The "unofficial" (as it isn't recognized by Vaikuntanathan) fourth generation is defined by the conception of **CKKS** [10], which is a scheme that promises much faster *scaling*; an operation that would require delinearization or boostrapping in LWE based FHEs such as BGV. The primary rationale behind the scheme being that *real-world* data often assumes the form of approximate values, inherently including the noise that would make (R)LWE work. It has also been under some scrutiny due to proposed passive attacks by a lengthy paper on the security of homomorphic schemes by Daniele Micciancio and Baiyu Li [30], to the point that users of libraries that implement it are now met with a disclaimer.

C. An evaluation of common schemes

The field of research into fully homomorphic encryption schemes has expanded rapidly in the past decades to the point of it being very difficult to provide an all-encapsulating evaluation. However, many of the aforementioned schemes originate from the same base principles. In this section I would like to briefly evaluate the implications of this strategy and how it is unlikely we will see a practical application of even the newest generation of algorithms in devices that cannot facilitate the processing power of cloud networks. In particular this section builds from Gentry's dissertation and compares it to the goals set for the scheme developed in this thesis [2][29][19]. The rationale behind opting for Gentry's, by now dated and improved scheme is because it encapsulates the most common concepts and pitfalls even in modern day FHEs. Even in the fourth generation these schemes see no use outside cloud networks and are even being described by SEAL's principal software engineer as "inefficient", while describing the use cases as "specific"[13].

Gentry's work "Fully Homomorphic Encryption Using Ideal Lattices" is what kick started the first generation [2]. It builds off constructing a ring of "coefficient polynomials" which comes equipped with the associated group operations $-\psi \leftarrow \psi_1 + \psi_2$ or $\psi \leftarrow \psi_1 * \psi_2$. (R)LWE, a generalisation of Parity learning problem, is based on introducing some amount of noise to the cipher block. This strategy thwarts linear attacks that rely on trying to retrieve the secret from a system of linear equations produced by the cipher because solving it would propagate and increase the error, making the system effectively *unsolvable*. Unfortunately the same issue applies to performing operations on it: at some point the error exceeds what the decryption algorithm would be able to detect. Equipped with no solution to this problem the scheme would only be *partially homomorphic*; it can only encode circuits of up to a specified depth.

Gentry applies the concept of *bootstrapping* to allow for the evaluation of arbitrary length circuits [3] [2]. To achieve this effect Gentry must first make sure the scheme is *bootstrappable* to begin with; implying that the expressiveness of the gate operations Γ are such that the the entirety of the decryption function can be evaluated homomorphically plus an additional NAND gate (serving as connector). On top of this requirement the complexity of the circuit needs to be such that it can be considered boostrappable in first place. Gentry refers to this as "bootstrappale with respect to Γ". Then, assuming some plain π that is encrypted under pk_1 resulting in a cipher ψ_1 we wish to refresh. We need the secret sk_1 used to decrypt psi_1 ecrypted under a second public key pk_2 denoted as sk_{1j}. After that we can define the bootstrapping procedure as:

$$Recrypt_\varepsilon(pk_2, D_\varepsilon, \langle\overline{sk_{1j}}\rangle, \psi_1) :$$

$$\overline{\psi_{1j}} \leftarrow Encrypt_\varepsilon(pk_2, \psi_{1j})$$

$$return\ Evaluate_\varepsilon(pk_2, \mathcal{D}_\varepsilon, \langle\overline{sk_{1j}, \psi_{1k}}\rangle))$$

Which Gentry describes as "An encryption under pk_2 of $Decrypt_\varepsilon(sk_1, \psi_1)$". This strategy both induces and reduces the error as it is another evaluation, but when applied strategically the decryption would reduce the size of the error vector dramatically. Since the complexity of the decryption scheme is too great to apply this strategy Gentry introduces the concept of *squashing* the circuit (which has little relevance to this section, but is still fundamental to the overall strategy). The combination of these two procedures is later applied identically in other works [41].

Unfortunately these reliniarization steps are very expensive and continue to form a big bottleneck in the first and (most of the) second generation of works. Note that in the second generation big advances have been made with regard to preventing linearization by bounded circuit depth, this was demonstrated by a homomorphic evaluation of the AES circuit[20]. Not only does bootstrapping take a full evaluation of the decryption circuit - potentially multiple times, it also requires a sequence of secret encrypted keys to be exposed to the operating (public) party. These are strikingly large as seen from the test bench we executed on SEAL (see IV-D). In 2013 Gentry, Sahai and Waters proposed a new scheme they dub the "approximate eigenvector method" [21] specifically designed around conceptual simplicity. Since it is Gentry's latest theoretical framework it is a good, probably even the best basis to compare from. This evaluation does not include the ABE solution (Attribute Based Encryption) due to a lack of relevancy. The remainder of this section serves as a guided analogy to the issues our new approach attempts to tackle (see V-A).

Recall the original definition of the Learning With Errors (LWE) problem[3]:

For a security parameter λ, let $n = n(\lambda)$ be an integer dimension. Let $q = q(\lambda) \geq 2$ be an integer and $\mathcal{X} = \mathcal{X}(\lambda)$ be a distribution over $\mathbb{Z}$. The $LWE_{n,q,\mathcal{X}}$-problem is defined as to distinguish the two following distributions: a distribution with samples $(\vec{a}_i, b_i)$ taken uniformly from $\mathbb{Z}_q^n$, and a distribution with $\vec{s} \leftarrow \mathbb{Z}_q^n$ drawn uniformly at random with $\vec{a}_i \rightarrow \mathbb{Z}_q^n$ sampled uniformly, $e_i \leftarrow \mathcal{X}$ and setting $b_i = \langle\vec{a}_1, \vec{s}\rangle + e_i$. The LWE assumption states that this problem is infeasible.

Definition 5. *The LWE problem*

This infeasibility is achieved by a reduction to a classic quantum hard lattice problem, described informally as[21][35]:

Solving an n-dimensional LWE with $poly(n)$ modulus implies an equally efficient solution to a worst-case lattice problem described by GapSVP in dimension $\sqrt{n}$.

Definition 6. *The LWE reduction theorem*

Gentry's method operates on the same basis which he refers to as a "noisy" method. To understand the naming of the method we merely have to look at the way data is encrypted. Below is Gentry's informal description:

For some modulus q and dimension parameter N, generate $C \leftarrow Z_q^{N\times N}$ uniformly, with entries "much smaller" than q. Then $\vec{v} \leftarrow \mathbb{Z}_q$ with randomly sampled entries with at least one $v_i \gg v_x, \forall v_x \in \vec{v}, x \neq i$ (at least one v_i is a significantly larger outlier). Then the message $\mu \in \mathbb{Z}_f, f \ll q$ (the message is significantly smaller than the quotient). Assume $\vec{e} \leftarrow \mathcal{X}^n$, some small error vector, then C is said to encrypt μ if

$$C * \vec{v} = \mu * \vec{v} + \vec{e}$$

Since C is a matrix clearly $\vec{v}$ is its eigenvector with eigenvalue μ differentiated only by the small error vector $\vec{e}$. Decryption is similar to Regev's PKE method[3]. Note that the decrypting party can index the outlying values in the cipher C:

$$x \leftarrow \langle C_i, \vec{v}\rangle = \mu * v_i + e_i$$
$$\mu = \lfloor x/v_i \rceil$$

For the following paragraph we must introduce the definition of a B-bounded distribution:

$$Pr_{e\leftarrow\mathcal{X}_n}[|e| > B] = negl(n)$$

Definition 7. *A B-bounded distribution*

Which can be intuitively understood as a distribution that, upon sampled, has a negligible chance of producing an error offset from 0 with a magnitude greater than B.

Consider the following homomorphic operations:

$$C^+ = C_1 + C_2, C^+ * \vec{v} = (\mu_1 + \mu_2) * \vec{v} + (\vec{e_1} + \vec{e_2})$$
$$C^\times = C_1 * C_2, C^x * \vec{v} = \mu_1 * \mu_2 * \vec{v} + \mu_2 * \vec{e_1} + C_1 * \vec{e_2} = \mu_1 * \mu_2 * \vec{v} + er$$

in which er denotes the remaining "small" error. To evaluate the level of "how homomorphic" the scheme is we need to evaluate it at a B-bound. When bound to B the magnitudes of C_i, μ_i and e_i do not exceed B. In such a case we can express the bounded error in terms of the aforementioned operations:

$$er(C^+) \leq 2B$$
$$er(C^x) \leq (N+1)B^2$$

...or expressed in terms of the evaluation of a multivariate polynomial with degree d where P represents the coefficient vector associated to the polynomial:

$$(\sum_{p\in P} |p|)(N+1)^{d-1}B^d$$

Clearly this circuit's error grows at worst according to the function

$$T_{er}(L) \mapsto B^{2^L}$$

which is *doubly exponential in the circuit depth* (or exponential in the degree of the function). Here L represents the circuit depth. The resulting scheme would fit the definition of a SWHE (SomeWhat Homomorphic Encryption). To "promote" the scheme to a FHE the authors introduce the concept of *flattening*. To understand flattening we need the definition of a strong B-bound (or B-strongly-bounded):

For some cipher C_i, if all coefficients in C and μ_i are at most 1, with the associated coefficients of $\vec{e_i}$ have a magnitude of at most B then C_i is considered B-strongly-bound.

Definition 8. *A B-strongly-bounded cipher C*

Consider a new operation, NAND, on two ciphers C_1, C_2 yielding a new cipher C_3:

$$NAND(C_1, C_2) : I_N - C_1 * C_2 = C_3$$

Using the earlier expansion of classical multiplication and the assumption of C_1 and C_2 being B-strongly-bound the resulting error is contained within $(N+1)B$, which would yield an exponential version of $T_{er}(L) = (N+1)^L B$, promoting the scheme to the classification of LFHE (Leveled Fully Homomorphic Encryption). Unfortunately this does not hold, as C_3 is no longer B-strongly-bound; it's coefficients are clearly no longer bound by just 1.

To tackle this issue flattening is applied, which utilizes various functions that have proven to be agnostic towards the dot product over vectors (i.e. the result of the dot product between two vectors is unaffected after application)[9]:

$$BitDecomp(\vec{a}) = (a_{1,0}..., a_{1,\ell-1}, ..., a_{\kappa,0}..., a_{\kappa,\ell-1})$$

Which decomposes a vector $\vec{a}$ with κ coefficients in $\mathbb{Z}_q$ into its base 2 representation of length $\ell = \lfloor log_2(q) \rfloor$. The resulting is of length N. It's inverse is defined as:

$$BitDecomp^-1(\vec{a}) = (\sum 2^j * a_{1,j}, ..., \sum 2^j a_{k,j})$$

Which can be understood at reducing the individual bit-strings back to their base 10 representation. The auxiliary function $Powersof2(\vec{b})$ is defined as:

$$Powersof2(\vec{b}) = (b1, 2b1, ...2^{\ell-1}, ..., b_k, 2b_k, ..., 2^{\ell-1}b_k)$$

yielding another N-dimensional vector. This function can be understood intuitively as "stretching" the coefficients of the input vector $\vec{b}$ into powers of 2. Then we can define $Flatten(\vec{a}')$ as

$$Flatten(\vec{a}') = BitDecomp(BitDecomp^-1(\vec{(a)}'))$$

Flattening makes the coefficients of the affected vector small, but has the additional property that it leaves the inner product with some $Powersof2(\vec{b})$ intact:

$$\langle BitDecomp(\vec{a}), Powersof2(\vec{b})\rangle = \langle \vec{a}, \vec{b}\rangle$$

To facilitate this flattening $\vec{v}$ must be picked such that $\vec{v} = Powersof2(\vec{s})$ for some secret $\vec{s}$. Clearly $Flatten(C) * \vec{v} = C * \vec{v}$ (assuming the aforementioned identity of $\vec{v}$). Now C^{NAND} can be defined as $Flatten(C_3)$, which in turn yields a cipher that is again B-strongly-bounded, effectively obtaining a Leveled Fully Homomorphic scheme.

As one might have noticed the scheme is expressed closely to the classical homomorphic operations, which makes it algebraically interesting for workloads that expose a similar circuit (for example convolution). This was an implicit demand in one our goals described in V-A. The scheme allows for a circuit evaluation up to a variable depth, which too is sufficient for our goals. The key sizes are drastically reduced due to no longer requiring any form of boostrapping. Unfortunately for our use case this approach seems infeasible due to the high amount of data inflation (as seems to be the common issue with LWE based solutions): messages can only be expressed in $\mu \ll q$ and the resulting cipher assumes the form of $C \leftarrow Z_q^{N*N}$. Even though a direct comparison cannot be made within reason (as Gentry's work is a full encryption scheme, whereas here we are describing a mere starting point) it is very unlikely to ever reach the dimensionality we deliberately sought to achieve.

D. A demonstration in SEAL

Microsoft SEAL is an open source library and implementation of BGV, CKKS and BFV [38]. In this demonstration we have opted for the use of BGV as was described in IV-B. In this example we use the *batching strategy* as described by the SEAL documentation to speed up the process. Batches are formed for similar coefficients of values ranging from 1 to 255, explicitly excluding 0. The demonstration was performed as a corroboration on the rather extreme memory differential between the size of the input data and the cipher. The testbench has been published in the research group repository[44]. Below is the depiction of the logic involved. It mimicks a single layer of a feed-forward neural network with an increase size, excluding the ReLU. It is designed to fit the description of being suitable as an application of SEAL as it is explicitly encoding logic trivially expressed in addition and multiplication. All calculations are performed on a Ryzen 9 5900x @4.2Gh with 48GB of DDR4 RAM.

Below, in Fig.2, are the results of the *increase* in dimensions of the input data. The input data consists of arrays of bytes (which is not technically true as SEAL doesn't allow for anything less than *long* encoding, but the overhead is corrected with this knowledge accordingly). The oscillations occur when the batches exceed the current lowest possible modulus, which will then, according to the documentation's specification increase by a power of two.

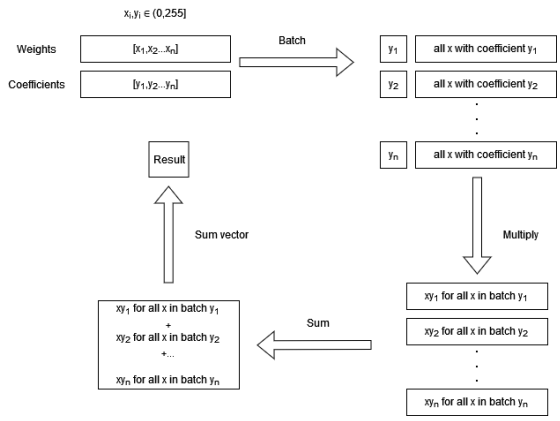

Fig. 1. The circuit executed by the SEAL testbench

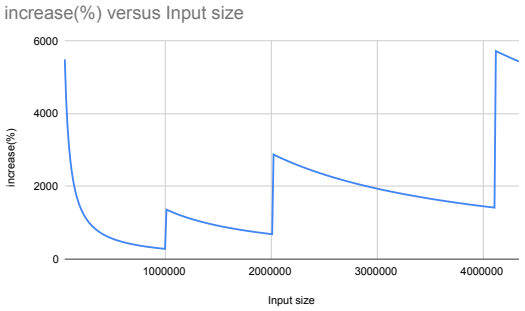

Fig. 2. Size input batches in bytes vs increase output

Even though it is not immediately apparent from the graph in Fig.2, even at the tightest fit of a single batch (close to the size the modulus allows) the output size almost quadruples (lowest point in the graph is at 282% increase).

The speed of the evaluation function and key generation is determined entirely by the modulus and displays only minute oscillations when the batch size increases. A rather speculative comparison is drawn in the following graph in Fig.3, which compares a very naive "native" implementation of the same operations to using BGV. Here we see that the run-time starts to converge as the batch size increases, but this might also very well be due to the fact that SEAL is simply much better optimized (it is best not to read too much into this graph):

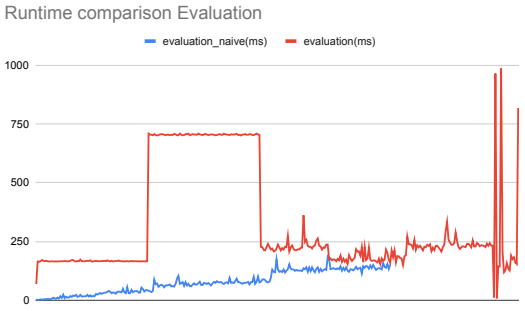

Fig. 3. Runtime comparison naive (blue) vs BGV (red)

Throttling starts to occur at the end of the graph due to the machine running out of available RAM. Upon this occurring the circuit appears to have been terminated, yielding a uncharacteristically low run-time.

V. The proposed scheme

The resulting scheme is a consequence of very specific design goals. It is *incomplete* as a cryptographic scheme but comes with potential associated improvements mentioned in section VIII-B. It is designed to accommodate little overhead above all else, making it a practical and feasible angle to homomorphic obfuscation in e.g. embedded- or edge devices.

A. The design goals

The current state of fully homomorphic encryption schemes make it wildly infeasible for practical use in small scale applications. Even though it already has some application for specific computationally simple circuits in the cloud there are a myriad of problems when applied on smaller consumer- or embedded hardware. These include problems like a large operation overhead, very large keys and a very large memory overhead. This scheme was specifically designed to be feasible in the aforementioned cases, while sacrificing many of the properties conventional FHEs offer.

Specifically the design goals were to trade in the high conventional security demands to obtain a FHE-scheme that abides by the following strict requirements (expressed in functions from the C-evaluation scheme mentioned in I):

- $|P| = |Enc(pk, m)| + \epsilon(m)$, informally "negligible co-domain bloat"
- $|P| = |Eval^*(evk, C, c_1, ..., c_n)| + \epsilon(m)$, informally "negligible evaluation bloat"
- Overhead imposed by addition and multiplication close to the native cost of said operations

While still striving for the maximum obfuscation achievable by abiding by Shannon's fundamental requirements[1] of *diffusion* and *confusion*. A combination of both is known to thwart statistical attacks and both are considered the building blocks of any cryptographic scheme. Unfortunately the current scheme does not offer proper confusion and as such should not (yet) be considered for practical use.

The summarized definitions of confusion and diffusion can be understood as was described by Shannon (these are heavily reliant of the context in which they are applied, as recognized by the authors of Rijndael [14])[1]:

- Confusion means that each binary digit (bit) of the ciphertext should depend on several parts of the key, obscuring the connections between the two
- Diffusion means that if we change a single bit of the plaintext, then about half of the bits in the ciphertext should change, and similarly, if we change one bit of the ciphertext, then about half of the plaintext bits should change

Both can be understood visually in terms of a cipher on an image as demonstrated in VII: diffusion implies that even the most marginal change in the pixel value can completely change it corresponding output. Confusion would imply that even the same pixel values are not necessarily mapped to the same output value.

Early in the scheme's design it became apparent it would be operating on the same bundle (word size) as its input. The properties of operating on a bundle level would allow for very efficient encryption and decryption which automatically abiding by the first two requirements. This design is one of the many inspirations taken from the Advanced Encryption Standard (AES, Rijndael) [14].

B. High level overview

The scheme corresponds closely to the common interpretation of a C-evaluation scheme mentioned in I, but rather than having conventional keys it has LookUp Tables (LUTs). The image blow depicts the relation between the components:

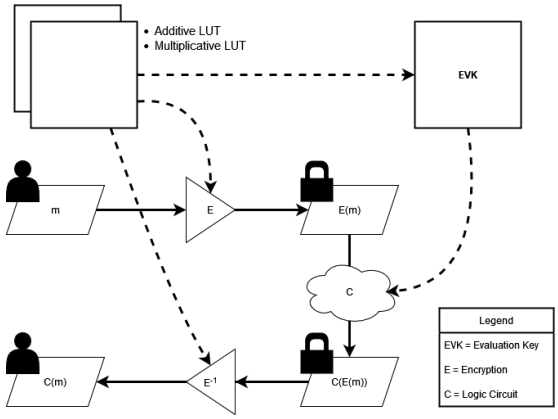

Fig. 4. A high level overview

Direct edges depict a transition, whereas dotted edges depict a one way dependency. As the image shows the *public component*, or *evaluation key* depends on the constructed lookup tables that are being used to encrypt the message. The logic circuit C can be performed remotely with only the knowledge of the evaluation key. Once the computation has been performed and decrypted the user can re-obtain their plaintext, but now transformed by the provided circuit. This process is demonstrated in VII.

C. The template

The design of the scheme is very generic, allowing for a wide range of applications of various sizes.

The basic construction can be described as follows:

1) Determine the maximum value the application needs to encode at any time and to what field or ring this value belongs
2) Find a prime that is part of the same field or group as the value mentioned in 1)
3) Construct a prime quotient field or group using the prime to construct a quotient ranging over the irreducible (prime) elements

4) Deconstruct the field or ring into its corresponding groups and determine outer automorphisms for either
5) Compose the automorphisms in such a way that would suit the circuit one would wish to encode
6) Construct the public component

In the associated implementation the scheme over constructed over the integers, but there is no inherent limitation of the underlying datatype as long as it contains a field structure and the concept of a *prime*. One could opt for polynomial evaluation or perhaps even evaluation of lattice- or vector spaces just as easily.

D. Implementation details

The scheme relies on utilizing the automorphisms of the groups that compose the closest encapsulating prime field of the largest value in the input data. This field is defined as a quotient group over the integers. More formally:

$$GF(p_e) = \mathbb{Z}/p_e\mathbb{Z}, p_e = inf(\{p \mid p \in \mathbb{Z}, p >= sup(i)\})$$

This construction consequently yields two well defined groups as both come from the generalization of the underlying ring:

$$G_+ = (\mathbb{Z}/p_e\mathbb{Z}, +)$$
$$G_* = (\mathbb{Z}/p_e\mathbb{Z}, +)$$

The construction of the latter groups is a necessity for we want to generate a permutation of automorphisms, and the original group $GF(p_e)$ has only the single trivial one.

At this point we make the observation that the underlying ring has a prime characteristic and is an integral domain. This implies the Frebenius morphism is an automorphism in this ring:

$$F(r) = r^p \in Aut((\mathbb{Z}/p_e\mathbb{Z}, *, +))$$

This gives some insight in what to expect when applying each of the individual group automorphisms in composition (hint: multiplication will be trivial, whereas addition will require additional work). Our next step is to actually define this composition. Given two automorphisms, each on their corresponding group, we define it as follows:

$$F_+(x) \in Out(G_+)$$
$$F_*(x) \in Out(G_*)$$
$$E(x) = F_* \circ F_+$$

It is worth noting that even though the composition of these functions does not technically form a group (it is not abelian), the algebraic structure of defining them “the other way around” is almost identical. The choice of the final composition is thus also chosen entirely arbitrarily.

To find both the required functions to construct e we need to find outer automorphisms for each individual group. This is achieved by sending the smallest generator of an order equal to the characteristic of the ring for each corresponding group to any other generator of an order equal to the characteristic of the ring. We parameterize the function with this ”target”. For each individual group the set of candidate generators we will define thusly:

$$CG_+(i_+, G_+) = \{g \mid gcd(g, p_e) = 1, g \neq i_+\}$$
$$CG_*(i_*, G_*) = \{g \mid ord(g) = char(GF(p_e)), g \neq i_*\}$$

Note: i here is **not** an imaginary component, it is mnemonic variable for ”input”!

Since the multiple of the quotient (the modulus) is prime it follows the entirety of the additive generator set minus the first argument (which can be fixed at 1) is available as a target generator. For the multiplicative set this is not the case, and the generators must be actively filtered on their order first. To do so we apply Legrange’s theorem to check whether each element does not yield the multiplicative identity when raised to a divisor of $p_e - 1$ that isn’t exactly $p_e - 1$.

With these functions fixed we can look into the resultant function they encode. Since we assign generators to generators these form a straightforward relation.

$$i_+ \mapsto t_+ : F^+(t_+, x) \mapsto abi = ci, ai = t, bi = x, c = ab$$
$$i_* \mapsto t_* : F^*(t_*, x) \mapsto (i^d)^e = i^h, i^e = t, i^d = x, h = ed$$

Here we expect that $t_+ \in CG_+(i_+, G_+), t_* \in CG_*(i_*, G_*)$.

The additive function encodes a multiplication, whereas the multiplicative function encodes an exponential. The resulting composition thus assumes the form:

$$E(t_+, t_*, x) = F^*(t_*, F^+(t_+, x)) \mapsto (zx)^y, z = c, y = h$$

Clearly the targets, and consequently the scalar and exponent form the secret of this simple encoding.

We now turn both the additive and multiplicative function into a lookup table, where the index of the table corresponds to the output.

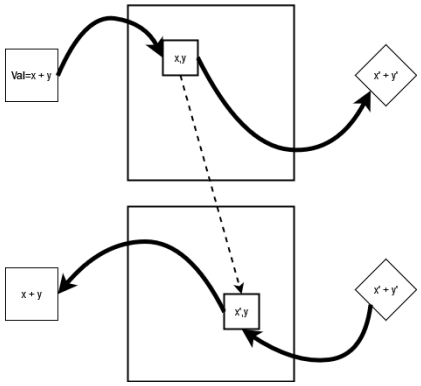

Fig. 5. Lookup tables and their corresponding inverse

We obtain the inverse function by taking the output and assigning it the value of the index it occurs at in our first table. We can now look at the algebraic properties of our

elements after having done some prior investigation. For brevity the target arguments are not denoted here.

For multiplication we derive the following identities:

$$E(a) * E(b) = (za)^y * (zb)^y = z^{2y}a^y b^y$$

$$F_*^{-1}(z^{2y}a^y b^y) = z^2 ab$$

$$F_+^{-1}(z^2 ab) = zab \neq ab$$

Clearly the answer decodes wrong, but it is easily patched by keeping track of the amount of multiplications in the circuit and then applying the appropriate amount of additional additive inverse transformations (another application in the aforementioned case). For addition things are less trivial:

$$E(a) + E(b) = (za)^y + (zb)^y$$

$$F_*^{-1}((za)^y + (zb)^y) = ((za)^y + (zb)^y)^{1/y}$$

$$F_+^{-1}(\sqrt[y]{(za)^y + (zb)^y}) = \frac{\sqrt[y]{(za)^y + (zb)^y}}{z} \neq a + b$$

The operands cannot be reduced in the first step due to being mismatched terms, and the consequence is an escalating error. One way to go about this is to simply take the yth root, but unfortunately that would imply giving away a secret. Another approach is to take the root of merely the mismatched terms expressed as a multiple of the encoded generator:

$$E(a)+E(b) = (za)^y+(zb)^y = z^y w^y g^y + z^y d^y g^y, wg = a, dg = b$$

$$P(x) = \sqrt[y]{(\frac{x}{g^y})} g^y$$

$$E(p(a))+E(p(b)) = zwg^y+zdg^y = (zw+zd)g^y, wg = a, dg = b$$

$$P^{-1}((zw + zd)g^y) = (zw + zd)^y g^y, wg = a, dg = b$$

$$F_*^{-1}((zw + zd)^y g^y) = (zw + zd)g$$

$$F_+^{-1}((zw + zd)g) = (w + d)g = a + b$$

The function p is then distributed as a public component to accommodate addition. This is done in the form of a lookup-table. This strategy guarantees the intermediate value is still parametric in the secret exponent, albeit at the cost of having to provide the aforementioned function.

These form the fundamentals of the scheme and provide observable impressive diffusion, but no confusion. Even though this is an unsolved problem potential solutions are discussed in the following section VIII-B.

E. A demonstration on the SparkFun RED-V

To corroborate the feasibility of the scheme even in the context of small scale embedded devices the template was applied and run on a SparkFun RED-V thing Plus. As the name suggests this board implements the RISC-V ISA. Central to the board is the FE310 SoC (Freedom E310), which includes the E31 CPU.

The board has 16Kb of available SRAM and offers 1.61 DMIPS/MHz operating at 150 MHz. More information about this board can be found in the attached manual[40].

In the run test-bench operations very similar to the ones described in IV-D are performed. The data-sets are either added or multiplied according to the column in both table 1 and 2, the data is then reduced under addition to yield a scalar value. These operations happen element-wise. Similar to SEAL zero encoding is explicitly forbidden and excluded from the input range. Operations are performed over 8 bit bundles with the prime quotient chosen to be 251. The tests are run over 5 differently sized randomized data sets, with the last set being specifically designed to target the largest amount of available RAM (98,1% usage). The included values are unsigned and the operations include no integer promotion at any stage; instead the modulation occurs through explicit wraparound overflowing.

The memory footprint of the LUTs scales exclusively with the choice of quotient (and hence the bundle size). For the choice of this quotient the encryption tables consume exactly 1kb, whereas the public component consumes 0.5kb.

The discrepancy between plain- and encoded data displays a clear pattern: some initial overhead is involved, but the actual performance thereafter is very comparable. I.e., the strategy scales well with circuit complexity, but has to pay for the initial footprint. The recorded data is captured in table 1 and 2. The metric of choice is in ticks. One tick corresponds to $1e-4$ seconds, or 0.1ms. Each stage is subjected to 10 runs, over which the average is calculated in conclusion.

Input size	$\blacksquare + \square$	$\blacksquare * \square + \triangle$	$\blacksquare * \square * \circ + \triangle$
4kb	2246	2276	2309
6kb	3369	3413	3460
8kb	4490	4552	4612
10kb	5611	5683	5755
12Kb	6732	6821	6909

TABLE I
PERFORMANCE EVALUATION OVER ENCODED DATA

Input size	$\blacksquare + \square$	$\blacksquare * \square + \triangle$	$\blacksquare * \square * \circ + \triangle$
4kb	32	62	92
6kb	47	92	138
8kb	62	123	184
10kb	77	148	220
12Kb	93	178	263

TABLE II
PERFORMANCE EVALUATION OVER PLAIN DATA

Table 3 displays the cost of encryption (or analogously decryption as the circuit complexity is completely identical). It would be unusual for data to be both encrypted as well evaluated on the same device, hence the separation into different tables.

Input size	$\blacksquare + \square$	$\blacksquare * \square + \triangle$	$\blacksquare * \square * \circ + \triangle$
4kb	2967	4427	5902
6kb	4450	6642	8851
8kb	5930	8856	11800
10kb	7413	11068	14754
12Kb	8895	13282	17704

TABLE III
ENCODING EVALUATION PER CIRCUIT

As was discussed in earlier sections the memory inflation is practically zero in most use cases. As the quotient was deliberately chosen to fit within the range of base 2 encodings of the bundle size ($log_2(251) < 8$) there is no inflation on the encoded date. Hence no data is presented addressing this.

VI. THEORETICAL EVALUATION

The overall complexity of the scheme depends strongly on the knowledge of the logic to be executed. This is a common theme in homomorphic encryption schemes, and often even a requirement [5]. For example in the standard implementation of a C-evaluation scheme the circuit must be fully known prior to execution (see I). Equipped with prior knowledge of the circuit one could reduce the amount of modulo operations and lookups aggressively on the computational end. This would yield operations surprisingly close to their corresponding computations on native (plain) data.

Due to its rather generic nature especially the memory overhead scales primarily in the choice of parameters. This is also discussed in the previous and coming sections.

A. Computational overhead

The computational overhead of the individual operations is very modest in comparison to homomorphic encryption schemes. The primary factor causing overhead is the appending of a modulo operation to both addition and multiplication. This can be optimized by working on larger, local intermediate values. Upon being fed to a lookup table the values need to be reduced back to elements in $GF(p_e)$. The application of the inverse of the public component can also be optimized. Below is an elaborate example of a juxtaposition between a naive and optimized approach for multiplying and summing:

$$a)\ \sum_{i=0}^{n} x_i = P^{-1}(P(x_0) + P(x_1)\ mod\ p_e) + \\ ... + P^{-1}(P(x_{n-1}) + P(x_n)\ mod\ p_e)$$

$$b)\ \sum_{i=0}^{n} x_i = P^{-1}(P(x_0) + ... + P(x_n)\ mod\ p_e)$$

Example 1. *Naive (a) versus optimized chain addition (b)*

$$a)\ \prod_{i=0}^{n} x_i = (x_0 * x_1\ mod\ p_e) * ... * (x_{n-1} * x_n\ mod\ p_e)$$

$$b)\ \prod_{i=0}^{n} x_i = (x_0 * ... * x_n\ mod\ p_e)$$

Example 2. *Naive (a) versus optimized chain multiplication (b)*

$$a)\ \sum_{i=0}^{n} x_i y_i = P^{-1}(P(x_0 * y_0\ mod\, p_e) + P(x_1 * y_1\ mod\, p_e)\ mod\, p_e) + \\ ... + P^{-1}(P(x_{n-1} * y_{n-1}\ mod\ p_e) + P(x_n * y_n\ mod\ p_e)\ mod\ p_e)$$

$$b)\ \sum_{i=0}^{n} x_i y_i = P^{-1}(P(x_1 * y_1\ mod\ p_e) + ... + P(x_n * y_n\ mod\ p_e))$$

Example 3. *Naive (a) versus optimized multiply then sum (b)*

As is evident from the above examples the user stands to gain a lot of performance knowing and optimizing their logic in advance. Compared to native operations the cipher introduces two new sources of overhead:

1) The modulo operations
2) The lookups from the public component

For each example we can reduce the complexity in which these scale.

1) For example 1 and 3
   a) Modulo operations from $\mathcal{O}(n)$ to $\mathcal{O}(1)$
   b) Inverse lookups from $\mathcal{O}(n)$ to $\mathcal{O}(1)$
2) For example 2
   a) Modulo operations from $\mathcal{O}(n)$ to $\mathcal{O}(1)$

These reductions can be achieved practically by trying to work on intermediate values for as long as possible. This in term can be achieved by not performing any lookups or alternation of operations in between.

Lookup tables are known to impose very little overhead during runtime and have an indexing complexity of just $\mathcal{O}(1)$. Since all lookup tables in this scheme are static by design they could potentially be further optimized in tailored hardware. When knowledge of the circuit is present and the operations are consistent the amount of required lookup tables to encrypt and decrypt can be halved from 4 to 2, further reducing the amount of lookups required.

B. Memory overhead

One of the strong points of the scheme is that it imposes no memory overhead on the data outside the expected range of values. Better yet, due to its inherent simplicity it can be easily adapted to the user's needs by simply taking the nearest greater prime number than they envision to encode.

On average one should expect to require one bit more than would be needed to encode the desired values natively, but this is stated without a formal proof (which would be

extremely elaborate). This is to facilitate the expected "gap" between the maximum desired value and the nearest prime greater than the aforementioned' largest value.

The total amount of bit bloat could be computed according to:

$$b = \lceil log_2(p_e) \rceil - \lceil log_2(sup(i)) \rceil$$

Further memory overhead is imposed by the requirement for lookup tables. The lookup tables are all of the size (in bits):

$$2 * sup(i) * log_2(sup(i))$$

The scaling of two is imposed by every lookup table having a corresponding inverse. When knowledge of the circuit is known beforehand and the operations are consistent the amount of lookup tables can be condensed into just E, reducing the the total amount of tables from 4 to 2. The computating side always has a single lookup table, so 2 tables in total.

C. Non-linearity and diffusion

When it comes to the design of S-boxes the design is rather meticulous, as described in [14]. Rijndael describes the design criteria for S_{RD} (**S**-box for **R**ijn**D**ael) as follows:

1) Non-linearity
   a) Correlation: The maximum input-output correlation must be as small as possible
   b) Difference propagation: the maximum difference propagation must be as small as possible
2) Algebraic complexity: the algebraic expression for S_RD has to be complex

To achieve a low input-output correlation the authors of the paper take a candidate from a paper by K. Nyberg [33] described simply as the multiplicative inverse over $GF(2)[x]$:

$$g : a \to b = a^{-1}$$

This choice is not particularly surprising and aligns quite well with the minimum input-output correlation this scheme can achieve, which will be described later in this section.

Furthermore they define a requirement in the form of finding no fixed or opposite fixed point in the resulting transformation. Described as follows respectively:

$$S_{RD}[a] \oplus a \neq 00, \forall a$$

$$S_{RD}[a] \oplus a \neq FF, \forall a$$

They then apply an affine transformation preserving the aforementioned properties to achieve point 2). Unfortunately this is not a valid strategy for this scheme because it would not preserve the desired algebraic properties described in V-D.

The input-output correlation can be expressed as the *branch* of the primitive, which in term is a measure of diffusion. For a vectorial boolean function F the branch is defined as (differential and linear respectively):

$$B_d(F) = min_{a \neq b}(W(a \oplus b) + W(F(a) \oplus F(b)))$$

$$B_l(F) = min_{\alpha \neq \beta, LAT(\alpha,\beta) \neq 0}(W(\alpha \oplus \beta) + W(F(\alpha) \oplus F(\beta)))$$

Note that an S-box can be interpreted as a vectorial boolean function of the form:

$$S : \{0,1\}^n \to \{0,1\}^m$$

Unfortunately neither of these metrics is easily quantified for this scheme, considering the size of the S-box is parametric in the choice of the size of the field. However, for demonstration purposes we can look at the differential branch level of the S-box F in $\mathbb{F}_2^4$, which is constructed by $GF(p_e), p_e = 17$.

First we fix b at 0, which results in a reduced formula:

$$B_d(F) = min_{a \neq 0}(W(a) + W(F(a))$$

Now if we assume (which is an informed choice) $t_+ = 12$ and $t_* = 6$ we end up with F:

$$F = \begin{bmatrix} 10 & 5 & 9 & 11 \\ 2 & 13 & 16 & 14 \\ 3 & 1 & 4 & 15 \\ 6 & 8 & 12 & 7 \end{bmatrix}$$

For which we find that $B_d(F) = 3$ for a word length of just 4 bits, with a theoretical upper bound of $s = 4$, where s is the number of components. This branch is frequently classified as "near-MDS" [24]. This lower bound occurs at the values 1 and 5, and their respective inverse:

$$1 : 0001 \mapsto 1010$$

$$3 : 0010 \mapsto 0101$$

A theoretical perfect score would occur when every value is sent to its respective complement.

When looking at the above construction the matrices produced are very unsurprising:

$$F_+ = \begin{bmatrix} 12 & 7 & 2 & 14 \\ 8 & 4 & 16 & 11 \\ 6 & 1 & 13 & 8 \\ 3 & 15 & 10 & 5 \end{bmatrix}$$

$$F_* = \begin{bmatrix} 1 & 9 & 6 & 13 \\ 7 & 3 & 5 & 15 \\ 2 & 12 & 14 & 10 \\ 4 & 11 & 8 & 16 \end{bmatrix}$$

F_* corresponds perfectly to the choice made by the authors of Rijndael mentioned earlier in this section, by being the multiplicative inverse of each entry. The optimal choice for the automorphism targets is determined by the *highest average Hamming distance* of all values. Which can be defined as:

$$D_W = \frac{\sum^{p_e - 1} W(x \oplus F(x))}{|\mathbb{F}_2^4|}$$

The resulting S-box has no fixed- or opposite fixed points.

Lastly, as explained in point 1) a) in the Rijndael requirements for S_{RD} we wish to minimize *difference propagation*. The latter is defined as:

$$Prob^h(a', b') = 2^{-n} \sum_a \delta(b' \oplus h(a \oplus a') \oplus h(a))$$

informally described as "for a pair chosen uniformly from the set of all pairs $(a, a*)$ where $a \oplus a* = a'$ $Prob^h(a', b')$ is the probability that $h(a) \oplus h(a*) = b'$".

When applied to the aforementioned rather small S-box we observe a maximum difference propagation of $0.4285..$, which occurs at the following differences (here C is an auxiliary classification function to match pairs of an identical difference):

$$C : W(\triangle I) \mapsto \{(1{+}0, 14{-}0), (1{+}1, 14{-}1)...(8, 7)\}, \triangle I = 4$$
$$|C(W(\triangle I))| = 7$$

The above set consists of the pairs with an input difference, expressed in the Hamming weight of the difference ($W(a \oplus b)$), of 4. There are 7 of such pairs.

In slightly less than 43% of the cases these will propagate to an output difference of 2. This property is shared among other input difference classes except the class for input difference 2, which yields a lower result at $\frac{100}{3}\%$. Since there are a total of 4 different input differential classes available the best theoretical case for a random function would be 25%, a perfect distribution. This implies there is unfortunately a very noticeable difference propagation.

D. Security

As mentioned in the earlier sections, in particular V-A, this scheme was not designed with security as a priority. This section will elaborate upon its rather weak properties in this regard.

Relative security is achieved when the workload of an attack is no better than an exhaustive key search on the scheme. Actual security is achieved when this also becomes intractable to execute for the attacker. Actual "proofs" of security do not exist in cryptography because the existence of such a proof would imply $P \neq NP$; instead we propose it as an *assumption* [28]. It is a common practice in cryptography to build ciphers around the concept of a computationally hard problem, or one that reduces to a hard problem, to facilitate this desired infeasibility [22]. This scheme does, as was previously stated, not abide by this strategy and does not achieve true security, potentially not even relative security in its current form.

The security of the scheme relies on the secrecy of the lookup tables to encrypt and decrypt, which effectively makes it a symmetric scheme. It is also partially hybrid due to having to expose a public table to facilitate addition, which is supposedly shared at the start of a session.

The scheme has many security flaws and is extremely prone to interpolation attacks in its current state. The exposed function is parametric in only a *single* of the two secrets, potentially weakening the scheme further. Interpolation attacks were first mounted in 1997 by Thomas Jakobsen & Lars Knudsen and describe a situation almost identical to the form this scheme assumes [26]. Unfortunately there appears to be no straightforward way to patch this. It ties into the second requirement of S_{RD} described in VI-C. The reality of the situation is that the S-boxes in this scheme need to maintain their algebraic structure to preserve the homomorphic properties. As such interpolation attacks are going to remain a prominent attack vector.

The scheme in its naive form acts like a substitution cipher, implying it inherits all weaknesses associated to such a cipher. Specifically frequency analysis is a straightforward attack vector [39]. But being a homomorphic scheme and relying solely on an S-box substitution it also inherits the issue of *zero encoding*. Zero, either as input or as intermediary value, will always be propagated as zero.

No security evaluation would be complete without considering linear and differential crypt analysis. The first recorded demonstration of linear crypt analysis was performed by M. Matsui and A. Yamagishi in 1993 [32]. The basic concept of linear crypt analysis is to find the linear relation between plain and cipher, yielding a system of linear equations that would be easily solvable. Naturally schemes with a large degree of linearity (which might also occur under certain *weak keys*) are vulnerable to this as the system could end up in a practically feasible size. Differential cryptanalysis instead relies on finding the difference between input and output and trying to expose non-random behaviour. The most common interpretation is as described by E.Bahim and A. Shamir[31]. Since it relies on chosen plaintexts it is not particularly relevant for this scheme.

Like every other encryption scheme this one is prone to *weak keys*. Keys can be classified to a class of weak keys under a specific angle of attack [7]. Weak keys in this scheme include for example reflections (which ironically yield the highest Hamming-diffusion). Since inner automorphism are explicitly excluded they are not part of any class of weak keys.

In terms of exhaustive search the space of possible permutations would be equal to:

$$CG_+ * CG_* \leq sup(i)^2$$

or in case the confusion as proposed in VIII-B is applied:

$$CG_+ * CG_* * A$$

Which implies we could model the computational security as follows:

$$\sqrt{\kappa} = log_2(sup(i)), E = \circ^2 f_o, o \in \{+, *\}$$

Note: the aforementioned notation of E is written as a composition of the available automorphisms.

The resulting amount of computational effort would then be expressed as:

$$O(2^{\kappa})$$

In a very esoteric situation one could also opt to expand the composition of E, but this is very unlikely to yield either meaningful or algebraically simple solutions. This is explored briefly in VIII-B.

Clearly the security scales quadratically with the length of the input. Assuming the scheme would be perfectly secure this would yield an axis of optimisation. In case a third operation would be applied to the composition of e this would be much more lenient.

VII. Practical examples

This section is entirely dedicated to some worked examples of the involved theory. They are explicitly made very "visually appealing" and involve little to no opaque concepts. Both demonstrations in this sections can be performed locally using the provided API and corresponding example source files.

A. Implementing a simple scheme

Assume a situation in which we would wish to operate on values of 4 bits. We take the template as described in V-C as blueprint.

1) We want to work on 4 bits, so our largest value would need to be able to encode $2^4 - 1 = 15$
2) We want to work on integers, so we seek the closest greater prime than 15. This is 17
3) The prime quotient field would be $F = \mathbb{Z}/17\mathbb{Z}$ or $M = \mathbb{Z}\ mod\ 17$
4) The corresponding groups are $(M, +)$ and $(M, *)$

Now we wish to actually find the automorphisms. To find an automorphism with a high degree of diffusion we look for the *highest average Hamming distance*. This can be computed automatically by the provided API. In our case this yields:

$$t_+ = 12, t_* = 6$$

which will send the smallest generator of the same order to the aforementioned value. 1 in case of the additive automorphism, and 3 in case of the multiplicative automorphism. Equipped with the automorphisms we can construct the corresponding S-boxes (identical to the ones we see in VI-C):

$$F_+ = \begin{bmatrix} 12 & 7 & 2 & 14 \\ 8 & 4 & 16 & 11 \\ 6 & 1 & 13 & 8 \\ 3 & 15 & 10 & 5 \end{bmatrix}$$

$$F_* = \begin{bmatrix} 1 & 9 & 6 & 13 \\ 7 & 3 & 5 & 15 \\ 2 & 12 & 14 & 10 \\ 4 & 11 & 8 & 16 \end{bmatrix}$$

We now wish to perform a matrix multiplication and addition. We start with the following matrices:

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$b = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$c = \begin{bmatrix} 2 & 4 & 2 \\ 3 & 5 & 3 \\ 4 & 6 & 4 \end{bmatrix}$$

The operation we wish to perform is $a * b + c$. We will refer to this operation as our *circuit*. First we want to take a look at whether anything needs optimizing before we perform the calculation. In this particular case the operations are simple enough nothing can be improved from the native computation. We do however notice that the *factor level* of the c matrix will be incompatible with the intermediary result of multiplying $a * b$. We can alleviate this issue by explicitly raising c to a second factor level. This can be done through the available API.

We then proceed to construct the *public component*. Once again this is automated by the provided API. The result is another S-box we distribute to the computing party. The public component is calculated from F_* and looks like this:

$$P = \begin{bmatrix} 8 & 13 & 12 & 2 \\ 14 & 11 & 7 & 16 \\ 1 & 10 & 6 & 3 \\ 15 & 5 & 4 & 9 \end{bmatrix}$$

After having done so we can start to actually encrypt our matrices and ready them for transmission! As was described in the previous paragraph we will explicitly raise matrix c to a higher factor level. The resulting computation looks like this:

$$c_a = F_*(6, F_+(12, a))$$

$$c_b = F_*(6, F_+(12, a))$$

$$c_c = F_*(6, F_+(12, F_+(12, a)))$$

The obfuscated matrices resulting from the aforementioned transformations assume the following form:

$$c_a = \begin{bmatrix} 14 & 10 & 13 \\ 12 & 1 & 2 \\ 9 & 11 & 6 \end{bmatrix}$$

$$c_b = \begin{bmatrix} 14 & 0 & 0 \\ 0 & 13 & 0 \\ 0 & 0 & 14 \end{bmatrix}$$

$$c_c = \begin{bmatrix} 4 & 15 & 4 \\ 12 & 14 & 12 \\ 13 & 11 & 15 \end{bmatrix}$$

The example displays a prominent issue: *zero encoding*! Since no automorphism can map zero to anything but itself,

zero will be propagated as zero always. It is advisable to try and prevent sparse matrices from occurring where possible, lest the information isn't too confidential.

After applying the circuit and decoding it according to the appropriate factor level we obtain the result as expected:

$$result = \begin{bmatrix} 3 & 10 & 5 \\ 7 & 3 & 9 \\ 11 & 13 & 13 \end{bmatrix}$$

This brief demonstration was initially conceived to demonstrate how to use the simple API, and can be found in the attached repository under **Matrix_example.py**.

B. Image convolution

Image convolution is an operation derived from mathematical convolution, which in term is defined as:

$$(F \circ G)(x) \int_{-\infty}^{\infty} F(\tau)G(x-\tau)\, d\tau$$

Intuitively it can be understood as a measure of "similarity" in two functions. In the above example G is flipped upside down (reflection over the y-axis) and then the overlap between the product is integrated. Convolution and cross correlation have a very wide range of applications, including image process and artificial intelligence [46] [11]. In this particular example we will take a look at image convolution through a kernel.

Convolution is a very straightforward use case of homomorphic schemes because it is easily expressed in addition and multiplication. Similar use cases include statistics and numerical methods. For this scheme's particular case it is also very easy to optimize, allowing for performance almost identical to native operations.

We start this example with one image taken from Quake. Quake is a title developed by ID Software in 1996 [27]. It is renowned for many reasons but was also critiqued for it's rather "muddy and brown" color palette. The latter makes it an excellent title to take as example for this particular demonstration. The image we will perform convolution on is shown in Fig.6.

The image kernel we will apply is known as the "embross kernel" and is depicted below:

$$K = \begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

For convolution to perform as expected we need intermediate values significantly larger than the bytes that encode the RGB values. This is due to the fact that comparisons are very expensive once the data has been transformed to the cipher domain, implying that just performing a local bound operation ($max(x, y)$ or $min(x, y)$) should be considered impractical. To facilitate these larger intermediate values we construct our scheme from $GF(p_e), pe = 2039$, or F_2^{11}.

Fig. 6. A screenshot taken from Quake

We then apply the transformation, yielding a cipher with some notable flaws/features:

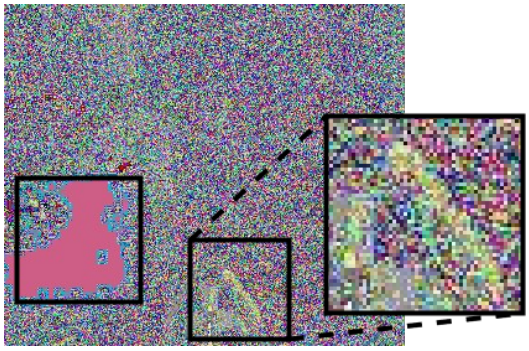

Fig. 7. The screenshot after transformation

Most of the image turns into white noise, but clearly it leaves some recognizable features intact. All of these originate from the lack of *confusion* in the current scheme. The grenade-launcher (the weapon at the bottom of the screen) has contours consisting of equally valued pixels, which translate to the same values in the cipher. The same problem arises much more prominently when looking at the particles on the left of the image. But when looking at the image some closer we can start to distinguish more shapes suffering the same issue.

This makes the image we obtain *after* convolution even more interesting, because it introduces a form of "local confusion" by introducing the aforementioned similarity of neighbouring pixels. The cipher after convolution is in Fig.8.

Many of the features recognizable in the previous image have now faded, excepting the big black blob (dark grey, in fact) that encodes the particles. As explained previously this effect is caused by the aggregation of neighbouring pixels the kernel imposes. Unfortunately the kernel dimensions are too modest to overcome the blob, which has merely shrunk in size.

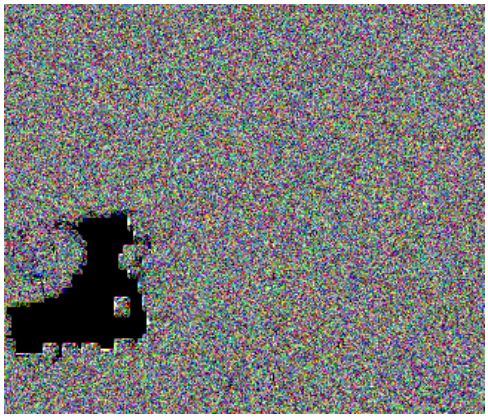

Fig. 8. The cipher after convolution

Upon decoding the above image we obtain the recognizable image after the application of the embross kernel, which is an easily recognizable effect:

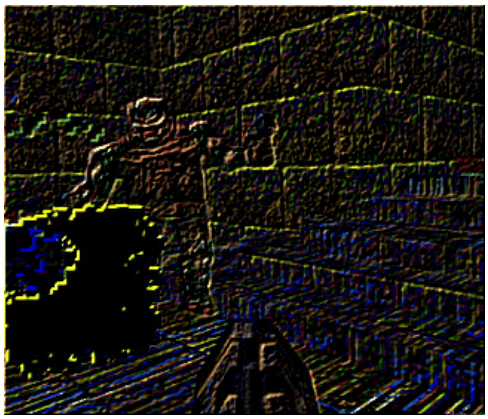

Fig. 9. The convoluted cipher after decoding

This example was part of a demonstration. It can be performed locally with the file **Imageconvolution_example.py** To view this demonstration in real time please take a look at the attached reference [43].

C. The associated library

At the time of writing the library has not been published publicly. It is a single (double in case you would also want to export generated S-boxes to a statically compiled language) file library with minimal dependencies. In case of interest please contact the CAES-group at the University of Twente.

VIII. Conclusion

This thesis set out to scrutinize an unorthodox angle to a well-known problem: can we achieve the homomorphic properties of FHEs at a theoretical overhead low enough to make it practically feasible to implement on devices with little processing power? In doing so equally unorthodox sacrifices would be made in the form of conventional security requirements and quantum security. Even though the scheme is not a complete cryptosystem in its current form it meets those requirements and might offer some insight into avenues other authors might not yet have considered.

A. The achievements

Despite its many shortcomings as a full product, the primitive offers some remarkable features. The amount of overhead when operating on the cipher (the $Eval$ function in the C-evaluation scheme discussed in I) is very close to the native cost of the same operations. Due to it being a very simple, generic template above all else, it can be applied to operate on many domains, as long as they offer a ring structure and some concept of an *irreducible element* or *prime*.

The overall computational overhead imposed by the scheme is entirely dictated by the use of the *public component* (which acts a lookup table) and the modulo operator associated to each. As was described in VI-A, even these operations can be optimized aggressively provided the user has an intimate knowledge of the circuit they wish to apply. Fortunately in all devices that would benefit from this implementation that follows almost naturally. The memory overhead imposed on the input data, as was described in VI-B, is negligible, increasing the length of the boolean vector by at most a mere bit.

Despite not always being easy to evaluate by conventional metrics the scheme performs admirably on some of them (VI-C). Though not offering any form of confusion the diffusion of the scheme is high, opening potential new angles to explore such as the use in *reliability schemes*.

The trade-offs are also significant: offering only a small key space to explore for exhaustive key search (see V-D) and not being a complete cryptographic scheme due to lacking any *confusion* properties. On top of that the initial memory footprint is marred by the need to store the lookup tables required for encryption and decryption.

B. Future work

This section is dedicated primarily to conjecture about potential improvements that could be made to the scheme. It includes little to no corroboration on the effect of any such changes and should be treated as pure speculation.

Even though rather simple, the current implementation does certainly not lack its share of issues. As mentioned in V-A it offers no *confusion* whatsoever. Confusion is defined as having different key components occur at parts of the cipher, ”confusing” the relation between the index and the key. Whereas diffusion is commonly achieved through nonlinear functions in the form of s-boxes [31], confusion is commonly achieved by permuting or mixing.

Confusion is achieved by some form of *permuting*, which in this scheme’s case is most easily achieved by mixing of some kind. Permuting also allows for covering a larger part of the key space, which is an additional bonus. Since no confusion

has been achieved with the current incomplete scheme the effectiveness will not be evaluated. A proposition would be to turn the scheme into an effective *block cipher* similar to Rijndael, as opposed to its current form as a substitution cipher. Blocks of a variable size would be encoded by sequentially mixed layers of the current S-boxes. The concept of such a modification is depicted below:

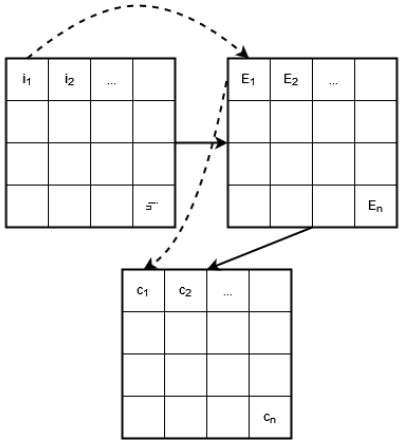

Fig. 10. Confusion through permutating blocks

In this depiction, one would expect to mix the second box filled with differently parameterized compositions as described in the previous section. This would not sacrifice any parallelism but would cause severe overhead in terms of the initial memory footprint. It would not cause any overhead on the actual computational complexity either. It could diminish the diffusion properties on a local scale due to the need for a larger spanning of the key space (which, as mentioned earlier, is a double-edged sword). It would serve as protection against the scheme's strongest attack vector: interpolation attacks. The concept would collide with the need for the public component. It would also cause the scheme to be harder to evaluate from a security perspective.

The public component could potentially be eliminated if knowledge of the circuit before execution is made a strict requirement on the user's end. This is not uncommon in current FHE libraries such as FHELib.

Lastly one could increase the span of the compositions by adding another function to it, but finding such a function with nice algebraic properties is surprisingly difficult. The most logical candidate would be a simple addition, but unfortunately, it has very poor algebraic properties. Currently, the composition function E only spans a very small part of the symmetric group S_{p_e}, which is an issue considering the strength of the scheme relies on its secrecy.

REFERENCES

[1] A mathematical theory of cryptography. 1945.

[2] *Fully Homomorphic Encryption Using Ideal Lattices*. PhD thesis, 2009.

[3] On lattices, learning with errors, random linear codes, and cryptography. *Journal Of The ACM*, 56(6):1–40, 2009.

[4] "hetest: A homomorphic encryption testing framework". *Financial Cryptography And Data Security*, pages 213–230, 2015.

[5] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

[6] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A. Reuter, and Martin Strand. A guide to fully homomorphic encryption. Cryptology ePrint Archive, Paper 2015/1192, 2015. https://eprint.iacr.org/2015/1192.

[7] Alex Biryukov. *Weak Keys*, pages 1366–1367. Springer US, Boston, MA, 2011.

[8] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In Joe Kilian, editor, *Theory of Cryptography*, pages 325–341, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[9] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. Cryptology ePrint Archive, Paper 2011/344, 2011. https://eprint.iacr.org/2011/344.

[10] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. Cryptology ePrint Archive, Paper 2016/421, 2016. https://eprint.iacr.org/2016/421.

[11] Nikolay I. Chervyakov, Pavel Alekseevich Lyakhov, Maxim Anatolievich Deryabin, N. N. Nagornov, Maria Vasilyevna Valueva, and Georgii V. Valuev. Residue number system-based solution for reducing the hardware cost of a convolutional neural network. *Neurocomputing*, 407:439–453, 2020.

[12] Bram Cohen. A simple public key algorithm. http://bramcohen.com/simple_public_key.html, 1998.

[13] Jose Daniel Contreras. Homomorphic encryption with seal. https://learn.microsoft.com/en-us/azure/architecture/solution-ideas/articles/homomorphic-encryption-seal, 2019.

[14] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer Berlin Heidelberg, 2020.

[15] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally distributed content delivery. *IEEE Internet Computing*, 6(5):50–58, 2002.

[16] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[17] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. https://eprint.iacr.org/2012/144.

[18] Craig Gentry. Computing on the edge of chaos: Structure and randomness in encrypted computation. Cryptology ePrint Archive, Paper 2014/610, 2014. https://eprint.iacr.org/2014/610.

[19] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. Cryptology ePrint Archive, Paper 2010/520, 2010. https://eprint.iacr.org/2010/520.

[20] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the aes circuit. Cryptology ePrint Archive, Paper 2012/099, 2012. https://eprint.iacr.org/2012/099.

[21] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. Cryptology ePrint Archive, Paper 2013/340, 2013. https://eprint.iacr.org/2013/340.

[22] Oded Goldreich. *Computational complexity*. Cambridge University Press, Cambridge, England, April 2008.

[23] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, page 365–377, New York, NY, USA, 1982. Association for Computing Machinery.

[24] Kishan Chand Gupta, Sumit Kumar Pandey, and Susanta Samanta. On the construction of near-MDS matrices. *Cryptography and Communications*, aug 2023.

[25] Shai Halevi and Victor Shoup. Design and implementation of helib: a homomorphic encryption library. Cryptology ePrint Archive, Paper 2020/1481, 2020. https://eprint.iacr.org/2020/1481.

[26] Thomas Jakobsen and Lars R. Knudsen. The interpolation attack on block ciphers. In Eli Biham, editor, *Fast Software Encryption*, pages 28–40, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[27] John Cash John Carmack, Micheal Abrash. Quake. https://www.idsoftware.com, 1996.

[28] Dima Kogan. Topics in cryptography. University course, 2023.

[29] Eric Levieil and David Naccache. Cryptographic test correction. In Ronald Cramer, editor, *Public Key Cryptography – PKC 2008*, pages 85–100, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[30] Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 648–677, Cham, 2021. Springer International Publishing.

[31] Rijmen Vincent Leander Gregor Liu, Yunwen. Nonlinear diffusion layers. Technical report, 2018.

[32] Mitsuru Matsui and Atsuhiro Yamagishi. A new method for known plaintext attack of feal cipher. In Rainer A. Rueppel, editor, *Advances in Cryptology — EUROCRYPT' 92*, pages 81–91, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[33] Kaisa Nyberg. Differentially uniform mappings for cryptography. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, pages 55–64, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[34] V. Pathak. Notes on lattices, homomorphic encryption, and ckks. 2022.

[35] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: Extended abstract. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 333–342, New York, NY, USA, 2009. Association for Computing Machinery.

[36] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, feb 1978.

[37] Ronald L. Rivest and Michael L. Dertouzos. On data banks and privacy homomorphisms. 1978.

[38] Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL, January 2023. Microsoft Research, Redmond, WA.

[39] Simon Singh. The black chamber. https://www.simonsingh.net/The$_B lack_C hamber/hintsandtips.html$, 2010.

[40] SparkFun. Sifive e31 core complex manual. https://static.dev.sifive.com/SiFive-E31-Manual-v2p0.pdf, 2018.

[41] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. Cryptology ePrint Archive, Paper 2009/616, 2009. https://eprint.iacr.org/2009/616.

[42] Daniele Miccianico Vinod Vaikuntanathan. Homomorphic encryption references. https://people.csail.mit.edu/vinodv/FHE/FHE-refs.html, 2023.

[43] Lars Willemsen. Image convolution real time example. https://www.youtube.com/watch?v=ECzyaqurFs0, 2023.

[44] Lars Willemsen. Seal test bench .net. https://gitlab.utwente.nl/dcs-group/SEALTestbench, 2023.

[45] T. Ye, S. R. Kuppannagari, R. Kannan, and V. K. Prasanna. Performance modeling and fpga acceleration of homomorphic encrypted convolution. pages 115–121, sep 2021.

[46] Yingjie Zhang, Geok-Soon Hong, Ye Dongsen, J. Fuh, and Kunpeng Zhu. Powder-bed fusion process monitoring by machine vision with hybrid convolutional neural networks. *IEEE Transactions on Industrial Informatics*, PP:1–1, 11 2019.