Pasta: A Case for Hybrid Homomorphic Encryption

Christoph Dobraunig[4], Lorenzo Grassi[3], Lukas Helminger[1,2] Christian Rechberger[1], Markus Schofnegger[1] and Roman Walch[1,2]

[1] IAIK, Graz University of Technology
firstname.lastname@iaik.tugraz.at
[2] Know-Center GmbH, Graz, Austria
[3] Digital Security Group, Radboud University, Nijmegen
lgrassi@science.ru.nl
[4] Lamarr Security Research, Graz, Austria
christoph.dobraunig@lamarr.at

Abstract. The idea of hybrid homomorphic encryption (HHE) is to drastically reduce bandwidth requirements when using homomorphic encryption (HE) at the cost of more expensive computations in the encrypted domain. To this end, various dedicated schemes for symmetric encryption have already been proposed. However it is still unclear if those ideas are already practically useful, because *(1)* no cost-benefit analysis was done for use cases and *(2)* very few implementations are publicly available.
We address this situation in several ways. After we formally define HHE in a broader sense than before, we build an open-source benchmarking framework involving several use cases covering three popular libraries.
Using this framework, we explore properties of the respective HHE proposals. It turns out that even medium-sized use cases are infeasible, especially when involving integer arithmetic.
Consequently, we propose Pasta, a cipher thoroughly optimized for integer HHE use cases. Pasta is designed to minimize the multiplicative depth, while also leveraging the structure of both state-of-the-art integer HE schemes (BFV and BGV) to minimize the homomorphic evaluation latency. Using our new benchmarking environment, we extensively evaluate Pasta in SEAL and HElib and compare its properties to 7 existing ciphers in two use cases. Our evaluations show that Pasta outperforms its competitors for HHE both in terms of homomorphic evaluation time and noise consumption, showing its efficiency for applications in real-world HE use cases. Concretely, Pasta outperforms Agrasta by a factor of up to 82 and Masta by a factor of up to 6 when applied to the two use cases.

Keywords: homomorphic encryption · hybrid homomorphic encryption · Pasta · SEAL · HElib · TFHE

1 Introduction

In recent years, people have become increasingly concerned about the privacy of their data, and new regulations like the General Data Protection Regulation (GDPR)[1] forbid sharing and processing sensitive data. However, many applications, such as machine learning and statistics, require a vast amount of data to be as accurate as possible. With GDPR and similar regulations it is therefore difficult or even impossible to gather enough data to create useful and accurate models. One solution to this problem is employing privacy-preserving

[1]https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=celex:32016R0679

cryptographic protocols and primitives, such as secure multi-party computation (MPC) or homomorphic encryption (HE). Homomorphic encryption schemes allow performing computations on encrypted data without having access to the secret decryption key. Many privacy-preserving applications which employ homomorphic encryption use the following design principle: First, the data holder encrypts their dataset using a homomorphic encryption scheme and sends the ciphertexts to a server. The server then performs the computations on the ciphertexts and produces an encrypted result. Only the data holder knows the secret decryption key, so the server has to send the encrypted result to the data holder who can then decrypt it to get the final result of the computation.

While this approach protects both the privacy of the input data and the secrecy of the applied computations, it comes with several drawbacks: First, applying homomorphic encryption results in a drastic performance penalty. Secondly, HE schemes suffer from ciphertext expansion. This means that the ciphertexts in HE schemes are several orders of magnitude larger than the corresponding plaintexts. This expansion negatively impacts the amount of data which has to be transferred from the data holder to the server. Especially on embedded devices, with limited bandwidth, memory, and computing power, this expansion can have a considerable impact on the overall performance of the application.

The academic literature proposes two orthogonal solutions to this ciphertext expansion: Using symmetric ciphers in hybrid homomorphic encryption, or using LWE encryption and efficient conversion algorithms [CDKS20].

1.1 Hybrid Homomorphic Encryption (HHE)

Hybrid homomorphic encryption was first mentioned in [NLV11]. The main idea behind HHE is the following: Instead of encrypting the data with HE schemes, encrypt the data with a symmetric cipher (expansion factor of 1) and send the symmetric ciphertexts to the server. The server then first homomorphically performs the symmetric decryption circuit to transform the symmetric ciphertext into a homomorphic ciphertext and then proceeds with performing the actual computations. This procedure requires that the data holder first sends the symmetric key encrypted under homomorphic encryption to the server.

At first, researchers tried to evaluate existing ciphers, like AES [DR00, DR02], with homomorphic encryption [GHS12, CCK+13, CLT14]. However, despite their plain efficiency, existing ciphers were not well-suited for HHE. Especially their large multiplicative depth deemed to be incompatible with modern HE schemes. As a consequence, researchers came up with symmetric cipher designs [ARS+15, CCF+16, DEG+18, MCJS19, HL20] with different optimization criteria compared to, e.g., AES, mainly minimizing the multiplicative depth to be efficiently computable under HE.

Today's homomorphic encryption schemes naturally operate on plaintexts in $\mathbb{Z}_q$. In most applications, it is desirable to either choose $q = 2$ or q to be a prime number, with the majority of statistics and machine learning applications requiring the latter [BBH+20, JVC18, BCD+20b]. Once chosen, however, it is not possible to convert a homomorphic ciphertext encrypting plaintexts in $\mathbb{Z}_{q_1}$ to a ciphertext encrypting plaintexts in $\mathbb{Z}_{q_2}$ for $q_1 \neq q_2$. Most state-of-the-art symmetric ciphers for HHE are designed over $\mathbb{Z}_2$, which is why one has to build binary circuits to realize integer HHE use cases. For this reason, using HHE in use cases over $\mathbb{F}_p$ already implies a heavy performance loss compared to just implementing the use case with homomorphic encryption. In this paper, we therefore approach the following issues: Can we efficiently and securely realize hybrid homomorphic encryption over $\mathbb{F}_p$? What is the performance gain compared to state-of-the-art HHE over $\mathbb{Z}_2$?

1.2 Contribution

Our contribution is the following.

- Based on previous definitions in the literature, we are the first to formally define hybrid homomorphic encryption for arbitrary use cases.
- To the best of our knowledge, we are the first to provide an extensive comparison of different symmetric ciphers in the context of hybrid homomorphic encryption. Notably, this increases the number of publicly implemented HHE schemes from only one to a total of 16, aiding public verifiability.[2] We come to the conclusion that most existing designs are not well-suited for large classes of use cases.
- We propose a new symmetric cipher, dubbed Pasta, optimized for integer HHE use cases. Pasta is defined to operate on plaintexts in $\mathbb{F}_p^t$, greatly increasing the performance compared to most previously proposed symmetric ciphers which are defined over $\mathbb{Z}_2$. Further, Pasta is designed to make use of the structure of both state-of-the-art integer HE cryptosystems (BFV and BGV) to minimize HHE decompression latency while still maintaining a small number of rounds and multiplicative depth. Our extensive benchmarks confirm the advantage of Pasta compared to all other symmetric ciphers for HHE. Concretely, Pasta outperforms Agrasta, the currently fastest $\mathbb{Z}_2$ cipher for HHE, by a factor of 82 when applied to a small use case in HElib, and it outperforms Masta, the single $\mathbb{F}_p^t$ contender, by a factor of up to 6 when applied to a larger use case in SEAL.

1.3 Outline

In Section 2, we first recall homomorphic encryption, before we formally define hybrid homomorphic encryption in Section 3. In Section 4, we evaluate the usability of existing cipher designs for HHE. In Section 5, we then explore the design space for efficient ciphers for HHE over $\mathbb{F}_p$, before we give the full specification of our new cipher, dubbed Pasta, in Section 6. In Section 7, we then analyze the security of Pasta, followed by benchmarks of Pasta in comparison to existing ciphers for HHE in Section 8. Finally, we conclude the paper in Section 9.

1.4 Notation

Let $t \geq 1$. For each vector $\vec{x} \in \mathbb{F}_p^{2t}$ and $\vec{y} \in \mathbb{F}_p^t$, we use the following notations.

1. $\vec{x} := \vec{x}_L \| \vec{x}_R \in \mathbb{F}_p^{2t}$, where $\vec{x}_L, \vec{x}_R \in \mathbb{F}_p^t$ are respectively the left and the right t words.
2. $\vec{x} := x_0 \| x_1 \| \ldots \| x_{2t-1} \in \mathbb{F}_p^{2t}$, where $x_i \in \mathbb{F}_p$ is the i-th word. The same holds for $\vec{x} \in \mathbb{F}_p^t$.
3. We write $\texttt{rot}_i(\vec{y})$ to indicate a rotation of the vector $\vec{y} \in \mathbb{F}_p^t$ by i steps to the left.
4. We denote by $\vec{y} \circ \vec{m}$ the element-wise product (Hadamard product) between two vectors $\vec{y}, \vec{m} \in \mathbb{F}_p^t$

[2]All our implementations are open source and available at https://github.com/IAIK/hybrid-HE-framework.git.

2 Homomorphic Encryption

Homomorphic encryption has often been labeled the holy grail of cryptography, since it allows to perform any computation on encrypted data without knowledge of the secret decryption key. The concept of HE was introduced by Rivest et al. [RAD78], but first schemes were only capable of performing one specific operation on encrypted data (e.g., multiplication with RSA [RSA78], addition with Paillier [Pai99]). The breakthrough came with Gentry's work from 2009 [Gen09], showing the first fully homomorphic encryption (FHE) scheme which in theory can perform any computation on encrypted data. Although deemed impractical, this work led the way for many improvements and follow-up publications [Bra12, FV12, BGV12, CGGI20, CKKS17].

Today's HE schemes base their security on the *learning with errors* (LWE) hardness assumption [Reg05], and its optimization over polynomial rings (Ring-LWE, or R-LWE) [LPR10]. In these schemes, random Gaussian noise is added during the encryption process. A homomorphic operation then increases this noise, negligible for homomorphic addition, but significant for homomorphic multiplication. Once the noise exceeds a specific threshold, the decryption will fail. The resulting schemes, therefore, allow the evaluation of arbitrary circuits over encrypted data up to a specific multiplicative depth which depends on the encryption parameters. Such a scheme is called a *somewhat homomorphic encryption* (SHE) scheme. In general, increasing the parameters to support a bigger circuit depth comes with a great performance penalty.

In [Gen09], Gentry introduced the bootstrapping technique, a method to reset the noise in a homomorphic ciphertext. Bootstrapping allows to evaluate circuits of arbitrary depth on encrypted data and turns a (bootstrappable) SHE scheme into an FHE scheme. However, bootstrapping comes with a significant performance overhead, which is why it is often faster to choose an SHE scheme with sufficiently large parameters.

2.1 Packing

Many modern HE schemes allow to encode a vector of n plaintexts into only one polynomial, and therefore, encrypt a vector into only one ciphertext. Thereby, the size of the ciphertext does not depend on the exact number of slots ($\leq n$) of the vector filled during encryption. Homomorphic operations on the ciphertexts then correspond to element-wise operations on the encrypted vector. This packing is similar to single-instruction-multiple-date (SIMD) instructions on modern CPUs and can be used to massively increase the throughput and decrease the ciphertext expansion of HE applications. Operations supported by this packing include addition, subtraction, multiplication, and slot rotation. However, once encrypted, one cannot directly access individual slots of the encrypted vector. The number of slots[3] n available depends on the parameters of the HE scheme and can range up to several thousand slots.

Slot rotation is implemented by evaluating Galois automorphisms $\tau_i : a(X) \mapsto a(X^i)$ on encoded polynomials. Since Galois automorphisms are by definition homomorphic to addition and multiplication, one can evaluate these automorphisms on homomorphic ciphertexts $c = (c_0, c_1)$ by evaluating these automorphisms on both polynomials c_0, c_1, followed by a well-known key switching operation:

$$\textsc{EvalAuto}(c, i, K_{\tau_i(s)\to s}) = \textsc{KeySwitch}_{\tau_i(s)\to s}((\tau_i(c_0), \tau_i(c_1)); K_{\tau_i(s)\to s}). \qquad (1)$$

Equation (1), however, requires knowledge of the corresponding Galois key, i.e., the key switching key from $\tau_i(s) \to s$. This key has to be generated by the party knowing the secret key s for each required automorphism.

[3] $n = \frac{\phi(m)}{d}$, where ϕ is Euler's totient function, m indicates the used m-th cyclotomic reduction polynomial, and $p^d \equiv 1 \bmod m$ with p being the prime plaintext modulus.

2.2 HE Schemes and Libraries

In this paper, we consider three HE schemes and their implementation in three libraries. We will discuss these schemes and libraries in this section.

BFV [Bra12, FV12] in SEAL [SEA20]. In BFV in SEAL plaintexts are elements in $\mathbb{Z}_q$. However, to support the packing described in the previous section, q has to be prime and congruent to 1 mod $2N$, where N is the degree of the cyclotomic reduction polynomial, which has to be a power of two. This condition implies that $\mathbb{Z}_2$ plaintexts can never have a packed implementation in SEAL. We use SEAL version 3.6.2 in the paper. The runtime and added noise by homomorphic additions is negligible, which is why additions are considered free in the BFV cryptosystem. Therefore, the most relevant performance metric is the multiplicative depth of the evaluated circuit.

BGV [BGV12] in HElib [HS20]. The BGV scheme, and its implementation in HElib, allows plaintexts in $\mathbb{Z}_{p^r}$ and offers more flexibility for choosing HE parameters than SEAL. It allows arbitrary cyclotomic reduction polynomials and it is possible to find parameters which allow packing for $\mathbb{Z}_2$ plaintexts. However, this flexibility comes with the drawback that parameterizing for HElib is more difficult than finding parameters in SEAL, and the limited parameter sets in SEAL allow for more optimized implementations. In this paper we use the HElib version 2.1.0. Similar to BFV in SEAL, additions are considered free in BGV, and the multiplicative depth of the circuit is the most relevant performance metric.

TFHE [CGGI20] in TFHE [CGGI16]. The TFHE library is vastly different from SEAL and HElib. It only allows the encryption of boolean values (i.e., plaintexts are in $\mathbb{Z}_2$), but it is optimized for fast gate bootstrapping. This basically means that after the evaluation of a homomorphic gate the noise in the ciphertext is reset. As a consequence, contrary to most other modern homomorphic encryption schemes, the multiplicative depth of a circuit is no relevant metric in TFHE. However, each homomorphically evaluated gate requires the same computational effort, thus additions are not considered to be free as in the BFV or BGV cryptosystems. The most relevant metric for TFHE is, therefore, the total number of gates. Furthermore, packing is not supported in TFHE. Since TFHE only allows to encrypt boolean values, we do not implement and consider $\mathbb{F}_p$ ciphers for TFHE.

Remark 1. Another HE scheme, the CKKS [CKKS17] cryptosystem (and its implementation in SEAL and HElib), is relevant for private statistics and machine learning. However, the scheme includes approximation errors, which makes it incompatible with homomorphically evaluating symmetric ciphers. Therefore, we do not consider it in this paper.

3 Hybrid Homomorphic Encryption Formally Defined

In the literature, HHE has not yet been formally defined for a generic use case, which is why we aim to do this here. We start by reviewing the previous work done in the light of defining HHE. In 2014, [BV14] generalized a construction for efficient private information retrieval (PIR) [CGKS95] used in an earlier version of the paper [BV11]. This can be seen as the first attempt to formalize what they dubbed HHE. Their definition is quite rigorous but has some minor technical issues, such as that the decryption circuit being built into the evaluation algorithm. Therefore it can only be called symmetric ciphertext. Shortly thereafter, [GGI+15] came up with their own HHE construction, which suffers from the same issue. In contrast to the last two approaches, the definition in the first work solely dedicated to HHE [CCF+16] does not have this issue, but it is restricted to additive stream ciphers. [MJSC16] is able to extend the definition to generic symmetric ciphers.

Although it is rather close to our own definition, it has two drawbacks. First, they include evaluating a function and sending the ciphertext back – which is done after a ciphertext reduction – into their definition of HHE. We strongly argue that these phases should not be within the definition of HHE. For instance, such a reduction only depends on the underlying HE scheme's properties but has no connection to the hybrid property, thereby unnecessarily complicating the definition. Secondly, their definition is a precise description but lacks the rigor to be considered a formal definition.

3.1 Definition

We now aim to come up with a generally valid definition of HHE. The definition will be highly influenced by the literature and based on the conclusions drawn from the different approaches. Before we can define HHE, we want to recall the definition of HE. There are numerous definitions of HE. We choose to follow the definition of [BV14], only adapting it for arbitrary messages. A homomorphic public key encryption scheme is a quadruple of probabilistic polynomial-time algorithms $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$:

- $(\mathsf{pk}, \mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KGen}(1^n)$,
- $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$,
- $m \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$, and
- $c_f \leftarrow \mathsf{Eval}(\mathsf{pk}, f, (c_1, \dots, c_n))$.

The correctness of a homomorphic encryption scheme is in a way incorporated into the ability to perform homomorphic computations. More concretely, let $\mathsf{HE.Enc}_{\mathsf{pk}}(m) = c$, then we say a homomorphic encryption scheme HE is correct if

$$\Pr[\mathsf{HE.Dec}_{\mathsf{sk}}(\mathsf{HE.Eval}_{\mathsf{evk}}(f, c)) \neq f(m)] \tag{2}$$

is negligible.

Definition 1 (Public-Key Hybrid Homomorphic Encryption (HHE))**.** Let HE be a public-key homomorphic encryption scheme and SYM a secret-key encryption scheme. Further, let $\mathcal{M}$ be the plaintext message space, $m \in \mathcal{M}$, and κ the security parameter. Construct a public-key homomorphic encryption scheme $\mathsf{HHE} = (\mathsf{HHE.KGen}, \mathsf{HHE.Enc}, \mathsf{HHE.Dec}, \mathsf{HHE.Eval})$ as follows:

```
$\mathsf{HHE.KGen}(1^\kappa)$
    $(\mathsf{sk}, \mathsf{evk}, \mathsf{pk}) \leftarrow \mathsf{HE.KGen}(1^\kappa)$
    return $(\mathsf{sk}, \mathsf{evk}, \mathsf{pk})$

$\mathsf{HHE.Enc}(1^\kappa, \mathsf{pk}, m)$
    $k \leftarrow \mathsf{SYM.KGen}(1^\kappa)$
    $\hat{k} \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}}(k)$
    $c \leftarrow \mathsf{SYM.Enc}_k(m)$
    return $(\hat{k}, c)$

$\mathsf{HHE.Dec}(\mathsf{sk}, c)$
    return $\mathsf{HE.Dec}(\mathsf{sk}, c)$

$\mathsf{HHE.Eval}(f, \mathsf{evk}, c_1, \dots, c_n)$
    return $\mathsf{HE.Eval}(f, \mathsf{evk}, c_1, \dots, c_n)$

$\mathsf{HHE.Decomp}(c, \hat{k}, \mathsf{evk})$
    $\hat{c} \leftarrow \mathsf{HE.Eval}_{\mathsf{evk}}(\mathsf{SYM.Dec}, \hat{k}, c)$
    return $\hat{c}$
```

Scheme 1: Hybrid Homomorphic Encryption

We call HHE a public-key hybrid homomorphic encryption scheme. HHE is correct if it is correct in the sense of homomorphic encryption. Note that the relation between ciphertext and message in Equation (2) changes to $\hat{c} = \mathsf{HHE.Decomp}_{\mathsf{evk}}(\mathsf{HHE.Enc}_{\mathsf{pk}}(m))$. In Appendix A, we show that every HHE scheme is indeed an HE scheme.

3.2 Security

The high-level idea of HHE is related to the KEM-DEM[4] [KL14] paradigm, as pointed out by [CCF+16]. However, they only discussed this aspect informally, and [Méa17] later pointed out that there has been no particular investigation in the security of HHE when seen in the light of the KEM-DEM paradigm. Therefore, we decided to perform an analysis of the connection between HHE and KEM-DEM and possible implications for HHE schemes' security.

To see if there is a direct formal relation between HHE and KEM-DEM, we instantiate the KEM-DEM paradigm with a public-key HE scheme. This is well-defined because every public-key encryption scheme is sufficient for a KEM. If we compare the resulting construction to Definition 1, we immediately see that the encryption algorithm is identical. In contrast, the decryption algorithm is neither identical to the decryption algorithm nor the decompression algorithm of the HHE scheme. Even more problematic, the central idea of HHE – the homomorphic decryption of symmetric ciphertexts with a public evaluation key – is not captured by the KEM-DEM paradigm at all. Therefore, we argue that HHE is not a mere instantiation of the KEM-DEM paradigm, although both have similar objectives.

Does this mean that we have to do the security analysis of HHE from scratch? First, recall that a public-key HE scheme is semantically secure if it is secure as a public-key encryption scheme [BV14]. Secondly, the proofs of the KEM-DEM construction's semantic security only depend on the encryption algorithms. Therefore, we can apply the KEM-DEM construction statements, albeit our decryption deviates from the KEM-DEM paradigm. We restate the semantic security theorem of the KEM-DEM paradigm for HHE. A proof can be found in [KL14].

Theorem 1. *Let* HE *be an IND-CPA-secure public-key homomorphic encryption scheme and* SYM *an IND-CPA-secure secret-key encryption scheme. Then* HHE *is an IND-CPA-secure public-key* HE *scheme.*

3.3 Comparison with LWE-Native Symmetric Encryption

In [CDKS20], the authors describe efficient algorithms to convert many LWE ciphertexts into a packed (see Section 2.1) R-LWE one. These algorithms can also be used to reduce ciphertext expansion of homomorphic encryption. Their approach works as follows: First, they encrypt each plaintext $m_i \in \mathbb{F}_p$ under a secret key $\vec{s} \in \mathbb{Z}^N$ using basic LWE encryption by sampling a random vector $\vec{a}_i \leftarrow U(\mathbb{Z}_q^N)$ and calculating $b_i = -\langle \vec{a}_i, \vec{s} \rangle + \mu_i$, where $\mu_i \in \mathbb{Z}_q$ is a randomized encoding of m_i (with Gaussian noise). The LWE ciphertext then is $(b_i, \vec{a}_i) \in \mathbb{Z}_q^{N+1}$. To further reduce the size of the ciphertexts, one can use a random seed se and generate $\vec{a}_i$ using a pseudo-random number generator (PRNG). The seed can be reused to generate the random part of each ciphertext as $\vec{a}_i = f(\mathsf{se}; i)$. The resulting ciphertexts are semantically secure in the random oracle model. The client then transmits all b_i alongside the seed se to the server, which then transforms all LWE ciphertexts into a packed HE one using the algorithms described in [CDKS20]. The total communication cost for this approach is one $\mathbb{Z}_q$ element for each plaintext $m_i \in \mathbb{F}_p$, plus one seed se to generate the random part of the ciphertexts.

[4]Key-Encapsulation Mechanism-Data-Encapsulation Mechanism

According to the benchmarks in [CDKS20][5] the LWE encryption approach seems to be computationally more efficient than HHE. However, their algorithms do not achieve a ciphertext expansion factor of 1, but a factor of $\frac{\log q}{\log p} + \mathcal{O}(1)$. For many HE applications, the size of q can easily exceed 800 bits, resulting in big expansion factors. Furthermore, as we show with our benchmarks in this paper, it is not enough to compare the performance of just transforming ciphertexts into HE ciphertexts, but one also has to consider the use case afterwards (with increased HE parameters) to get a good picture of the computational overhead of the ciphertext expansion prevention. Not only are the benchmarks in [CDKS20] not extensive, they are also missing the final transformation to get a packed HE ciphertext and give no performance numbers for HE parameters required for larger use cases.

3.4 Hybrid Homomorphic Encryption in Practice

Contrary to KEM-DEM, the performance, advantages, and disadvantages of HHE are not so well understood so far. Therefore, we start with an investigation of applying HHE to a real use case and report the results of using the SEAL library in Table 1. Matrix multiplications over integers are a basic building block in many applications involving statistics or machine learning. Our use case, therefore, is an application with three affine transformation $\vec{x}'_i = M_i \cdot \vec{x}_i + \vec{b}_i$, where $\vec{x}_i, \vec{x}'_i, \vec{b}_i \in \mathbb{F}_p^{200}$, $M_i \in \mathbb{F}_p^{200\times 200}$, and p is a 60-bit prime. To make the use case more generic, we square the output vector after the first two affine transformation, resulting in a multiplicative depth of 3 plaintext-ciphertext and 2 ciphertext-ciphertext multiplications. We benchmark this use case after the initial setup phase, i.e., the server knows an HE encryption of the symmetric key and all HE evaluation keys. Further, we repeat this 1000 times, and the server aggregates the final results before sending them back to the client. In a real-world scenario, this would be equivalent to, e.g., a sensor device sending statistics in fixed intervals to a server. In Table 1, we give results for HHE using 3 different ciphers (Pasta-3 as defined in Section 6, Agrasta, and AES) and compare it to just using homomorphic encryption and also to an estimation of using LWE-native encryption [CDKS20]. As Table 1 shows, using HHE reduces the total client-to-server communication from 7.4 GB to 1.5 MB, the exact size of sending the input vector consisting of 200 60-bit field elements 1000 times. Furthermore, data encryption is also faster and requires less RAM. However, to support the homomorphic evaluation of the HHE decompression circuit, the server-side requires larger HE parameters with higher noise budget, increasing the server-side runtime and RAM requirements. For HHE using Pasta-3, the server-side runtime increases by a factor of 10. However, using HHE with $\mathbb{Z}_2$ ciphers (Agrasta or AES) requires to implement binary circuits for the use case, resulting in a significant multiplicative depth requiring large HE parameters, and thus in infeasibly long server runtimes.

Remark 2. As discussed in Section 2, one can use bootstrapping to reset the noise in a homomorphic ciphertext to allow the evaluation of circuits with arbitrary multiplicative depth. However, SEAL does not support bootstrapping, which is why we do not consider it in this example. Furthermore, bootstrapping is still very inefficient in HElib and does not result in faster runtimes for the $\mathbb{Z}_2$ ciphers compared to Pasta in HHE. Therefore, we do not give explicit bootstrapping benchmarks for HElib in this paper. In TFHE, bootstrapping is already applied after each gate, which is why TFHE allows to evaluate circuits with arbitrary depth.

As discussed in Section 3.3, LWE-native encryption [CDKS20] has larger ciphertext expansion then HHE. We estimate that the overhead of the server-side computations does not require larger HE parameters as just using homomorphic encryption, which is why the expansion factor is $\frac{881}{60} = 14.68$ for the used parameter set.

[5]Their source code is not public at the time of writing.

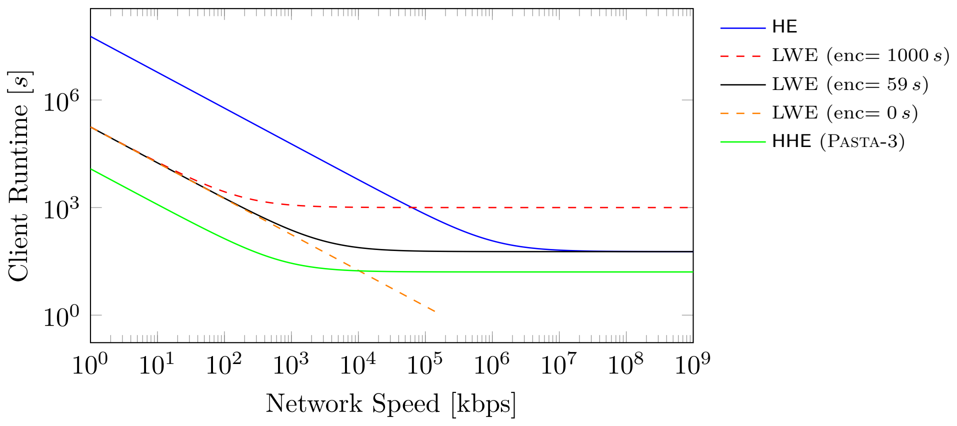

Figure 1: Encryption + upload time of HE, HHE with Pasta, and LWE-native encryption [CDKS20] (three different encryption timings, since no explicit benchmarks are known) depending on network speed.

Table 1 clearly shows that just using homomorphic encryption would result in unnecessarily large client-to-server communication. To further demonstrate the performance loss, we show the combined client timings (for encryption and client-to-server communication) for different network speeds in Figure 1. We depict timings for using only HE, HHE using Pasta-3, and our estimates for the LWE-native approach. We omit HHE using $\mathbb{Z}_2$ ciphers, since they result in infeasible server runtimes. Since we do not have accurate performance numbers for the LWE-native client-side encryption, we show three different encryption numbers: *(1)* a bad case, where the total encryption time is 1000 s, *(2)* timings for an equal encryption time to just using HE, and *(3)* the best case with a 0 s encryption time. Figure 1 shows that using HHE always results in the fastest client-side latencies, especially for network speeds below 10 Mbps (the average LTE upload speed in the USA is 5 Mbps[6]) where runtime is fully dominated by the data transmission. The advantages compared to just using HE, however, also range up to speeds of 1 Gbps.

Both using homomorphic encryption and using LWE-native encryption require sampling Gaussian noise during encryption. Constrained devices, however, often do not have access to a reliable source of randomness. Therefore, we also list the number of random Gaussian words required on the client side to perform the encryption in Table 1. HHE does not require sampling random values during encryption, which is why using HHE is the preferable choice on constrained devices without a reliable source of randomness.

To summarize, if the encryption time on a client is the bottleneck, then using HHE with Pasta-3 is the preferred choice. Only HHE using AES is faster, but using AES results in infeasibly long server-side computations. On the other hand, if the client bandwidth is the bottleneck, then HHE (using Pasta-3) has a considerable advantage. The concrete communication advantage depends on the HE parameters. In our concrete example ($N = 32768$ for both, HE and using LWE-native conversions), HE requires a factor of 4936× more communication than HHE, the LWE-native approach a factor of 14.86×. Since HHE has the largest server-side runtime overhead, using HHE has the best effect on constrained clients or in slow network settings. The choice of the symmetric cipher used in HHE has similar effects on the client side (all have ciphertext expansion of 1), but severly effects the server-side runtime, which is why we investigate server-side computation in the remainder of the paper.

[6] https://www.verizon.com/articles/4g-lte-speeds-vs-your-home-network/

Table 1: Comparison of a use case with HHE to only using HE in SEAL.

	Client				Server		
	Random Words	Enc. *s*	RAM GB	Comm. kB	Runtime *s*	RAM GB	Comm. kB
HE	65 536 000	59	0.550	7 404 700	61 900	2.24	987.3
HHE **(Pasta-3)**	**0**	**16**	**0.005**	**1 500**	**669 400**	**23.0**	**2 097.3**
HHE (Agrasta)	0	2 200	0.004	1 500	?[a]	?[a]	> 12 000 000[b]
HHE (AES)	0	0.040	0.003	1 500	?[a]	?[a]	> 12 000 000[b]
LWE [CDKS20]	200 000	?	0.550[c]	22 025	120 000[c]	2.24[c]	987.3[c]

[a] Multiplicative depth of binary circuit (> 400) far too large for feasible HE parameters.
[b] No packing in SEAL for $\mathbb{Z}_2$ plaintexts, i.e., one HE ciphertext per bit.
[c] Estimates based on HE benchmarks and [CDKS20].

4 Usability of Existing Symmetric Ciphers

In this section, we evaluate the usability of proposed symmetric ciphers for the HHE use case. Most existing designs are ciphers over $\mathbb{Z}_2$. In the following section we benchmark instances of these ciphers which provide 128 bits of security with a simple HHE use case, which already highlights the requirement of ciphers over $\mathbb{F}_p$. Furthermore, we discuss existing $\mathbb{F}_p$ designs in this section.

4.1 Ciphers over $\mathbb{Z}_2$

As outlined above, the majority of ciphers designed for HHE are defined over $\mathbb{Z}_2$. The main design criteria of all these ciphers is to reduce the AND depth of the decryption circuit. In the following we will shortly discuss each of the ciphers, in Table 2 we summarize the parameters of the ciphers in their respective modes of operation.

LowMC [ARS+15]. LowMC is a very parameterizable blockcipher, designed to minimize the number of AND gates required for secure encryption. In this work, we benchmark LowMC ($n = 256$, $k = 128$, $m = 63$, $r = 14$, $d = 128$), the LowMC instance proposed for HHE in the original publication. To encrypt plaintext of arbitrary length, we use LowMC in the Counter (CTR) mode of operation.

Rasta [DEG+18]. Rasta is a family of stream ciphers, in which a permutation is applied to the secret key to produce the keystream. The permutation consists of several rounds of affine layers and an S-box instantiated with the χ-transformation [Dae95]. The affine layers are randomly generated for each new block from an extendable-output function (XOF) [NIS15] seeded with a nonce N and the block counter i. We depict the Rasta permutation in Figure 2. In this work, we benchmark Rasta ($n = 525$, $r = 5$) and Rasta ($n = 351$, $r = 6$). Since Rasta is a stream cipher similar to the CTR mode of operation, no further modifications are required to support encrypting plaintexts of arbitrary length.

Agrasta [DEG+18]. Agrasta is an aggressive instantiation of the Rasta stream cipher with significantly smaller security margin. The smaller security margin allows to further reduce the permutation width compared to more conservative parameter sets. In this work we benchmark Agrasta ($n = 129$, $r = 4$).

Dasta [HL20]. Dasta is another family of stream ciphers, similar to the Rasta family. It keeps Rasta's basic design, but substitutes the random affine layer with a permutation layer followed by a constant matrix multiplication. With that, Dasta achieves faster encryption in plain (i.e., without HE), but keeping roughly the same performance for the

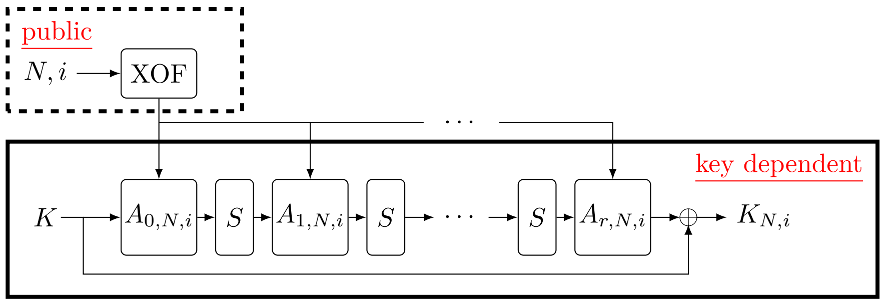

Figure 2: The r-round Rasta construction to generate the keystream $K_{N,i}$ for block i under nonce N with affine layers $A_{j,N,i}$. The picture is taken from [DEG+18].

homomorphic evaluation. In this paper, we benchmark Dasta ($n = 525$, $r = 5$) and Dasta ($n = 351$, $r = 6$).

Kreyvium [CCF+16]. Kreyvium is a 128 bit security variant of the low-depth stream cipher Trivium [Can06]. In Kreyvium, the AND-depth of the cipher grows with the number of required keystream bits, which is undesired in most homomorphic encryption libraries. To keep the AND-depth constant, even for larger plaintexts, we use Kreyvium as a block cipher in the Counter (CTR) mode of operations, with the block size being the maximum amount of keystream bits for a given AND-depth (including plaintext-ciphertext AND gates). In this work, we benchmark Kreyvium as stream cipher alongside the two depth-bounded modes Kreyvium-12 and Kreyvium-13.

FiLIP [MCJS19]. FiLIP is another family of stream ciphers designed to minimize the number of AND gates required for secure encryption. In contrary to Kreyvium, FiLIP's AND depth does not increase the more bits are encrypted, so no further modifications are required for efficient homomorphic decryption. In this work, we benchmark FiLIP-1216 and FiLIP-1280.

Table 2: Parameters of the benchmarked $\mathbb{Z}_2$ ciphers in their respective modes of operations in bits.

Cipher	Blocksize	Keysize	Rounds	AND-depth
LowMC	256	128	14	14
Rasta-5	525	525	5	5
Rasta-6	351	351	6	6
Dasta-5	525	525	5	5
Dasta-6	351	351	6	6
Agrasta	129	129	4	4
Kreyvium	-	128	-	-
Kreyvium-12	46	128	-	12
Kreyvium-13	125	128	-	13
FiLIP-1216	-	16384	-	3
FiLIP-1280	-	4096	-	4

Remark 3. Kreyvium, FiLIP-1216, and FiLIP-1280 are stream ciphers without defined block size. In our benchmarks, we therefore define one block to be 46 bits for Kreyvium and 64 bits for both FiLIP instances.

4.1.1 Benchmarked Application

Hybrid homomorphic encryption aims to reduce the communication overhead for outsourcing computations to a cloud. Therefore, in our benchmarks, we do not only measure and compare the performance of the decryption circuit of each cipher under homomorphic encryption, but also the performance of the cipher in a complete HHE use case. The use case we benchmark is a server which computes $\vec{r} = M \cdot \vec{v} + \vec{b}$, where $\vec{r}, \vec{v}, \vec{b} \in \mathbb{Z}_{2^{16}}^5$ and $M \in \mathbb{Z}_{2^{16}}^{5\times 5}$, i.e., a 5×5 matrix-vector multiplication of 16-bit integers. The matrix M and the vector $\vec{b}$ are private and owned by the server, whereas $\vec{v}$ is a private vector owned by the client. The client uses HHE to send $\vec{v}$ in encrypted form to the server, and will get $\vec{r}$ in encrypted form as a result.

As described above, the choice of a cipher over $\mathbb{Z}_2$ also requires that we compute the integer matrix multiplication over $\mathbb{Z}_2$. This requires the implementation of binary circuits for addition[7] and multiplication, which have a much higher AND depth than performing the same operations over $\mathbb{F}_p$. Despite being only a very small matrix multiplication (5×5 with 16-bit integers), our benchmarks show that the evaluation is already very slow, making it infeasible for $\mathbb{Z}_2$ ciphers to be applied to real-world statistics or machine learning use cases with multiple chained matrix multiplications of larger integers with matrices consisting of hundreds of entries.

4.1.2 Benchmark Platform

We run all benchmarks on a Linux server with an Intel Xeon E5-2699 v4 CPU (2.2 GHz, turboboost up to 3.6 GHz) and 512 GB RAM. Each of the individual benchmarks has only access to one thread.

4.1.3 SEAL Benchmarks

In SEAL, the available noise budget (i.e., how much further noise can be introduced until decryption will fail) depends on the ciphertext modulus q. However, big moduli q require a big degree N of the cyclotomic reduction polynomial for security. N, which is always a power of two, has a severe impact on the performance of the HE scheme. While a larger N allows for larger q to increase the noise budget, it exponentially increases the runtime of homomorphic operations.

In Table 3 we present the benchmarks for the SEAL library, for homomorphically decrypting only one block, and for the small HHE use case. For both benchmark we give timings for homomorphically encrypting the symmetric key and homomorphically decrypting the symmetric ciphertexts (i.e., decompressing the HHE ciphertext) for the smallest N allowing enough noise budget for correct evaluation. We parameterize q such that the HE scheme has a security of 128 bits. For the HHE use case we additionally give the runtime for the matrix multiplication. In Table 4 we additionally give the remaining noise budget after encrypting the symmetric key, homomorphically decrypting the symmetric ciphertexts, and performing the matrix multiplication. Since SEAL does not allow to use packing with plaintexts in $\mathbb{Z}_2$, all implementations are bitsliced (i.e., one HE ciphertext per bit).

4.1.4 HElib Benchmarks

In HElib, the security and available noise budget mainly depend on the choice of the cyclotomic reduction polynomial, as well as the size of the ciphertext modulus. A bigger modulus provides a bigger noise budget at the cost of less security. A bigger cyclotomic

[7]We implemented depth-optimized carry-lookahead adders (CLA) in HElib and SEAL, and standard ripple carry adders (RCA) in TFHE.

Table 3: Benchmarks of the $\mathbb{Z}_2$ ciphers in the SEAL library.

Cipher	1 Block			Small HHE use case			
	N	Enc. Key	Decomp.	N	Enc. Key	Decomp.	Use Case
		s	s		s	s	s
LowMC	16384	1.75	613.9	32768	6.12	2 702.7	1 202.1
Rasta-5	8192	2.12	135.9	32768	25.4	2 618.5	1 201.8
Rasta-6	8192	1.42	88.5	32768	17.1	1 802.0	1 199.6
Dasta-5	8192	2.20	134.1	32768	25.4	2 594.0	1 209.2
Dasta-6	8192	1.49	88.7	32768	17.2	1 811.8	1 209.8
Agrasta	8192	**0.534**	**16.3**	16384	**1.76**	**76.2**	**241.0**
Kreyvium	16384	1.84	412.8	32768	6.17	2 028.5	1 210.7
Kreyvium-12	16384	1.75	414.8	32768	6.30	3 925.8	1 217.9
Kreyvium-13	16384	1.83	442.1	32768	6.18	1 999.0	1 199.3
FiLIP-1216	8192	66.1	1 064.7	16384	223.9	6 619.0	244.5
FiLIP-1280	8192	16.7	1 251.6	16384	56.0	7 783.2	242.0

Table 4: Noise budget after each operation for $\mathbb{Z}_2$ ciphers in the SEAL library.

Cipher	1 Block			Small HHE use case			
	N	Enc. Key	Decomp.	N	Enc. Key	Decomp.	Use Case
		bit	bit		bit	bit	bit
LowMC	16384	379	68	32768	815	491	190
Rasta-5	8192	165	46	32768	814	686	384
Rasta-6	8192	165	31	32768	815	670	361
Dasta-5	8192	165	46	32768	815	686	380
Dasta-6	8192	165	32	32768	815	670	365
Agrasta	8192	165	82	16384	379	292	10
Kreyvium	16384	379	200	32768	815	611	312
Kreyvium-12	16384	379	201	32768	815	624	323
Kreyvium-13	16384	379	188	32768	815	612	314
FiLIP-1216	8192	165	122	16384	379	334	51
FiLIP-1280	8192	164	109	16384	379	319	37

polynomial provides more security, but is bad for performance. In our benchmarks we try to find parameters which provide roughly 128 bits of security.

In Table 5 we present the benchmarks for the HElib library, for homomorphically decrypting only one block, and for the small HHE use case. For both benchmarks we give timings alongside the choice of the m-th cyclotomic reduction polynomial and the estimated security λ'. For the HHE use case we additionally give the runtime for the matrix multiplication. In Table 6 we additionally give the remaining noise budget after encrypting the symmetric key, homomorphically decrypting the symmetric ciphertexts, and performing the matrix multiplication. To compare the benchmarks to SEAL and TFHE, all implementations are bitsliced (i.e., one HE ciphertext per bit).

Remark 4. HElib supports packing for $\mathbb{Z}_2$ plaintexts. Even though a packed implementation of the symmetric ciphers will increase their overall performance, it complicates the evaluation of an integer matrix-vector multiplication based on binary circuits. Therefore, packed implementations do not fix the main issue of $\mathbb{Z}_2$ ciphers for HHE, which is supporting integer arithmetic over $\mathbb{F}_p$. For this reason, we do not provide explicit packed benchmarks for the ciphers in the paper.

Table 5: Benchmarks of the $\mathbb{Z}_2$ ciphers in the HElib library.

Cipher	1 Block				Small HHE use case				
	m	λ' bit	Enc. Key s	Decomp. s	m	λ' bit	Enc. Key s	Decomp. s	Use Case s
LowMC	23377	110	9.22	1 132.4	43691	108	27.5	3 708.8	1 618.8
Rasta-5	11441	111	11.7	284.2	31609	118	57.7	1 666.9	922.4
Rasta-6	11441	111	7.79	207.7	31609	108	41.8	1 401.0	1 037.2
Dasta-5	11441	111	11.8	276.7	31609	118	57.9	1 608.4	922.0
Dasta-6	11441	111	7.87	201.7	31609	108	41.6	1 357.3	1 042.6
Agrasta	10261	117	**2.38**	**38.3**	32767	108	**13.7**	**276.9**	**853.6**
Kreyvium	14351	108	3.97	497.0	43691	144	22.0	3 392.6	1 431.9
Kreyvium-12	14351	108	4.06	498.3	43691	147	22.0	6 657.1	1 392.6
Kreyvium-13	15709	113	4.38	577.1	43691	144	21.7	3 407.3	1 420.9
FiLIP-1216	5461	113	131.4	1 357.5	23311	108	1 010.0	17 919.7	566.6
FiLIP-1280	8435	119	47.3	2 197.4	24929	105	337.2	27 613.9	745.2

Table 6: Noise budget after each operation for $\mathbb{Z}_2$ ciphers in the HElib library.

Cipher	1 Block				Small HHE use case				
	m	λ' bit	Enc. Key bit	Decomp. bit	m	λ' bit	Enc. Key bit	Decomp. bit	Use Case bit
LowMC	23377	110	427	13	43691	108	823	396	21
Rasta-5	11441	111	188	27	31609	118	549	378	23
Rasta-6	11441	111	188	5	31609	108	574	381	20
Dasta-5	11441	111	188	27	31609	118	549	378	23
Dasta-6	11441	111	188	5	31609	108	574	381	20
Agrasta	10261	117	144	17	32767	108	492	372	24
Kreyvium	14351	108	242	18	43691	144	652	394	21
Kreyvium-12	14351	108	242	18	43691	147	640	397	20
Kreyvium-13	15709	113	247	2	43691	144	652	393	21
FiLIP-1216	5461	113	83	19	23311	108	442	380	25
FiLIP-1280	8435	119	86	2	24929	105	456	375	20

4.1.5 TFHE Benchmarks

Since in TFHE the noise in the ciphertexts is reset after every homomorphic operation, we do not have to choose any parameters for the benchmarks (except the security level, which we set to 128 bits). In Table 7 we present the benchmarks for the TFHE library for homomorphically decrypting only one block, and for the small HHE use case. We give timings for homomorphically encrypting the symmetric key, homomorphically decrypting one block, and for the small HHE use case. Since TFHE does not support packing all implementations are bitsliced (i.e., one HE ciphertext per bit).

4.1.6 Discussion

Our benchmarks show that the runtime of the whole HHE use case (including cipher evaluation) using the $\mathbb{Z}_2$ ciphers is high, despite the tested use case being small. This emphasizes the requirement of $\mathbb{F}_p$ ciphers for HHE with integer use cases. In SEAL and HElib, the fastest ciphers are the ciphers based on the Rasta design strategy (Rasta, Dasta, Agrasta), with Agrasta being the fastest due to its small multiplicative depth. Only FiLIP has better noise propagation. However, due to its large symmetric key and long evaluation time, it is not competitive. For figures comparing the runtime of HHE in SEAL and HElib and a comparison to $\mathbb{F}_p$ ciphers, we refer to Section 8.1.

In Figure 3 we compare the runtime of homomorphically decrypting one block and the whole HHE use case (including homomorphic decryption) of the $\mathbb{Z}_2$ ciphers in TFHE. In TFHE the main performance metric is the total gate count, which is why Kreyvium is the fastest choice. Since the TFHE library only allows plaintexts in $\mathbb{Z}_2$ we do not implement

Table 7: Benchmarks for the TFHE library.

Cipher	Enc. Key	1 Block Decomp.	Small HHE use case Decomp.	Use Case
	s	s	s	s
LowMC	0.003	6 120.5	6 310.6	175.6
Rasta-5	0.013	5 728.8	5 807.8	164.0
Rasta-6	0.009	3 275.0	3 293.6	162.2
Dasta-5	0.013	5 642.6	5 664.7	165.2
Dasta-6	0.009	3 272.7	3 293.0	162.0
Agrasta	0.003	407.1	408.4	164.4
Kreyvium	**0.003**	**284.1**	**290.4**	**162.6**
Kreyvium-12	0.003	284.1	559.7	162.6
Kreyvium-13	0.003	310.1	290.4	162.6
FiLIP-1216	0.442	1 504.6	1 886.9	164.3
FiLIP-1280	0.107	1 594.5	1 981.8	162.8

and compare $\mathbb{F}_p$ ciphers in TFHE.

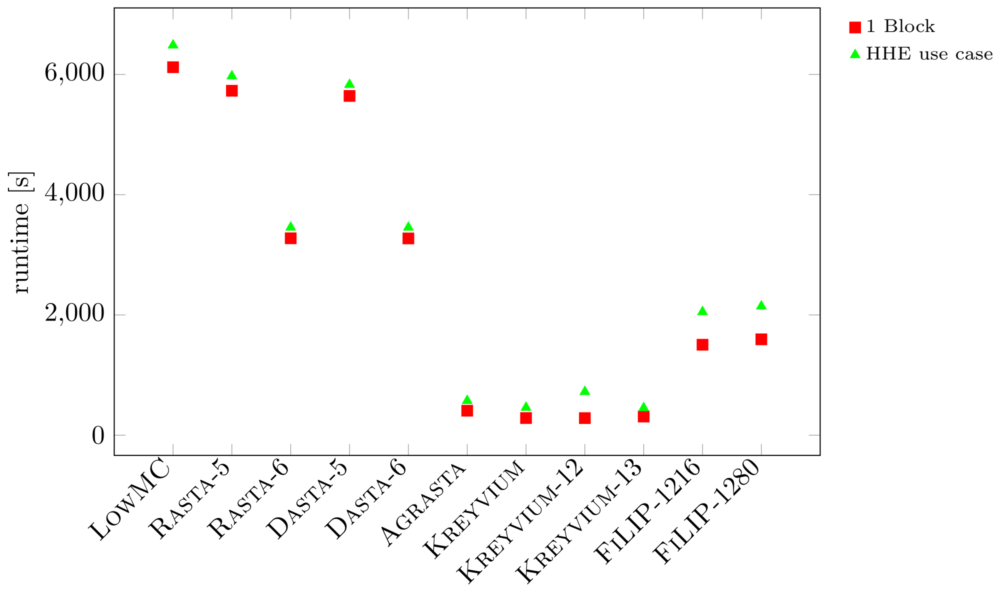

Figure 3: Runtime comparison of homomorphically decrypting one block, and the small HHE use case (including HHE decompression) of $\mathbb{Z}_2$ ciphers in TFHE.

4.2 Masta– A Cipher over $\mathbb{F}_p^t$

In independent and concurrent work the only other symmetric cipher over $\mathbb{F}_p^t$ proposed for HHE use cases we are aware of is Masta [HKC+20]. Similar to our work, the need for efficient ciphers over $\mathbb{F}_p^t$ for HHE is identified there. However, [HKC+20] is missing an extensive treatment of the HHE topic. On one hand, they do not provide any kind of design space exploration to find a well-optimized cipher for HHE, on the other hand they are also missing extensive benchmarks comparing their design to the state-of-the-art and do not provide benchmarks for any HHE use cases.[8]

Masta can be seen as a direct translation of Rasta (Figure 2) to $\mathbb{F}_p^t$, with the exception of a different strategy in sampling random invertible matrices. Their approach involves sampling a random polynomial $m \in \mathbb{Z}_p[X]/(X^t - \alpha)$ and translating m into a matrix M. This matrix is then invertible by design and they only have to sample s field elements $\in \mathbb{F}_p$. Even though the S-box used in Rasta is in general no permutation over $\mathbb{F}_p^t$, and therefore limits the possible outputs of the S-box layer in Masta, the designers did not consider any additional changes to the baseline design and do not leverage any advantages of HE over fields $\mathbb{F}_p$.

In this paper we consider the two 128-bit security instances of Masta with the lowest depth and use Shake128 to pseudorandomly generate all affine layers. We summarize these instances in Table 8. In Section 8 we benchmark and compare Masta to our Pasta design.

Table 8: Considered Masta instances (128-bit security).

Instance	Rounds	# Key Words	# Plain Words	# Cipher Words	XOF
Masta-4	4	128	128	128	Shake128
Masta-5	5	64	64	64	Shake128

5 Designing an Efficient Cipher for HHE over $\mathbb{F}_p^t$

The goal is to design an efficient cipher for HHE over $\mathbb{F}_p^t$ with $2^{16} < p < 2^{60}$.[9] Since in both BGV and BFV (and their respective implementations in SEAL and HElib) the most significant performance metric is the multiplicative depth, our main goal is to reduce this metric. Since every round contributes to the multiplicative depth, and therefore to the overall noise consumption during a homomorphic evaluation of the cipher, we aim to design a secure cipher with a minimal number of rounds. Further, high-degree polynomials have a large multiplicative depth, and hence we consider low-degree S-boxes. Meeting both of these requirements usually results in a large state size for security. However, large state sizes lead to a high runtime of the cipher evaluation. Therefore, our design will have to balance noise consumption and runtime to be efficiently usable in HHE.

Furthermore, most HE applications leverage packing (Section 2.1) to increase performance, which is why we also aim to design a packing-friendly cipher which produces packed homomorphically encrypted ciphertexts.

Cost of HE Operations. In Table 9 we summarize the cost of each HE operation in SEAL and HElib. Note that the key switching operation is free in terms of noise in SEAL, whereas it adds noise to the ciphertext in HElib. Key switching is required after a ciphertext-ciphertext multiplication and after an homomorphic Galois automorphism (required for rotation), which is why these operations require more noise in HElib. For both libraries the noise consumption depends on the size of the prime p, with larger p implying higher noise consumption, especially in (pt-ct and ct-ct) multiplications. Therefore, one cannot consider plaintext-ciphertext multiplications as negligible when working over $\mathbb{F}_p$ and we also have to consider the plaintext-ciphertext multiplicative depth when designing an efficient cipher over $\mathbb{F}_p$.

[8]We suspect that in [HKC+20] the authors only benchmark a word-sliced HE implementation of Masta, which is why our packed implementation of Masta is significantly faster.

[9]SEAL does not allow larger field sizes.

Table 9: Cost of HE operations in SEAL and HElib.

	SEAL		HElib	
Operation	Noise	Runtime	Noise	Runtime
pt-ct Add	negligible	cheap	negligible	cheap
ct-ct Add	negligible	cheap	negligible	cheap
pt-ct Mul	moderate	cheap	moderate	cheap
ct-ct Mul	expensive	expensive	expensive	expensive
Automorphism	negligible	expensive	moderate	expensive

5.1 Design Basis

Since our $\mathbb{Z}_2$ benchmarks indicate that designs based on Rasta are the preferred choice, we first consider an $\mathbb{F}_p^t$ version of Rasta with equal text/key size, and then modify it for security and efficiency. In the following, we analyze several candidates for each of the operations defining the cipher, and we also determine their implementation efficiency. Based on these results, we then design Pasta in Section 6.

5.2 S-Box

The original Rasta design uses the χ-transformation [Dae95] over $\mathbb{Z}_2^t$ as a single nonlinear layer. However, the χ-function is in general no permutation when working over $\mathbb{F}_p^t$, which is why we consider some alternatives. In this section we compare several S-boxes and describe how they can be efficiently implemented in a packed homomorphic evaluation. Despite not being a permutation, Masta still uses the χ-function naturally defined over $\mathbb{F}_p^t$, which is why we include it in our comparison.

χ-S-box. The χ-S-box is defined as

$$\chi(\vec{x})_i = x_i + x_{i+2} + x_{i+1} \cdot x_{i+2} = x_i + x_{i+2} \cdot (1 + x_{i+1}).$$

The indices in the χ-S-box are taken modulo t, which is why χ can be efficiently evaluated using rotations, i.e.,

$$\chi(\vec{x}) = \vec{x} + \mathtt{rot}_2(\vec{x}) \circ (\vec{1} + \mathtt{rot}_1(\vec{x})).$$

This works if the rotation is cyclic for the vector of size t. However, once encrypted, homomorphic rotations are cyclic over a larger vector of size n. Hence, we need to simulate cyclic rotation by preprocessing the state first. However, the resulting vector has more than t elements, which can influence further homomorphic operations. Thus, one has to apply a masking multiplication afterwards with a mask $\vec{m} = \vec{1} \in \mathbb{F}_p^t$:

$$\begin{aligned} \vec{x}' &= \vec{x} + \mathtt{rot}_{(-t)}(\vec{x}) \\ \Rightarrow \chi(\vec{x}) &= \left(\vec{x}' + \mathtt{rot}_2(\vec{x}') \circ (\vec{1} + \mathtt{rot}_1(\vec{x}'))\right) \circ \vec{m}. \end{aligned}$$

Cube S-box. Given a prime p, $\gcd(p-1, 3) = 1$, let

$$[S(\vec{x})]_i = (x_i)^3.$$

We recall that the cube S-box is the invertible power map with the smallest degree, and it can be efficiently evaluated by simply applying two homomorphic multiplications which affect the state elementwise, i.e., $S(\vec{x}) = \vec{x} \circ \vec{x} \circ \vec{x}$.

Feistel-Like S-Box (via a Quadratic Function).

$$S'(x_0 \| x_1 \| \cdots \| x_{s-1}) = x_0 \| (x_0)^2 + x_1 \| (x_1)^2 + x_2 \| \cdots \| (x_{s-2})^2 + x_{t-1}.$$

The Feistel-like S-box can also efficiently be implemented using rotations, i.e.,

$$S'(\vec{x}) = \vec{x} + \left(\texttt{rot}_{(-1)}(\vec{x}) \circ \vec{m}\right)^2,$$

where $\vec{m} \in \mathbb{F}_p^t$ is a masking vector $\vec{m} = [0, 1, \ldots, 1]^T$.

Alternative Feistel-Like S-Box (via the χ-Function).

$$S''(x_0 \| x_1 \| \cdots \| x_{t-1}) = x_0 \| x_1 \| x_0 \cdot x_1 + x_2 \| x_1 \cdot x_2 + x_3 \| \cdots \| x_{t-3} \cdot x_{t-2} + x_{t-1}.$$

The alternative Feistel-like S-box can also efficiently be implemented using rotations, i.e.,

$$S''(\vec{x}) = \texttt{rot}_{(-1)}(\vec{x}) \circ \texttt{rot}_{(-2)}(\vec{x}) \circ \vec{m} + \vec{x},$$

where $\vec{m} \in \mathbb{F}_p^s$ is a masking vector $\vec{m} = [0, 0, 1, \ldots, 1]^T$.

5.2.1 S-Box Cost Comparison

All S-box designs can efficiently be implemented on packed HE ciphertexts and require only a constant number of homomorphic operations independent of the state size. A summary of required homomorphic operations as well as the multiplicative depths of the different S-boxes is given in Table 10.

Table 10: Homomorphic operations and multiplicative depth of different S-boxes.

S-box	pt-ct Add	ct-ct Add	pt-ct Mul	ct-ct Mul	Rot	pt-ct Depth	ct-ct Depth
χ	1	2	1	1	3	1	1
S	-	-	-	2	-	-	2
S'	-	1	1	1	1	1	1
S''	-	1	1	1	2	1	1

Based on Table 10, we decide to choose the Feistel S-box S' as the main S-box for our nonlinear layers, and to use the cube S-box S to increase the degree of our cipher to combat linearization attacks and reduce the state size of the cipher.

5.3 Linear Layer

In Rasta, the homomorphic runtime is dominated by the linear layer. In this section we discuss how to efficiently implement matrix-vector multiplications on packed homomorphic ciphertexts and introduce optimizations to reduce the homomorphic evaluation time.

5.3.1 Choice of Random Matrices

In the original Rasta design, each random $t \times t$ matrix is directly sampled and checked for invertibility. However, doing the invertibility check is expensive in $\mathbb{F}_p$ in terms of computational complexity. Therefore, in Pasta we choose a different approach and generate each matrix as a sequential matrix [GPP11, GPPR11] (Section 6). These matrices are invertible by design and only require to sample t field elements and performing $t \cdot (t-1)$ field multiplications and $(t-1) \cdot (t-1)$ field additions. Compared to sampling polynomials $m_i \in \mathbb{Z}_p[X]/(X^t - \alpha)$ and translating them to matrices M_i (like in Masta), sequential matrices require to sample equally many field elements, but need more field additions and multiplications. Sampling sequential matrices is thus slower than the method used in Masta, but it comes with the cryptographic advantage of having less structure.

5.3.2 Babystep-Giantstep Matrix-Vector Multiplication

The most efficient way of evaluating the product between a plain matrix and an encrypted packed vector in HE is using the babystep-giantstep optimized diagonal method [HS14, HS15, HS18]. A matrix-vector multiplication of a matrix $M \in \mathbb{Z}^{t \times t}$ and a vector $\vec{x} \in \mathbb{Z}^t$ can be expressed by t elementwise vector-vector multiplications, $t-1$ rotations, and $t-1$ additions, operations that can easily be evaluated on packed ciphertexts:

$$M \cdot \vec{x} = \sum_{i=0}^{m-1} \texttt{diag}_i(M) \circ \texttt{rot}_i(\vec{x}) \tag{3}$$

$\texttt{diag}_i(M)$ in Eq. (3) expresses the i-th diagonal of matrix M in a vector of size t, with $i = 0$ being the main diagonal.

Rotations have a large evaluation time in modern HE schemes, which is why we reduce them using the babystep-giantstep optimization:

$$M\vec{x} = \sum_{k=0}^{t_2-1} \texttt{rot}_{(kt_1)} \left(\sum_{j=0}^{t_1-1} \texttt{diag}'_{(kt_1+j)}(M) \circ \texttt{rot}_j(\vec{x}) \right), \tag{4}$$

where $t = t_1 \cdot t_2$, $\texttt{diag}'_i(M) = \texttt{rot}_{(-\lfloor i/t_1 \rfloor \cdot t_1)}(\texttt{diag}_i(M))$.[10] Note that $\texttt{rot}_j(\vec{x})$ only has to be computed once for each $j < t_1$. Therefore, a matrix multiplication requires $t_1 + t_2 - 2$ rotations, t plaintext-ciphertext multiplications, and $t-1$ additions, and the total depth is 1 plaintext-ciphertext multiplication. For efficiency, we add words to the final state size of our design if t does not nicely split into $t = t_1 \cdot t_2$. Compared to the number of homomorphic operations required to evaluate the S-boxes (Table 10), it is clear that the runtime of the homomorphic evaluation of our cipher is dominated by the linear layer.

Remark 5. Similar to the implementation of the χ-S-box layer, we need to simulate a cyclic rotation over the vector of size t. This can be achieved by preprocessing the state with $\vec{x}' = \vec{x} + \texttt{rot}_{(-t)}(\vec{x})$, which requires an additional rotation and ciphertext-ciphertext addition. However, contrary to the χ-S-box, we can encode the masking multiplication together with the diagonals of Equation (4), which is why no additional plaintext-ciphertext multiplication is required.

5.3.3 Splitting the State

The babystep-giantstep algorithm dominates the runtime of the homomorphic Pasta evaluation and scales with the state size. Therefore, we propose to evaluate two individual instances of our cipher with state size t in parallel, with an efficient mixing step after each affine layer, allowing for an overall smaller state size. The final output of the design is then the output of the first half, and the second half is discarded. The result is a cipher with the following properties:

- The state size $s = 2 \cdot t$ is an even number and we truncate t words at the end.
- Instead of evaluating one large $s \times s$ matrix multiplication we perform two smaller $t \times t$ matrix multiplications.
- The S-box is applied on both branches individually.
- The key has now double the size of the keystream.

[10] In Eq. (4), $\lfloor i/t_1 \rfloor$ is equal to k.

On first inspection, this split does not impact the runtime, we still require $s = 2 \cdot t$ ciphertext-ciphertext additions and plaintext-ciphertext multiplications, and the number of required rotations either stays the same, or can potentially even become worse, depending on how good t can be split into $t_1 \cdot t_2$ compared to $s = s_1 \cdot s_2$. Furthermore, we now have double the key size compared to the keystream. However, the latter has no effect on the HHE use case, since in our packed homomorphic design we still require only one homomorphic ciphertext, which has the same size independent of the number of encoded words (as long as they are $\leq n$, where n is the number of available slots). Additionally, we can use the inner structure of homomorphic ciphertexts to parallelize both cipher evaluations, cutting the runtime down to an evaluation of one cipher instance of state size t.

Inner Structure of HE ciphertexts. In R-LWE based homomorphic encryption schemes (like BFV and BGV) the plaintexts are polynomials $\in R_p = \mathbb{F}_p[X]/\Phi_m(X)$, with $\Phi_m(X)$ being the m-th cyclotomic polynomial. Homomorphic additions and multiplications then correspond to polynomial addition and multiplication in plain. To get homomorphic integer operations one needs to encode integers efficiently into such polynomials. Using packing (Section 2.1) one can encode a vector of integers into one polynomial, and homomorphic additions and multiplications then affect these vectors element-wise. Further, one can use Galois automorphisms to permute the encoded vector. Thus, the encoded vector can be seen as a hypercube [HS14] and an automorphism rotates the data along one dimension. The precise structure of this hypercube depends on the choice of $\Phi_m(X)$. In general, it is possible to use these automorphisms to create linear rotations over the encrypted vector, but this requires masking multiplications [HS14], which when evaluated homomorphically require noise budget.

In terms of implementation efficiency, $\Phi_{2n}(X) = X^n + 1$, for n being a power of two, is a good choice. This polynomial is negacyclic and allows efficient polynomial multiplications via a negacyclic number theoretic transformation (NTT). Furthermore, decoding/encoding integer vectors then correspond to evaluating a NTT (on a permutation of the input vector) and its inverse respectively. For this reason, the homomorphic encryption standardization project[11] recommends using these power-of-two cyclotomic rings. Consequently, SEAL only implements HE with those rings and Masta is defined to use these rings as well [HKC+20]. The hypercube generated by such rings also has a nice structure: It corresponds to a matrix of two rows, each of size $\frac{\#\text{slots}}{2}$. Galois automorphisms can then directly be used to either linearly rotate both rows at once or rotate all columns simultaneously, i.e.,

$$\begin{bmatrix}\vec{x}_L \\ \vec{x}_R\end{bmatrix} \overset{\text{encode}}{\rightarrow} x \in R_p : \qquad \tau_{3^i}(x) \overset{\text{decode}}{\rightarrow} \begin{bmatrix}\texttt{rot}_i(\vec{x}_L) \\ \texttt{rot}_i(\vec{x}_R)\end{bmatrix}, \qquad \tau_{n-1}(x) \overset{\text{decode}}{\rightarrow} \begin{bmatrix}\vec{x}_R \\ \vec{x}_L\end{bmatrix},$$

for the Galois automorphism $\tau_i : a(X) \mapsto a(X^i)$.

Parallelizing Two Cipher Evaluations. In both state-of-the-art integer HE cryptosystems (BFV and BGV) we can use this inner structure of power-of-two homomorphic ciphertexts to parallelize both branches of our cipher. When encrypting the secret key and encoding vectors in the affine layer, one has to encode the vectors affecting the first branch of the cipher into the first row of the homomorphic ciphertext, and vectors affecting the second branch into the second row. As a result, all homomorphic operations are applied in parallel to both branches.

[11] https://homomorphicencryption.org/

Efficient Linear Layer. For security, we have to mix both branches of our cipher after each affine transformation. An efficiently implementable linear layer, which is also invertible, is the following matrix multiplication:

$$\begin{bmatrix}\vec{y}_L\\ \vec{y}_R\end{bmatrix} = \begin{bmatrix}2\cdot I & I\\ I & 2\cdot I\end{bmatrix}\cdot\begin{bmatrix}\vec{x}_L\\ \vec{x}_R\end{bmatrix} = \begin{bmatrix}\vec{x}_L\\ \vec{x}_R\end{bmatrix} + \begin{bmatrix}\vec{x}_L\\ \vec{x}_R\end{bmatrix} + \begin{bmatrix}\vec{x}_R\\ \vec{x}_L\end{bmatrix},$$

where I is the $t \times t$ identity matrix. This can be implemented by two homomorphic additions and a homomorphic rotation.

Cost Comparison. In Table 11 we compare the cost of the new linear layer (two parallel instances of state size t) to the cost of one larger linear layer of size $s = 2\cdot t$. The new linear layer effectively requires half the homomorphic additions and multiplications, and choosing t such that it splits nicely into $t = t_1 \cdot t_2$ the number of rotations is also halved.

Table 11: Homomorphic operations and multiplicative depth of the linear layers, with $t = t_1 \cdot t_2$ and $2\cdot t = s_1 \cdot s_2$.

Linear Layer	pt-ct Add	ct-ct Add	pt-ct Mul	ct-ct Mul	Rot	pt-ct Depth	ct-ct Depth
Split and Mix	1	$t+2$	t	-	t_1+t_2	1	-
No Splitting	1	$2\cdot t$	$2\cdot t$	-	s_1+s_2-1	1	-

5.4 Total Homomorphic Operations and Multiplicative Depth

In Table 12 we summarize the number of homomorphic operations and the multiplicative depth of each individual part of our resulting new cipher, dubbed Pasta, as well as the total count for Pasta-3 (3 rounds) and Pasta-4 (4 rounds). The table also highlights that the multiplicative depth of Pasta, and therefore its noise consumption, only depends on the number of rounds. Further, the runtime of homomorphically evaluating Pasta is dominated by the affine layer and scales with the state size and the number of rounds.

Table 12: Homomorphic operations and multiplicative depth of Pasta, with $t = t_1 \cdot t_2$.

	pt-ct Add	ct-ct Add	pt-ct Mul	ct-ct Mul	Rot	pt-ct Depth	ct-ct Depth
Affine	1	t	t	-	t_1+t_2-1	1	-
Mix	-	2	-	-	1	-	-
S'	-	1	1	1	1	1	1
S	-	-	-	2	-	-	2
Round	1	$t+3$	$t+1$	1	t_1+t_2+1	2	1
Last Round	1	$t+2$	t	2	t_1+t_2	1	2
Pasta-3	4	$4t+10$	$4t+2$	4	$4(t_1+t_2)+2$	6	4
Pasta-4	5	$5t+13$	$5t+3$	5	$5(t_1+t_2)+3$	8	5

5.5 Packed vs. Word-sliced Implementation

In the previous sections, we describe efficient SIMD algorithms to evaluate Pasta on a packed HE ciphertext. In this section, we want to compare them to a word-sliced implementation where one would encrypt only one field element $\in \mathbb{F}_p$ into one HE ciphertext.

A word-sliced implementation has several disadvantages. First, the homomorphic evaluation time of Pasta would be much slower. In a packed implementation, the S-boxes can be evaluated with $\mathcal{O}(1)$ homomorphic operations, and with $\mathcal{O}(t)$ HE operations in a word-sliced implementation. The word-sliced affine layer requires $\mathcal{O}(t^2)$ HE operations compared to $\mathcal{O}(t)$ operations when using packing. Secondly, the initial setup in the HHE use case requires the transmission of the HE encrypted symmetric key. In a packed implementation, this is always only one HE ciphertext. However, in a word-sliced implementation, on has to transmit $2 \cdot t$ HE ciphertexts, drastically increasing the communication cost of this setup phase. Finally, if the HHE use case leverages packing, one has to reconstruct a packed ciphertext from its word-sliced state using many rotations on the server.

However, word-sliced implementations have an advantage as well. They do not require homomorphic rotations (and, therefore, no Galois keys) and one can access each word of the state individually. This is why one can implement the S-boxes from Section 5.2 without requiring masking multiplications. As a consequence, word-sliced implementations have less noise consumption. Splitting the state in our Pasta design is also beneficial for word-sliced implementations, since it reduces the number of homomorphic multiplications from $(2 \cdot t)^2$ to $2 \cdot t^2$ per affine layer, reducing the runtime.

5.6 Truncation vs. Feed-Forward

The original Rasta design uses a feed-forward addition of the secret key to prevent meet-in-the-middle (MITM) attacks. Alternatively, one can truncate parts of the state to prevent these attacks (Section 7.1). Since our design already truncates t words of the state, we do not require an additional feed-forward operation.

6 Pasta Specification

Here we provide the complete Pasta specification. Pasta is a family of stream ciphers which applies a permutation to the secret key, followed by a truncation to produce the final keystream. The design of Pasta is shown in Figure 4.

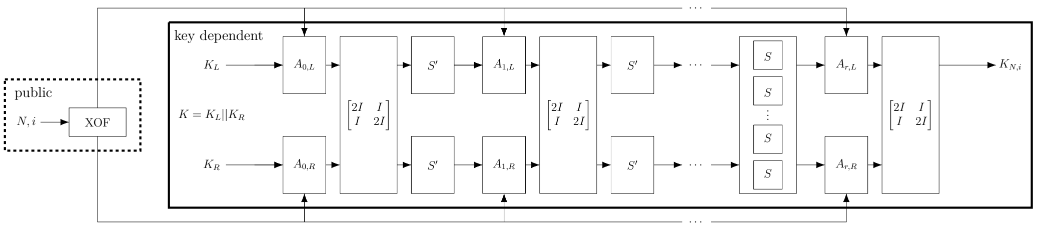

Figure 4: The r-round Pasta construction to generate the keystream $K_{N,i}$ for block i under nonce N with affine layers A_j.

For a prime p s.t. $\gcd(p-1, 3) = 1$, Pasta is defined as

$$\vec{x} \mapsto \text{left}_t(\Pi(\vec{x})),$$

where $\vec{x} \in \mathbb{F}_p^{2t}$, $\text{left}_t(\cdot)$ returns the first t words, and Π is defined over $\mathbb{F}_p^{2t}$ as

$$\Pi(\cdot) = A_r \circ S_{\text{cube}} \circ A_{r-1} \circ S_{\text{feistel}} \circ A_{r-2} \ldots \circ A_1 \circ S_{\text{feistel}} \circ A_0(\cdot),$$

where $r \geq 1$ is the number of rounds and where

- S_{feistel} is an S-box layer defined as

$$S_{\text{feistel}}(\vec{x}) = S'(\vec{x}_L) \| S'(\vec{x}_R),$$

where S' over $\mathbb{F}_p^t$ is a Feistel structure defined as

$$[S'(\vec{y})]_i = \begin{cases} y_0 & \text{if } i = 0, \\ y_i + (y_{i-1})^2 & \text{otherwise,} \end{cases}$$

where $\vec{y} = y_0 \| y_1 \| \cdots \| y_{t-1} \in \mathbb{F}_p^t$,

- S_{cube} is an S-box defined as

  $$S_{\text{cube}}(\vec{x}) = S(x_0)\|S(x_1)\|\cdots\|S(x_{s-1}),$$

  where $\vec{x} = x_0\|x_1\|\cdots\|x_{s-1} \in \mathbb{F}_p^{2t}$ and where S over $\mathbb{F}_p$ is the cube[12] S-box $S(y) = y^3$, and

- for each $i \in \{0, \ldots, r\}$, A_i is an affine layer defined as

  $$A_i(\vec{x}) = \begin{bmatrix} 2 \cdot I & I \\ I & 2 \cdot I \end{bmatrix} \times \begin{bmatrix} A_{i,L}(\vec{x}_L) \\ A_{i,R}(\vec{x}_R) \end{bmatrix},$$

  where $I \in \mathbb{F}_p^{t\times t}$ is the identiy matrix and where

  $$A_{i,j}(\vec{y}) = M_{i,j} \times \vec{y} + \vec{c_{i,j}}$$

  for $j \in \{L, R\}$ and for each $\vec{y} \in \mathbb{F}_p^t$, where

  - $M_{i,L}, M_{i,R} \in \mathbb{F}_p^{t\times t}$ are invertible matrices defined in the following, and
  - $\vec{c}_{i,L}, \vec{c}_{i,R} \in \mathbb{F}_p^t$ are random vectors.

  $M_{i,L}, M_{i,R} \in \mathbb{F}_p^{t\times t}$ and $\vec{c}_{i,L}, \vec{c}_{i,R} \in \mathbb{F}_p^t$ are generated for each round from an XOF seeded with a nonce N and a counter i.

To efficiently sample each matrix $M_{i,j} \in \mathbb{F}_p^{t\times t}$, we sample sequential matrices following [GPP11, GPPR11]. For each $j \in \{L, R\}$, we define $M_{i,j} := \left(\tilde{M}_{i,j}\right)^t$, where $\tilde{M}_{i,j} \in \mathbb{F}_p^{t\times t}$ is defined as

$$\tilde{M}_{i,j} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \alpha_3 & \cdots & \alpha_t \end{bmatrix}$$

for $\alpha_1, \ldots, \alpha_t \in \mathbb{F}_p \setminus \{0\}$. $M_{i,j}$ is an invertible matrix which can be built by sampling t random elements and performing $t \cdot (t-1)$ multiplications and $(t-1)\cdot(t-1)$ additions.

6.1 Concrete Instances

We propose a 3-round instance Pasta-3 as well as a 4-round instance Pasta-4 using Shake128 [NIS15] as XOF. These instances provide at least 128 bits of security for the prime fields $\mathbb{F}_p$ with $\log_2(p) > 16$ and $\gcd(p-1, 3) = 1$. The block and key sizes are shown in Table 13.

Table 13: Concrete 128 bit security instances of Pasta.

Instance	Rounds	# Key Words	# Plain Words	# Cipher Words	XOF
Pasta-3	3	256	128	128	Shake128
Pasta-4	4	64	32	32	Shake128

[12]We recall that the S-box $S(x) = x^d$ for $d \geq 2$ is invertible over $\mathbb{F}_p$ if and only if $\gcd(p-1, d) = 1$.

6.2 From Rasta to Pasta

In summary, Pasta is inspired by a port of Rasta to $\mathbb{F}_p^t$, where the following modifications have been made:

1. *The feed-forward operation has been replaced by a truncation operation.* This allows to prevent MITM attack in a more efficient way, at the cost of using a larger state. In the packed HE evaluation the truncated words, however, do not influence the runtime since they can be evaluated simultaneously to the non-truncated part of the state.
2. *Pasta uses two different kind of S-Box layers.* This is motivated by the desire of reducing the number of rounds while maintaining a reasonable state size. Having $r-1$ Feistel S-boxes (inspired by the chi function) and a final cube S-box with higher degree and depth allows us to build Pasta instances with comparable number of plain/cipher words as Masta with one round less. This implies both, a faster homomorphic evaluation time, as well as less noise consumption compared to Masta. We further explore the choice of two different S-boxes in Appendix C.
3. *The Linear Layer is much cheaper than the one used in Rasta.* Indeed, instead of generating a $2t \times 2t$ random invertible matrix directly, we pick up $2t$ random elements and construct two sequential matrices $M_i \in \mathbb{F}_p^{t \times t}$ as given in [GPP11, GPPR11]. These two matrices are then combined into one $2t \times 2t$ matrix via a cheap mixing operation. Having two split matrices allows an efficient packed homomorphic evaluation of Pasta.

We point out that these changes are a trade-off between the security and the efficiency reasons, as shown in the previous section.

7 Pasta Security Analysis

Given a certain number of rounds (fixed in advance), our goal is to find the minimum number of key words $s = 2t$ for which we can guarantee security of at least κ bits. If not specified otherwise, $\kappa \approx \log_2(p^s)$. This is slightly different from what is usually done in traditional symmetric cryptanalysis. Indeed, in general, given a state $\mathbb{F}_p^s$ and a security level κ, one looks for the minimum number of rounds which provide a security level of at least κ bits. Here we modify the approach since one of our main goals is to keep the depth as low as possible, focusing on 3 and 4 rounds.

7.1 Truncation vs. Feed-Forward

Consider a permutation $F : \mathbb{F}_p^s \to \mathbb{F}_p^s$, and assume it can be split as $F(\cdot) = F_2 \circ F_1(\cdot)$. The advantage of a truncation w.r.t. a feed-forward operation is that it prevents attacks using the backward direction without requiring a high degree of the inverse round function. Indeed, in the feed-forward case, given $y = F(x) + x$, one can set up a system of equations of the form

$$F_1(x) = F_2^{-1}(y - x).$$

In order to prevent the possibility to solve it using algebraic techniques (e.g., Gröbner bases), we need that both F_1 and F_2^{-1} have a high degree.

In the case of truncation, given $y = \text{left}_t(F(x))$, the system of equations becomes

$$F_1(x) = F_2^{-1}(y \,||\, y')$$

for a certain unknown $y' \in \mathbb{F}_p^t$. If t is large enough, the cost of solving it exceeds the security level. At the same time, the overall size of the state must be larger than in the feed-forward case due to losing part of the state.

7.2 Security of the Linear Layer

As the smaller linear layers, we use random invertible matrices over $\mathbb{F}_p^t$. To achieve security, the linear branch number of these matrices has to be sufficiently high. We analyze this property by bounding the probability that a randomly picked matrix M allows for transitions on the t-element mask vectors α to β, $\alpha = M^T\beta$, where α and β have many zeros. An overview of correlation analysis in $\mathbb{F}_p$ can be found in [DGGK21].

In Appendix B, we prove the following result.

Proposition 1. *Let $M \in \mathbb{F}_p^{t\times t}$ be an invertible matrix. Its branch number satisfies*

$$\Pr[\textit{branch number} \geq t/2] \geq 1 - \frac{2}{p^{t/2-2}},$$

for $p > t \geq 6$.

Note that $\Pr[\text{branch number} \geq t/2] \approx 1$ for $p \gg t \geq 6$.

However, in our case, the total number of sequential matrices that we can generate is limited by the t elements α_i we can choose. Hence, in total we can generate $\hat{\kappa} = (p-1)^t$ invertible matrices. Considering this special case, we get that

$$\Pr[\text{branch number} \geq z] \geq 1 - \frac{2zp^z \cdot t^t}{(p-1)^t},$$

that is,

$$\Pr[\text{branch number} \geq t/2] \geq 1 - \left(\frac{2t^2}{p-1}\right)^{t/2},$$

using $2(p-1) \leq p$. Hence, $\Pr[\text{branch number} \geq t/2] \approx 1$ for $p \gg 2t^2$, as in our case.

7.3 Algebraic Attacks

To describe our analysis, we focus on Pasta-3. Our input x consists of $s = 2t$ unknown key elements and the output y consists of t elements (after truncation). Hence, we have

$$\begin{aligned} x &= k_1 \,||\, k_2 \,||\, \cdots \,||\, k_s, \\ y &= \textsc{Pasta}(x) = o_1 \,||\, o_2 \,||\, \cdots \,||\, o_t. \end{aligned}$$

7.3.1 Linearization

In a linearization approach, the attacker replaces all monomials of degrees greater than 1 by new variables, and finally tries to solve the resulting system of linear equations. Assuming n_v variables and a maximum degree of d, the number of possible monomials is

$$N = \sum_{i=1}^{d} \binom{n_v + i - 1}{i}. \tag{5}$$

For Pasta-3 we have $d = 12$, and hence s variables in degree 12 after one function call. Further, we obtain t equations with each call. In order to get as many equations n_e as variables n_v, we can simply request more data, which eventually results in $n_e = n_v$ after $s/t = 2$ blocks (this has no effect on the efficiency of the linearization).

Due to the complexity of solving a linear system of equations in N variables, we target $\log_2(N) > 64$. Hence, $s \geq 207$ for a security of 128 bits. Following the same analysis, we need $s \geq 51$ for Pasta-4 and $s \geq 101$ for a Masta-like 4-round instance which uses only degree-2 Feistel-like S-boxes.

In this analysis, we assume that almost all monomials appear in the final representations, since our design provides strong diffusion in half of the state by using dense invertible matrices, and full diffusion after two full linear layers.I n order to get more confidence in our design, we also did some practical tests abd show the results in Figure 5. To avoid the effect of cancellations, we used prime numbers of sizes larger than 2^{16}. We observe that for the state sizes we tested, the actual number of monomials in the output word with the smallest number of monomials is always very close to the upper bound for the number of monomials given in Eq. (5).

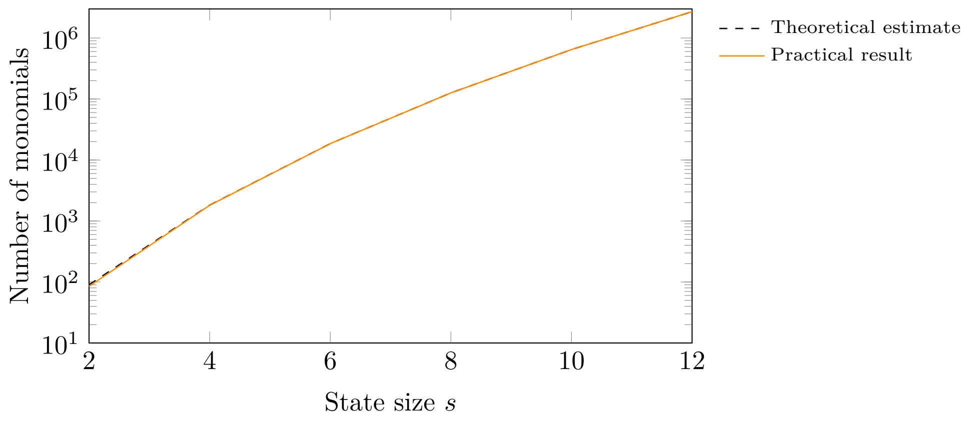

Figure 5: Comparison of the estimated number of monomials in each of the output words according to Eq. (5) and the lowest number of monomials found in a practical evaluation.

7.3.2 Gröbner Basis Attacks

Here we determine how large our key s has to be in order to provide security w.r.t. Gröbner basis attacks up to a complexity of 2^{128} function calls. As was the case above, we can simply generate sufficiently many equations by requesting at least $s/t = 2$ blocks. Hence, $n_v = n_e$, and we can estimate the complexity of solving such a system of equations by using theoretical bounds. However, these bounds assume a regular system of equations, and in practical tests we quickly observed that this is not the case for Pasta. Indeed, when building more full-round equations and hence an overdetermined system, we can force the degree of regularity to reach a minimum of 12. By reusing the estimate for the complexity of computing a Gröbner basis we need $s \geq 207$. Similar results can be obtained by assuming $d = 24$ for Pasta-4.

There is also a different way to argue the number of words to use. From the linearization analysis we know that there will be roughly 2^{64} different monomials in each of the resulting equations. Due to the internals of Gröbner basis algorithms, this results in around $\left(2^{64}\right)^{\omega}$ operations being necessary to compute a basis.[13] We pessimistically (from a designer's point of view) set $\omega = 2$ and thus have $\left(2^{64}\right)^{\omega} = 2^{128}$.

Additional Strategies. The strategy presented above is only one way to attack the system using Gröbner bases. It is common to also consider approaches which introduce new variables in each state. The main idea of this technique is to reduce the degrees of the equations at the expense of more variables, which is particularly useful when trying to represent high-degree equations in a more efficient way.

[13]For example, the F5 algorithm [Fau99] uses Gaussian elimination on a Macaulay matrix, whose rows indicate the equations in the system and whose columns are indexed by the monomials in these equations.

In more detail, we may introduce a new variable after each nonlinear operation. Considering a total state size of $s = 2t$ words, we need to introduce $2s(r-1)$ new variables for an r-round construction (note that no new variables are needed after the final round, since the stream output added to a plaintext is a degree-3 combination of the previous variables). Using this many variables and equations of a degree larger than or equal to 2 results in a high solving complexity when assuming nontrivial (i.e., dense) equations (we refer to [JV17, NNY18], in which degree-2 equation systems over $\mathbb{Z}_2$ are considered). We therefore conjecture that introducing intermediate variables will only increase the complexity needed to solve the final system when compared to using full-round equations.

7.4 Other Attacks

Many other known attacks are largely prevented by our random linear layers which are different in each Pasta evaluation, which is the same strategy used by e.g. Rasta and Masta.

Higher-Order Differential Attacks. Higher-order differential attacks [Lai94, Knu94] are essentially prevented by the fact that the attacker is only allowed to evaluate a single instance once due to the different linear layers. Moreover, the only subspaces of a finite field $\mathbb{F}_p$ with prime characteristic are $\{0\}$ and $\mathbb{F}_p$ itself, which makes higher-order differential attacks even harder (however, there have been variations of this attack vector which also work over $\mathbb{F}_p$ [BCD+20a]).

Differential Attacks. Inherently, classical differential attacks [BS90] are built upon a suitable differential characteristic. In Pasta, this is also prevented by the use of different linear layers for each instance. Indeed, every permutation can only by evaluated once, making this attack vector infeasible just as in the original Rasta.

7.5 Security Margin

Besides the attack strategies just discussed, we add a security margin to our construction. Concretely, we take the largest number of words s needed for security, we multiply this number by 1.2 for a 20% security margin, and we then take the smallest even integer larger than or equal to that.

8 Pasta Benchmarks

In this section, we benchmark a packed implementation of our Pasta design in both SEAL and HElib. We also reimplemented a packed version of Masta, using the same algorithms to generate random field elements and homomorphic matrix multiplications as in Pasta to compare both ciphers in a fair setting. Similar as in Section 4.1.1, we also benchmark both ciphers in a real HHE use case.

8.1 Comparing Pasta to $\mathbb{Z}_2$ Ciphers

We first compare Pasta and Masta to the $\mathbb{Z}_2$ benchmarks from Section 4. Therefore, we instantiate both ciphers with a 17-bit prime and benchmark their performance for the small use case from Section 4.1.1. The resulting benchmarks can be seen in Table 14 and Table 15 where we depict both runtime and remaining noise budget after each step of the HHE use case for SEAL and HElib respectively.

Table 14: Runtime and noise budget of the small HHE use case in the SEAL library.

Cipher	N	Enc. Key		Decomp.		Small Use Case	
		runtime	noise	runtime	noise	runtime	noise
		s	bit	s	bit	s	bit
$p = 65537$ (17 bit):							
Pasta-3	16384	**0.017**	364	**9.28**	95	**0.197**	51
Pasta-4	32768	0.059	800	21.0	451	1.11	406
Masta-4	32768	0.058	800	54.2	460	1.11	415
Masta-5	32768	0.057	800	39.2	386	1.13	341

Table 15: Runtime and noise budget of the small HHE use case in the HElib library.

Cipher	m	λ'	Enc. Key		Decomp.		Small Use Case	
			runtime	noise	runtime	noise	runtime	noise
		bit	s	bit	s	bit	s	bit
$p = 65537$ (17 bit):								
Pasta-3	65536	173	0.054	410	26.0	74	0.754	23
Pasta-4	65536	142	**0.054**	475	**13.0**	56	**0.737**	6
Masta-4	65536	133	0.054	502	36.7	86	0.740	36
Masta-5	131072	254	0.116	566	55.4	56	1.71	5

8.1.1 Discussion

In the following, we compare the runtime and noise consumption of all $\mathbb{Z}_2$ and $\mathbb{F}_p$ ciphers, with $p = 65537$, namely

- in Figure 6 for homomorphically decrypting one block in SEAL ($\mathbb{F}_p$ values from Section 8.2),
- in Figure 7 for the HHE use case (including HHE decompression) in SEAL,
- in Figure 8 for homomorphically decrypting one block in HElib ($\mathbb{F}_p$ values from Section 8.2), and
- in Figure 9 for the HHE use case (including HHE decompression) in HElib.

Our figures indicate that Pasta is always the fastest cipher – mainly Pasta-4 due to the small number of encrypted words. However, Pasta-3 is faster when evaluating the whole HHE use case in SEAL due to the small multiplicative depth requiring smaller HE parameters for security. Comparing Pasta to the $\mathbb{Z}_2$ ciphers, one can observe that homomorphically decrypting one block requires less noise budget for the $\mathbb{Z}_2$ ciphers. However, Pasta has (besides the runtime advantage) a noise advantage over the $\mathbb{Z}_2$ ciphers when considering the HHE use case due to the significantly larger multiplicative depth of the binary circuits for integer arithmetic. Concretely, decompression and use case evaluation is $33\times$ faster in SEAL using Pasta-3 and $82\times$ faster in HElib using Pasta-4 compared to Agrasta. Using TFHE for $\mathbb{Z}_2$ ciphers instead of e.g. SEAL does not help the $\mathbb{Z}_2$ ciphers either, since Pasta-3 in SEAL is $47\times$ faster than using Kreyvium in TFHE for the small HHE use case.

Increasing the bitsize of the encrypted integers or chaining multiple matrix multiplications would further demonstrate the advantage of Pasta over $\mathbb{Z}_2$ ciphers, since the drastic increase in the multiplicative depth of the use case would make using the $\mathbb{Z}_2$ ciphers infeasible.

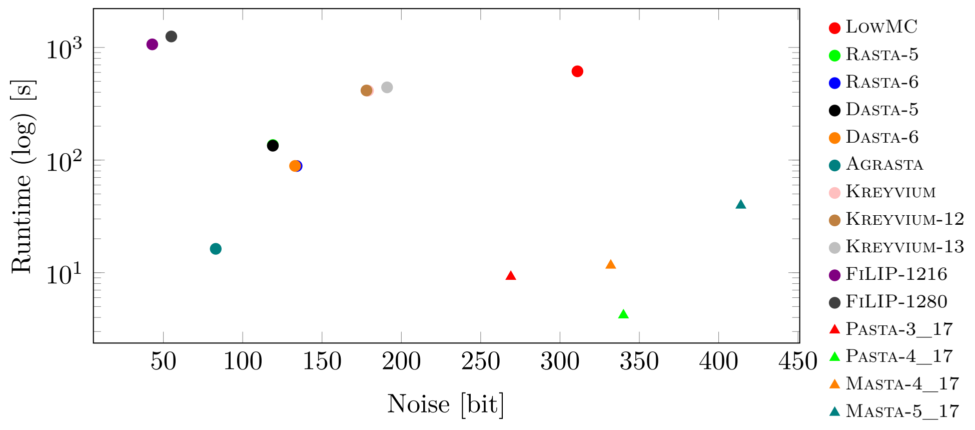

Figure 6: Runtime and noise comparison of $\mathbb{Z}_2$ ciphers for homomorphically decrypting 1 Block in SEAL.

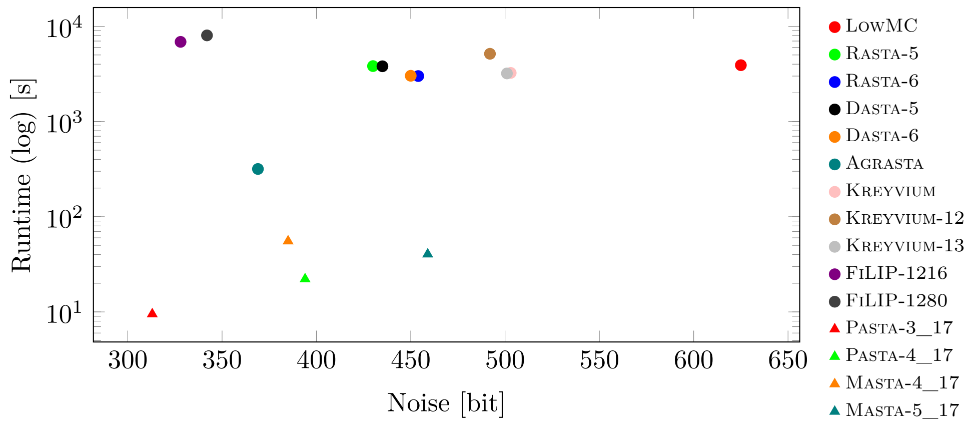

Figure 7: Runtime and noise comparison for the small HHE use case in SEAL.

8.2 Pasta vs. Masta

Since both Pasta and Masta outperform the $\mathbb{Z}_2$ ciphers for HHE, we continue with comparing these ciphers over $\mathbb{F}_p$. Similar to the $\mathbb{Z}_2$ benchmarks, we also compare Pasta and Masta in a real HHE use case. However, to further demonstrate the advantage of the $\mathbb{F}_p$ ciphers in HHE, we benchmark a more extensive use case with a significantly higher multiplicative depth. The use case this time is an application with three affine transformation $\vec{x}_i' = M_i \cdot \vec{x}_i + \vec{b}_i$, where $\vec{x}_i, \vec{x}_i', \vec{b}_i \in \mathbb{F}_p^{200}$ and $M_i \in \mathbb{F}_p^{200\times 200}$, with three different primes p. To make the use case more generic, we square the output vector after the first two affine transformation, resulting in a multiplicative depth of 3 plaintext-ciphertext and 2 ciphertext-ciphertext multiplications.

8.2.1 Plain Benchmarks

In Table 16 we compare the number of CPU cycles of the encryption curcuit of Pasta to the encryption circuit of Masta. Since both ciphers generate random matrices and round constants independent of the secret key which can be precomputed before encryption, we additionally give CPU cycles for generating these affine layers and the encryption circuit with precomputed randomness. Table 16 shows that Pasta-4, due to its small state size,

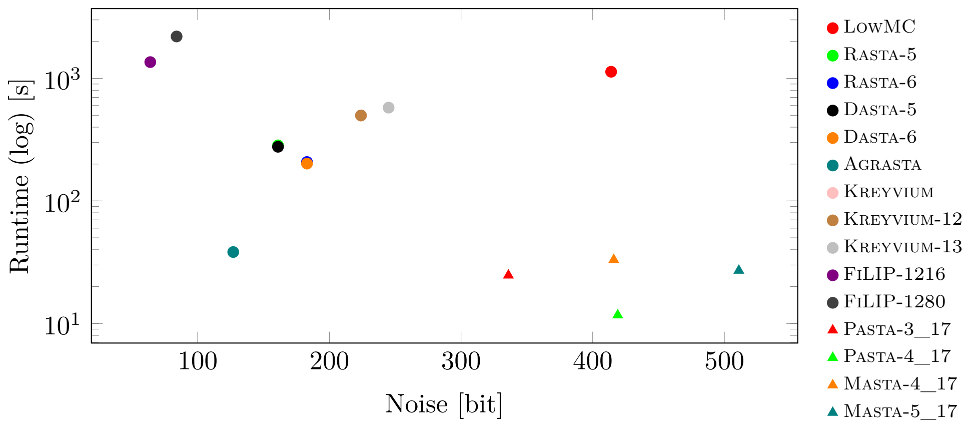

Figure 8: Runtime and noise comparison of $\mathbb{Z}_2$ ciphers for homomorphically decrypting 1 Block in HElib.

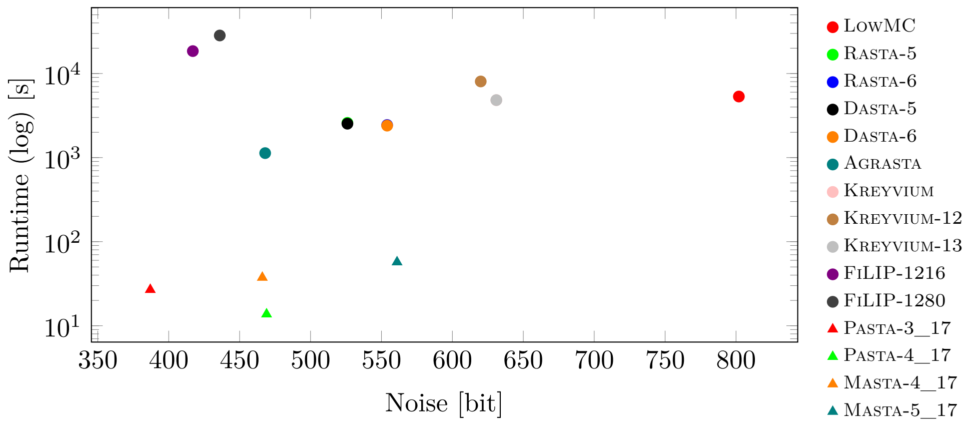

Figure 9: Runtime and noise comparison for the small HHE use case in HElib.

requires the smallest number of cycles to encrypt one block. Pasta-3, on the other hand, due to sampling sequential matrices instead of polynomials $m \in \mathbb{Z}_p[X]/(X^t - \alpha)$ (as in Masta) and requiring twice as many matrices per round, is the slowest cipher to encrypt one block in plain. However, the difference to Masta-4 is only a factor of 3, which in practice corresponds to latencies in the order of milliseconds.

8.2.2 SEAL Benchmarks

In Table 17 we present the benchmarks for the packed implementation of Pasta and Masta in the SEAL library. We give timings for homomorphically decrypting one block and additionally timings for the bigger HHE use case. We parameterize SEAL to provide 128 bits of security and use the smallest N allowing enough noise budget for correct evaluation. In Table 18 we additionally give the remaining noise budget after encrypting the symmetric key, homomorphically decrypting the symmetric ciphertexts, and performing the HHE use case.

Table 16: Cycles for encrypting one block in plain, averaged over 1000 executions.

Cipher	Total	Affine Generation	Encrypting
$p = 65537$ (17 bit):			
Pasta-3	17 041 380	9 196 314	7 845 066
Pasta-4	**1 363 339**	825 067	538 272
Masta-4	6 535 937	2 164 002	4 371 935
Masta-5	2 105 628	752 374	1 353 254
$p = 8088322049$ (33 bit):			
Pasta-3	22 429 444	11 637 800	10 791 644
Pasta-4	**1 750 420**	973 205	777 215
Masta-4	8 427 384	1 975 522	6 451 862
Masta-5	2 690 636	674 201	2 016 435
$p = 1096486890805657601$ (60 bit):			
Pasta-3	31 053 515	16 067 138	14 986 377
Pasta-4	**2 458 680**	1 315 770	1 142 910
Masta-4	11 405 862	1 968 100	9 437 762
Masta-5	3 542 410	669 220	2 873 190

8.2.3 HElib Benchmarks

In Table 19 we present the benchmarks for the packed implementation of Pasta and Masta in the HElib library. We give timings for homomorphically decrypting one block and additionally timings for the bigger HHE use case. We parameterize q to provide enough noise budget to evaluate the benchmark, and choose the m-th cyclotomic reduction polynomial, with m being a power of two, such that the HE scheme provides ≥ 128 bits security. In Table 20 we additionally give the remaining noise budget after encrypting the symmetric key, homomorphically decrypting the symmetric ciphertexts, and performing the HHE use case.

8.2.4 Discussion

In the following figures we compare the runtime and noise consumption of Pasta and Masta for 3 different prime fields $\mathbb{F}_p$, namely

- in Figure 10 for homomorphically decrypting one block in SEAL,
- in Figure 11 for the HHE use case (including HHE decompression) in SEAL,
- in Figure 12 for homomorphically decrypting one block in HElib, and
- in Figure 13 for the HHE use case (including HHE decompression) in HElib.

The figures show the advantage of Pasta compared to Masta. In all figures, Pasta-3 has a smaller runtime and noise consumption then Masta, especially when the smaller multiplicative depth allows for smaller HE parameters (compare, e.g., 33-bit prime fields in Figure 11, where Pasta-3 is $6\times$ faster than Masta-4). Pasta-3 is only outperformed by Pasta-4 for a small number of encrypted words (e.g., only encrypting one block, or the small HHE use case from Section 8.1 in HElib where Pasta-4 is $2.7\times$ faster then Masta-4) if the overall multiplicative depth allows Pasta-4 to use the same HE parameters as Pasta-3. Hence, we propose using Pasta-4 for HHE use cases with a small number of encrypted words, and Pasta-3 everywhere else. Since Masta uses different security

Table 17: $\mathbb{F}_p$ benchmarks for the SEAL library.

Cipher	1 Block			Bigger HHE use case			
	N	Enc. Key s	Decomp. s	N	Enc. Key s	Decomp. s	Use Case s
$p = 65537$ (17 bit):							
Pasta-3	16384	0.016	9.22	32768	**0.056**	**86.2**	**43.9**
Pasta-4	16384	**0.016**	**4.19**	32768	0.057	147.8	43.8
Masta-4	16384	0.016	11.6	32768	0.058	108.7	43.9
Masta-5	32768	0.062	39.6	32768	0.056	157.0	43.9
$p = 8088322049$ (33 bit):							
Pasta-3	32768	0.057	43.1	32768	**0.055**	**86.3**	**43.9**
Pasta-4	32768	**0.057**	**21.2**	65536	0.216	833.4	220.8
Masta-4	32768	0.058	54.4	65536	0.215	568.5	221.3
Masta-5	32768	0.055	39.3	65536	0.215	852.6	220.7
$p = 1096486890805657601$ (60 bit):							
Pasta-3	32768	**0.055**	**58.3**	65536	**0.212**	**448.6**	**220.8**
Pasta-4	65536	0.220	119.2	65536	0.212	833.6	221.0
Masta-4	65536	0.220	284.3	65536	0.212	571.9	223.1
Masta-5	65536	0.219	213.3	65536	0.212	853.3	220.9

margins than Pasta, we discuss the impact of the security margin in Appendix D, which further demonstrates the advantage of the Pasta construction over Masta.

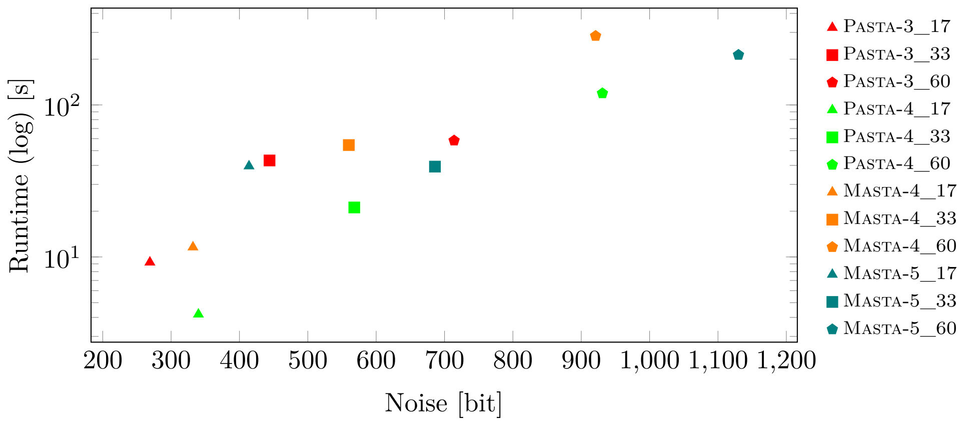

Figure 10: Runtime and noise comparison of $\mathbb{F}_p$ ciphers for homomorphically decrypting 1 Block in SEAL.

9 Conclusion

In this paper, we formally defined hybrid homomorphic encryption and evaluated existing ciphers for HHE. Since none of the existing designs is well-suited for HHE for integer arithmetic, we proposed Pasta, a low-depth cipher optimized for HHE. We evaluated the usability of Pasta against 7 ciphers for HHE in two state-of-the-art HE libraries, SEAL and HElib, and show, thate Pasta outperforms all previous designs in both homomorphic evaluation time and noise consumption.

Table 18: Noise budget after each operation for $\mathbb{F}_p$ ciphers in the SEAL library.

Cipher	1 Block			Bigger HHE use case			
	N	Enc. Key bit	Decomp. bit	N	Enc. Key bit	Decomp. bit	Use Case bit
$p = 65537$ (17 bit):							
Pasta-3	16384	365	96	32768	800	524	365
Pasta-4	16384	365	25	32768	800	451	292
Masta-4	16384	365	33	32768	800	459	300
Masta-5	32768	800	386	32768	800	386	226
$p = 8088322049$ (33 bit):							
Pasta-3	32768	783	339	32768	783	339	78
Pasta-4	32768	783	215	65536	1637	1058	793
Masta-4	32768	783	223	65536	1637	1068	802
Masta-5	32768	783	97	65536	1638	941	677
$p = 1096486890805657601$ (60 bit):							
Pasta-3	32768	756	42	65536	1611	888	461
Pasta-4	65536	1611	680	65536	1610	681	254
Masta-4	65536	1611	690	65536	1611	689	262
Masta-5	65536	1611	481	65536	1610	482	55

Acknowledgments

This work was supported by EU's Horizon 2020 project Safe-DEED under grant agreement n°825225, and by the "DDAI" COMET Module within the COMET – Competence Centers for Excellent Technologies Programme, funded by the Austrian Federal Ministry for Transport, Innovation and Technology (bmvit), the Austrian Federal Ministry for Digital and Economic Affairs (bmdw), the Austrian Research Promotion Agency (FFG), the province of Styria (SFG) and partners from industry and academia. The COMET Programme is managed by FFG. Lorenzo Grassi is supported by the European Research Council under the ERC advanced grant agreement under grant ERC-2017-ADG Nr. 788980 ESCADA.

Table 19: $\mathbb{F}_p$ benchmarks for the HElib library.

Cipher	1 Block				Bigger HHE use case				
	m	λ' bit	Enc. Key s	Decomp. s	m	λ' bit	Enc. Key s	Decomp. s	Use Case s
$p = 65537$ (17 bit):									
Pasta-3	65536	184	0.052	24.7	65536	128	**0.064**	**57.6**	**19.9**
Pasta-4	65536	163	**0.052**	**11.7**	131072	229	0.124	210.8	38.6
Masta-4	65536	163	0.062	33.1	131072	229	0.131	157.3	45.4
Masta-5	65536	133	0.064	27.1	131072	199	0.135	252.8	48.4
$p = 8088322049$ (33 bit):									
Pasta-3	65536	128	**0.057**	**28.7**	131072	162	**0.166**	**187.7**	**60.5**
Pasta-4	131072	204	0.166	35.3	131072	144	0.190	320.5	57.8
Masta-4	131072	196	0.165	101.3	131072	144	0.166	256.2	69.5
Masta-5	131072	166	0.168	82.4	131072[a]	117	0.242	427.8	80.0
$p = 1096486890805657601$ (60 bit):									
Pasta-3	131072	162	0.185	94.1	131072[a]	97	**0.285**	**268.8**	**84.4**
Pasta-4	131072	129	**0.183**	**50.5**	131072[a]	83	0.310	486.7	84.2
Masta-4	131072	129	0.208	144.7	131072[a]	83	0.289	387.4	101.6
Masta-5	131072[a]	99	0.233	122.1	131072[a]	70	0.300	635.5	111.9

[a] Further increasing m for security resulted in infeasibly long runtimes.

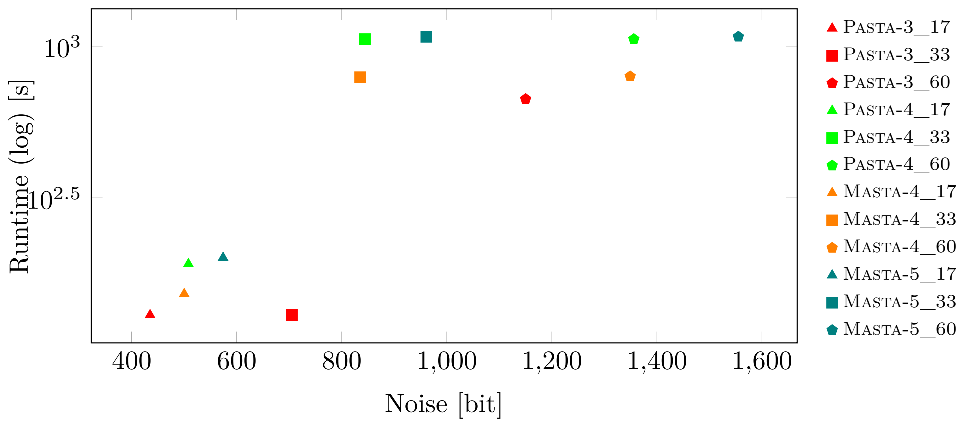

Figure 11: Runtime and noise comparison for the bigger HHE use case in SEAL.

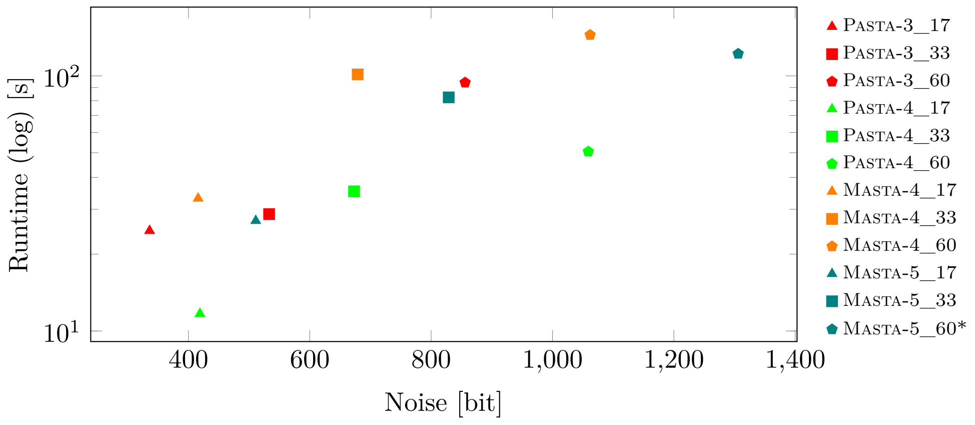

Figure 12: Runtime and noise comparison of $\mathbb{F}_p$ ciphers for homomorphically decrypting 1 Block in HElib. Ciphers marked with a * were evaluated with less than 128 bit HE security.

Table 20: Noise budget after each operation for $\mathbb{F}_p$ ciphers in the HElib library.

Cipher	1 Block				Bigger HHE use case				
	m	λ' bit	Enc. Key bit	Decomp. bit	m	λ' bit	Enc. Key bit	Decomp. bit	Use Case bit
$p = 65537$ (17 bit):									
Pasta-3	65536	184	368	32	65536	128	542	196	8
Pasta-4	65536	163	434	15	131072	229	645	205	14
Masta-4	65536	163	434	18	131072	229	645	215	24
Masta-5	65536	133	522	11	131072	199	715	205	15
$p = 8088322049$ (33 bit):									
Pasta-3	65536	128	535	2	131072	162	887	327	33
Pasta-4	131072	204	698	25	131072	144	987	314	20
Masta-4	131072	196	724	45	131072	144	987	309	16
Masta-5	131072	166	843	14	131072[a]	117	1161	347	53
$p = 1096486890805657601$ (60 bit):									
Pasta-3	131072	162	876	20	131072[a]	97	1366	505	49
Pasta-4	131072	129	1076	17	131072[a]	83	1567	496	40
Masta-4	131072	129	1076	14	131072[a]	83	1567	496	39
Masta-5	131072[a]	99	1356	50	131072[a]	70	1803	497	42

[a] Further increasing m for security resulted in infeasibly long runtimes.

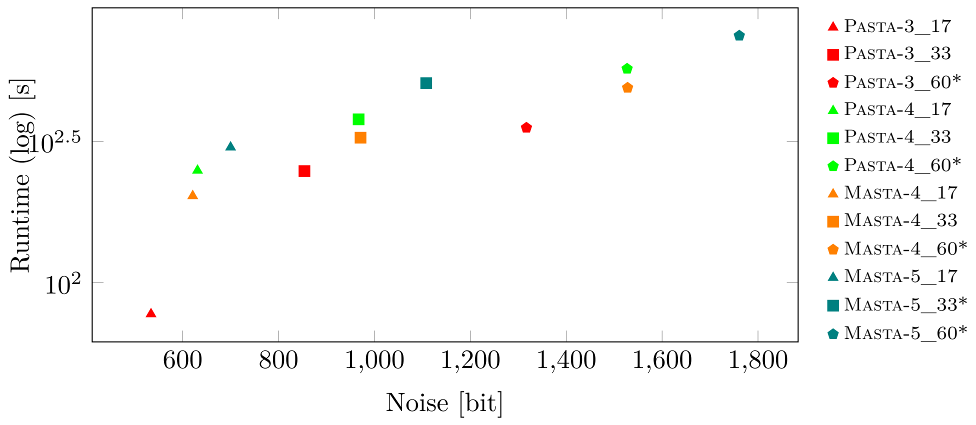

Figure 13: Runtime and noise comparison for the bigger HHE use case in HElib. Ciphers marked with a * were evaluated with less than 128 bit HE security.

References

[ARS^{+}15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *EUROCRYPT (1)*, volume 9056 of *LNCS*, pages 430–454. Springer, 2015.

[BBH^{+}20] Alexandros Bampoulidis, Alessandro Bruni, Lukas Helminger, Daniel Kales, Christian Rechberger, and Roman Walch. Privately connecting mobility to infectious diseases via applied cryptography. *IACR Cryptol. ePrint Arch.*, 2020:522, 2020.

[BCD^{+}20a] Tim Beyne, Anne Canteaut, Itai Dinur, Maria Eichlseder, Gregor Leander, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Yu Sasaki, Yosuke Todo, and Friedrich Wiemer. Out of oddity - new cryptanalytic techniques against symmetric primitives optimized for integrity proof systems. In *CRYPTO (3)*, volume 12172 of *LNCS*, pages 299–328. Springer, 2020.

[BCD^{+}20b] Fabian Boemer, Rosario Cammarota, Daniel Demmler, Thomas Schneider, and Hossein Yalame. MP2ML: a mixed-protocol machine learning framework for private inference. In *ARES*, pages 14:1–14:10. ACM, 2020.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325. ACM, 2012.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *CRYPTO*, volume 7417 of *LNCS*, pages 868–886. Springer, 2012.

[BS90] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In *CRYPTO*, volume 537 of *LNCE*, pages 2–21. Springer, 1990.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, pages 97–106. IEEE, 2011.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM J. Comput.*, 43(2):831–871, 2014.

[Can06] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In *ISC*, volume 4176 of *LNCS*, pages 171–186. Springer, 2006.

[CCF^{+}16] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In *FSE*, volume 9783 of *LNCS*, pages 313–333. Springer, 2016.

[CCK^{+}13] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi, and Aaram Yun. Batch fully homomorphic encryption over the integers. In *EUROCRYPT*, volume 7881 of *LNCS*, pages 315–335. Springer, 2013.

[CDK^{+}18] Benoît Cogliati, Yevgeniy Dodis, Jonathan Katz, Jooyoung Lee, John P. Steinberger, Aishwarya Thiruvengadam, and Zhe Zhang. Provable security of (tweakable) block ciphers based on substitution-permutation networks. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO*, volume 10991 of *LNCS*, pages 722–753. Springer, 2018.

[CDKS20] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient homomorphic conversion between (ring) LWE ciphertexts. *IACR Cryptol. ePrint Arch.*, 2020:15, 2020.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption library, August 2016. https://tfhe.github.io/tfhe/.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: fast fully homomorphic encryption over the torus. *J. Cryptol.*, 33(1):34–91, 2020.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 41–50. IEEE, 1995.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT (1)*, volume 10624 of *LNCS*, pages 409–437. Springer, 2017.

[CLT14] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Scale-invariant fully homomorphic encryption over the integers. In *Public Key Cryptography*, volume 8383 of *LNCS*, pages 311–328. Springer, 2014.

[Dae95] Joan Daemen. *Cipher and hash function design, strategies based on linear and differential cryptanalysis, PhD Thesis.* K.U.Leuven, 1995. http://jda.noekeon.org/.

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. Rasta: A cipher with low anddepth and few ands per bit. In *CRYPTO (1)*, volume 10991 of *LNCS*, pages 662–692. Springer, 2018.

[DGGK21] Christoph Dobraunig, Lorenzo Grassi, Anna Guinet, and Daniël Kuijsters. Ciminion: Symmetric encryption based on toffoli-gates over large finite fields. Cryptology ePrint Archive, Report 2021/267, 2021. https://eprint.iacr.org/2021/267, accepted at EUROCRYPT 2021.

[DR00] Joan Daemen and Vincent Rijmen. Rijndael for AES. In *AES Candidate Conference*, pages 343–348. National Institute of Standards and Technology,, 2000.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Information Security and Cryptography. Springer, 2002.

[Fau99] Jean-Charles Faugére. A new efficient algorithm for computing gröbner bases (f4). *Journal of Pure and Applied Algebra*, 139(1):61–88, 1999.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178. ACM, 2009.

[GGI+15] Craig Gentry, Jens Groth, Yuval Ishai, Chris Peikert, Amit Sahai, and Adam D. Smith. Using fully homomorphic hybrid encryption to minimize non-interative zero-knowledge proofs. *J. Cryptol.*, 28(4):820–843, 2015.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO*, volume 7417 of *LNCS*, pages 850–867. Springer, 2012.

[GPP11] Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON family of lightweight hash functions. In *CRYPTO*, volume 6841 of *LNCS*, pages 222–239. Springer, 2011.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In *CHES*, volume 6917 of *LNCS*, pages 326–341. Springer, 2011.

[HKC$^+$20] Jincheol Ha, Seongkwang Kim, Wonseok Choi, Jooyoung Lee, Dukjae Moon, Hyojin Yoon, and Jihoon Cho. Masta: An he-friendly cipher using modular arithmetic. *IEEE Access*, 8:194741–194751, 2020.

[HL20] Phil Hebborn and Gregor Leander. Dasta - alternative linear layer for rasta. *IACR Trans. Symmetric Cryptol.*, 2020(3):46–86, 2020.

[HS14] Shai Halevi and Victor Shoup. Algorithms in helib. In *CRYPTO (1)*, volume 8616 of *LNCS*, pages 554–571. Springer, 2014.

[HS15] Shai Halevi and Victor Shoup. Bootstrapping for helib. In *EUROCRYPT (1)*, volume 9056 of *LNCS*, pages 641–670. Springer, 2015.

[HS18] Shai Halevi and Victor Shoup. Faster homomorphic linear transformations in helib. In *CRYPTO (1)*, volume 10991 of *LNCS*, pages 93–120. Springer, 2018.

[HS20] Shai Halevi and Victor Shoup. Design and implementation of helib: a homomorphic encryption library. *IACR Cryptol. ePrint Arch.*, 2020:1481, 2020.

[JV17] Antoine Joux and Vanessa Vitse. A crossbred algorithm for solving boolean polynomial systems. In *NuTMiC*, volume 10737 of *LNCS*, pages 3–21. Springer, 2017.

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *USENIX*, pages 1651–1669. USENIX Association, 2018.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition.* CRC Press, 2014.

[Knu94] Lars R. Knudsen. Truncated and Higher Order Differentials. In *FSE 1994*, volume 1008 of *LNCS*, pages 196–211, 1994.

[Lai94] Xuejia Lai. *Higher Order Derivatives and Differential Cryptanalysis*, pages 227–233. 1994.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, volume 6110 of *LNCS*, pages 1–23. Springer, 2010.

[MCJS19] Pierrick Méaux, Claude Carlet, Anthony Journault, and François-Xavier Standaert. Improved filter permutators for efficient FHE: better instances and implementations. In *INDOCRYPT*, volume 11898 of *LNCS*, pages 68–91. Springer, 2019.

[Méa17] Pierrick Méaux. *Hybrid fully homomorphic framework. (Chiffrement complètement homomorphe hybride)*. PhD thesis, PSL Research University, France, 2017.

[MJSC16] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards stream ciphers for efficient FHE with low-noise ciphertexts. In *EUROCRYPT (1)*, volume 9665 of *LNCS*, pages 311–343. Springer, 2016.

[NIS15] NIST. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. National Institute of Standards and Technology (NIST), FIPS PUB 202, U.S. Department of Commerce, 2015.

[NLV11] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *CCSW*, pages 113–124. ACM, 2011.

[NNY18] Ruben Niederhagen, Kai-Chun Ning, and Bo-Yin Yang. Implementing joux-vitse's crossbred algorithm for solving MQ systems over GF(2) on gpus. In *PQCrypto*, volume 10786 of *LNCS*, pages 121–141. Springer, 2018.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.

[RAD78] R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93. ACM, 2005.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[SEA20] Microsoft SEAL (release 3.6). https://github.com/Microsoft/SEAL, November 2020. Microsoft Research, Redmond, WA.

A Proof HHE schemes are HE schmes

Lemma 1. *Let* HE *be a correct public-key homomorphic encryption scheme and* SYM *a correct secret-key encryption scheme. Then the resulting* HHE *scheme* HHE = (HHE.KGen, HHE.Enc, HHE.Dec, HHE.Eval) *is as well.*

Proof. The proof follows along the lines of [BV14]. Let m be an arbitrary message and let $\hat{c}$ be a ciphertext such that

$$\hat{c} = \mathsf{HHE.Decomp}_{\mathsf{evk}}(\mathsf{HHE.Enc}_{\mathsf{pk}}(m)).$$

To show correctness, we want to bound the following probability - for any function f - by a negligible function in the security parameter:

$$\Pr\left[\mathsf{HHE.Dec}_{\mathsf{sk}}\left(\mathsf{HHE.Eval}_{\mathsf{evk}}\left(f, \hat{c}\right)\right) \neq f(m)\right].$$

By definition, we have HHE.Eval = HE.Eval. If in addition we rewrite $\hat{c}$, we get

$$\Pr\left[\mathsf{HHE.Dec}_{\mathsf{sk}}\left(\mathsf{HE.Eval}_{\mathsf{evk}}\left(f, \mathsf{HHE.Decomp}_{\mathsf{evk}}(\mathsf{HHE.Enc}_{\mathsf{pk}}(m))\right)\right) \neq f(m)\right].$$

Looking into the internals of the encryption algorithm and decompression algorithm we first get

$$\Pr\left[\mathsf{HHE.Dec}_{\mathsf{sk}}\left(\mathsf{HE.Eval}_{\mathsf{evk}}\left(f, \mathsf{HHE.Decomp}_{\mathsf{evk}}(\mathsf{HE.Enc}_{\mathsf{pk}}(k), \mathsf{SYM.Enc}_{k}(m))\right)\right) \neq f(m)\right],$$

and finally arrive at

$$\Pr\left[\mathsf{HHE.Dec}_{\mathsf{sk}}\left(\mathsf{HE.Eval}_{\mathsf{evk}}\left(f, \mathsf{HE.Eval}_{\mathsf{evk}}(\mathsf{SYM.Dec}, \mathsf{HE.Enc}_{\mathsf{pk}}(k), \mathsf{SYM.Enc}_{k}(m))\right)\right) \neq f(m)\right].$$

Now, we can bound this probability by the union bound

$$\begin{aligned}\Pr\left[\mathsf{HHE.Dec}_{\mathsf{sk}}\left(\mathsf{HE.Eval}_{\mathsf{evk}}\left(f, \mathsf{HE.Enc}_{\mathsf{pk}}(m)\right)\right) \neq f(m)\right]& \\ + \Pr\left[\mathsf{SYM.Dec}_{k}\left(\mathsf{SYM.Enc}_{k}(m)\right) \neq m\right].&\end{aligned}$$

Both terms are negligible by assumption. The first one by the correctness of HE and the second one by the correctness of SYM. □

B About the Branch Number of a Bijective Matrix

Here we prove the following result.

Proposition 2. *Let $M \in \mathbb{F}_p^{s\times s}$ be an invertible matrix. Its branch number satisfies*

$$\Pr(\textit{branch number} \geq s/2) \geq 1 - \frac{2}{p^{s/2-2}},$$

for $p > s \geq 6$.

Proof. First of all, we are interested in the number $\hat{\kappa}$ of all possible bijective matrices M. A matrix M is bijective, if all its row vectors are linearly independent and different from the all 0 vector. So, for the first row, we have $p^s - 1$ possibilities to choose a row vector. For the second row, we have p^s possibilities to choose the coefficients minus p choices that is just are linear combination of the first row. In the third row, we now have $p^s - p^2$ choices etc. So we finally end up with

$$\hat{\kappa} = \prod_{i=0}^{s-1} \left(p^s - p^i\right) .$$

Now, we consider the number of matrices M, that allow a transition $\alpha = M^T\beta$ for fixed non-zero α and β. Considering the first row of M^T, we have p^{s-1} choices of the coefficients of the first row, so that β maps to the first coordinate of β. Since we require M to be bijective, the choice of the second row a limited to $p^{s-1} - p$. Following this reasoning, we get the number of matrices M that map α to β, i.e.,

$$\kappa = p^{s-1} \prod_{i=1}^{s-2} \left(p^{s-1} - p^i\right) = p \prod_{i=2}^{s-1} \left(p^s - p^i\right) .$$

Next, we have a look at how many different masks α and β exist, which have together z non-zero entries. This number is $(p-1)^z\binom{2s}{z}$. Now we have all ingredients we need to bound the probability that a randomly selected matrix M has a branch number smaller than z

$$\begin{aligned}
&\Pr(\text{branch number} \leq z) \leq \\
&\leq \frac{p \prod_{i=2}^{s-1} \left(p^s - p^i\right) \sum_{i=1}^{z} \left((p-1)^i \binom{2s}{i}\right)}{\prod_{i=0}^{s-1} \left(p^s - p^i\right)} \\
&\leq \frac{\sum_{i=1}^{z} \left((p-1)^i \binom{2s}{i}\right)}{\left(p^s - 1\right)\left(p^{s-1} - 1\right)} \\
&\leq \frac{z(p-1)^z \binom{2s}{s}}{\left(p^s - 1\right)\left(p^{s-1} - 1\right)} \\
&\leq \frac{2zp^z \cdot s^s}{\left(p^s - 1\right)\left(p^{s-1} - 1\right)} \,.
\end{aligned}$$

where $\binom{2s}{s} = \frac{2s\cdot(2s-1)\cdot\dots\cdot(s+1)}{s!} \leq \frac{(2s)^s}{2^{s-1}} = 2 \cdot s^s$ since $s! \geq 2^{s-1}$. We now set $z = s/2$ and assume that $p > s \geq 6$, we get

$$\begin{aligned}
\Pr(\text{branch number} \leq s/2) &\leq \frac{pp^{s/2}p^s}{\left(p^s - 1\right)\left(p^{s-1} - 1\right)} \\
&\leq \frac{16 \cdot p^{3s/2+1}}{9 \cdot p^{2s-1}} \leq \frac{2p^2}{p^{s/2}} \leq 1/2,
\end{aligned}$$

where $(x^s - 1) \geq 3/4 \cdot x^s$ for $x \geq 3$ and $s \geq 2$, which means that

$$\Pr(\text{branch number} \geq s/2) \geq 1 - \frac{2}{p^{s/2-2}}$$

for $p \gg s \geq 6$. □

C On Using Two Different S-Boxes

To be optimized for HHE, we designed Pasta to have a small number of rounds (implying less noise consumption) and a small state size (implying fast homomorphic evaluation time). Therefore, we make use of a Feistel S-box of degree 2 and a cube S-box of degree 3. Using only Feistel S-boxes would result in a design with worse performance: A 3-round design using only Feistel S-boxes would require $t \approx 500$ plain/cipher words (based on the security analysis in Section 7), which results in significantly longer homomorphic evaluation times. A 4-round design would have the same multiplicative depth as Masta-4, leading to the same HE parameters and noise consumption as Masta-4. Therefore, this design would be faster then Masta due to the smaller size t ($t = 55$ as shown in Section 7) in one evaluation branch. However, it would not have a noise advantage. Pasta-3, on the other hand, has both a runtime and a noise advantage due to requiring fewer rounds by having a slightly smaller size t than Masta-4.

The diffusion of a 2-round cipher based only cube S-boxes would largely rely only on the single layer between the matrix multiplication. Thus the resulting diffusion is likely bad potentially allowing to separate the cipher [CDK+18]. Therefore, we chose to instantiate Pasta-3 by using the smallest depth which allows a 3-round cipher with approximately the same number of plain/cipher words t as Masta-4, which is using two Feistel S-boxes and one cube S-box.

D Pasta vs. Masta with matching Security Margins

Since Masta has a different security margin than Pasta, we want to argue, that the runtime/noise advantage of Pasta is due to its construction and not due to different margins. A Pasta-3 instance (dubbed Pasta-3-sm) with the same margin as Masta-4 would require a state size of 140 plain/cipher words, a Pasta-4 instance matching the margin of Masta-5 would requre 30 words. Therefore, our Pasta-4 instance already has a higher margin then Masta-5, which is why we focus on Pasta-3-sm in this section. We ran our benchmarks also for Pasta-3-sm with 140 plain/cipher words and report the runtime in Table 21 and Table 22. Our benchmarks confirm, that for both libraries (SEAL and HElib) Pasta-3-sm has the same noise consumption as Pasta-3 which is why we refrain from reporting the noise consumption in extra tables. Table 21 and Table 22 confirm, that Pasta-3-sm still outperforms Masta-4 in every metric further proofing the advantage of the Pasta design strategy.

Table 21: Additional benchmarks for the SEAL library.

Cipher	1 Block			Bigger HHE use case			
	N	Enc. Key s	Decomp. s	N	Enc. Key s	Decomp. s	Use Case s
$p = 65537$ (17 bit):							
Pasta-3	16384	0.016	9.22	32768	0.056	86.2	43.9
Pasta-3-sm	16384	**0.016**	**9.79**	32768	**0.056**	**91.2**	**44.5**
Masta-4	16384	0.016	11.6	32768	0.058	108.7	43.9
$p = 8088322049$ (33 bit):							
Pasta-3	32768	0.057	43.1	32768	0.055	86.3	43.9
Pasta-3-sm	32768	**0.058**	**45.5**	32768	**0.056**	**90.9**	**44.1**
Masta-4	32768	0.058	54.4	65536	0.215	568.5	221.3
$p = 1096486890805657601$ (60 bit):							
Pasta-3	32768	0.055	58.3	65536	0.220	448.6	220.8
Pasta-3-sm	32768	**0.056**	**62.2**	65536	**0.224**	**470.0**	**222.5**
Masta-4	65536	0.220	284.3	65536	0.212	571.9	223.1

Table 22: Additional benchmarks for the HElib library.

Cipher	1 Block				Bigger HHE use case				
	m	λ' bit	Enc. Key s	Decomp. s	m	λ' bit	Enc. Key s	Decomp. s	Use Case s
$p = 65537$ (17 bit):									
Pasta-3	65536	184	0.052	24.7	65536	128	0.064	57.6	19.9
Pasta-3-sm	65536	184	**0.059**	**26.7**	65536	128	**0.064**	**61.4**	**20.1**
Masta-4	65536	163	0.062	33.1	131072	229	0.131	157.3	45.4
$p = 8088322049$ (33 bit):									
Pasta-3	65536	128	0.057	28.7	131072	162	0.177	187.7	60.5
Pasta-3-sm	65536	128	**0.056**	**31.7**	131072	162	**0.174**	**200.4**	**61.1**
Masta-4	131072	196	0.165	101.3	131072	144	0.166	256.2	69.5
$p = 1096486890805657601$ (60 bit):									
Pasta-3	131072	162	0.181	94.1	131072[a]	97	0.285	268.8	84.4
Pasta-3-sm	131072	162	**0.199**	**99.1**	131072[a]	97	**0.264**	**281.7**	**85.4**
Masta-4	131072	129	0.208	144.7	131072[a]	83	0.279	387.4	101.6

[a] Further increasing m for security resulted in infeasibly long runtimes.