Grant Agreement Number: 825225

Safe-DEED

www.safe-deed.eu

Low complexity primitives v2/2

Deliverable number	*D5.7*
Dissemination level	*Public*
Delivery data	*due 30.11.2020*
Status	*Final*
Authors	*Lukas Helminger*

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825225.

Changes Summary

Date	Author	Summary	Version
30.10.2020	Lukas Helminger	First Draft (for internal review)	0.1
16.11.2020	Ludovico Boratto	Internal Review	0.2
17.11.2020	Alexandros Bampoulidis	Internal Review	0.3
21.11.2020	Lukas Helminger	Final Version (incorporated reviews)	1.0

Executive Summary

This deliverable D5.7 - Low complexity primitives v2 - is an update of D5.2. It is together with D5.8, the final outcome of task T5.1. This task was concerned with developing and improving WP5's core technologies with respect to both practical and theoretical aspects.

The research efforts conducted within this deliverable scope led to one additional scientific article accepted at Asiacrypt 2020. In addition, parts of this deliverable will be extended to a scientific article in the near future.

The two main results have direct as well as indirect consequences for Safe-DEED's privacy-preserving technology. The work on hybrid homomorphic encryption schemes can reduce the bandwidth requirement in all scenarios where homomorphic encryption is applied. In particular, it can be used in WP4's Data Valuation Component to minimize the cloud service's communication complexity. Besides, the security analysis on MIMC, a privacy-friendly encryption scheme, broadens the knowledge in this direction.

Table of Contents

List of Figures

Abbreviations

AES	Advanced Encryption Standard
DEM	Data-Encapsulation Mechanism
HE	Homomorphic Encryption
HHE	Hybrid Homomorphic Encryption
IND-CPA	Indistinguishability under Chosen Plaintext Attack
KEM	Key-Encapsulation Mechanism
MPC	Multi-Party Computation
PET	Privacy-Enhancing Technology
PIR	Private Information Retrieval
PRG	Pseudorandom Generator
RLWE	Ring-Learning with Errors
ZKP	Zero-Knowledge Proof

1 Introduction

This deliverable is the second within Safe-DEED that focuses on the improvement of low complexity primitives for privacy-preserving protocols. The first version - D5.2 - reported on a new symmetric encryption scheme for multi-party computation (see D5.1) partly developed in Safe-DEED. We will give an update on the research in this direction. However, the main focus of this version is on minimizing the communication complexity for homomorphic encryption schemes, which is another promising technology for realizing privacy-preserving protocols. This focus is motivated by privacy-preserving data analytics (WP4 and WP5 collaboration), but naturally generalized to many use-cases outside of the project.

Cloud services give individuals as well as businesses easy and cheap access to powerful tools. The demand for cloud services dramatically increased during the pandemic, especially when looking at teleconferencing (such as Zoom, GoToMeeting, and WebEx) and tools for enabling collaborative working (such as Google Apps, and Dropbox). Within Safe-DEED, we also develop a tool that can be seen as a cloud service.

1.1 Data Valuation Component

The data valuation component (DVC) is a convenient way to assess a company's data value. It consists of two subcomponents. First, a company has to answer several questions about their data. In a second step, the company has to upload (a sample of) the actual data. In the end, the DVC returns a score corresponding to the data's value. Clearly, many companies are looking for a service that can tell them how much their data are worth (for details see in Safe-DEED's deliverable D4.2 "Baseline prototypes for data valuation").

However, there is a general rise of concern when using cloud services over the loss of individual privacy and business values. Therefore, resulting in skepticism, which is a barrier to the adoption of cloud services. This issue also affects our DVC.

1.2 Homomorphic Encryption

In order to assuage these privacy concerns, we apply state-of-the-art privacy-preserving cryptographic primitives in Safe-DEED. More concretely, the companies' data, including the questionnaire's answers, can be protected by homomorphic encryption (HE). Homomorphic encryption enables for the first time to compute on encrypted data without decrypting illustrated in Figure 1. For technical details, please see Safe-DEED's deliverable D5.1 "Requirements for secure computation on large datasets with multiple owners". Theoretically, homomorphic encryption solves our issue because the DVC only sees encrypted data, but can still provide the data's valuation. Unfortunately, homomorphic encryption has some performance issues.

All known HE encryption schemes suffer from a very large expansion factor (the ratio between the ciphertext and plaintext length). Usually, HE schemes have at least a thousand-fold expansion, i.e., for sending only one HE encrypted image, the upload is around 1GB in size. Therefore, the transmission of the ciphertext between client and server is often prohibitive in terms of bandwidth.

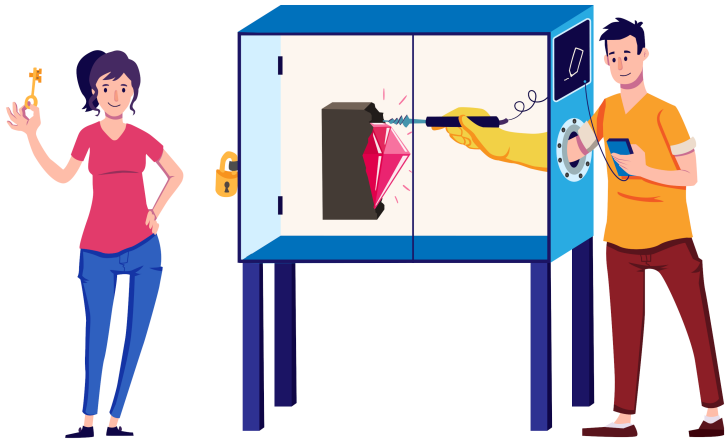

Figure 1: Homomorphic Encryption

Alice locks her diamond (her data) into a box. She ”encrypts” it. Charlie can work on the diamond (analyze the data) through the opening. He cannot steal (decrypt) it because only Alice has the key to unlock (decrypt) the box.

1.3 Hybrid Homomorphic Encryption

In order to minimize the communication complexity of HE schemes, Naehrig et al. [35] introduced the paradigm of hybrid homomorphic encryption (HHE). The idea is to use a (classical) symmetric encryption scheme (expansion factor nearly optimal) in combination with an HE scheme depicted in Figure 2. In particular, the data will be encrypted by a symmetric encryption scheme, and an HE scheme will encrypt only the symmetric key. Then, in addition to the application-specific computation, the server has to homomorphically evaluate the decryption of the data (with the HE encrypted symmetric key). This results in a homomorphic encryption of the original data. HHE has an expansion factor that is close to optimal, and so achieves its goal.

This approach's downside is that the computation on the server-side becomes more costly than without a symmetric encryption scheme. Namely, before any application-specific computation can be done, the server has to decrypt the data homomorphically. Hence, the cost of a homomorphic evaluation of a symmetric cipher is crucial for an HHE scheme's success. Several symmetric ciphers have been proposed so far, reaching from optimized Advanced Encryption Standard (AES) variants, through lightweight block ciphers to lattice-based encryption schemes. All of them provide implementations and corresponding benchmarks.

1.4 Benchmark Platform

However, as we will show in Section 3.4, none of these benchmarks are comparable, so it is impossible to tell which symmetric cipher is the best one. The lack of comparability is owed to various factors. First, the cipher designs were evaluated together with different HE libraries. Even if they used the same HE library, they often instantiated it with different security param-

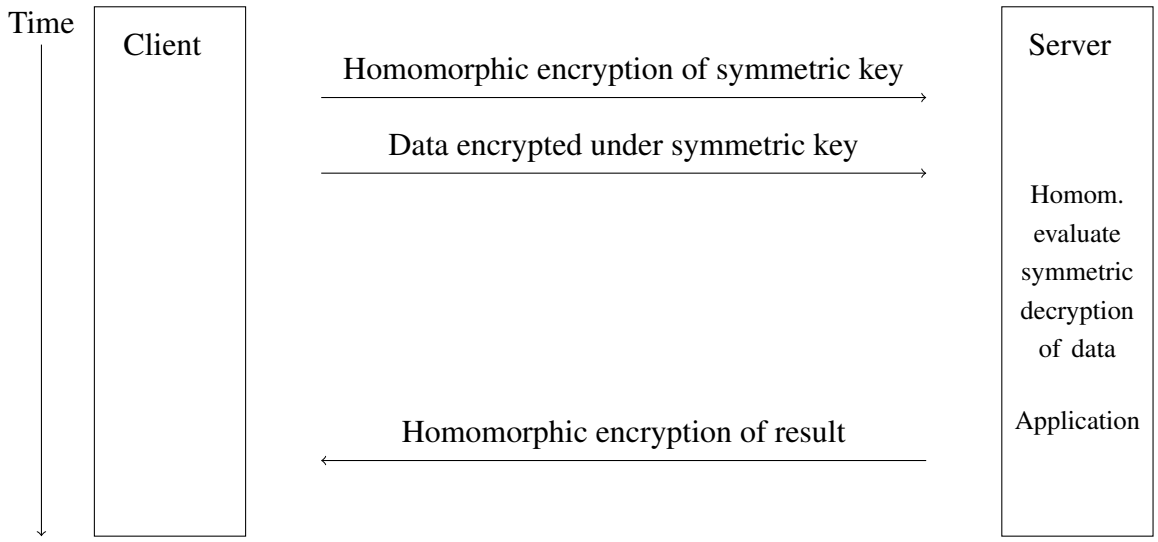

Figure 2: Hybrid Homomorphic Encryption

eters or relied on special packing methods. In addition, every benchmark was concluded on different infrastructure. Therefore, we do not have a clear understanding of the best symmetric candidate for HHE.

In this deliverable, we present the first benchmark platform for symmetric cipher candidates. We emphasize that beyond the task of comparing different ciphers, we revisit the HHE paradigm in order to come up with a unified formal definition.

Target Audience. This deliverable is targeted primarily towards researchers working in the realm of privacy-preserving data analytics. It is of particular interest to researchers working with large data sets. Further, this deliverable's outcome is relevant for software engineers because it gives the first comprehensive guide for choosing the appropriate symmetric cipher for homomorphic encryption.

1.5 Road-map

In Section 2, we give a high-level summary of a research article (co-written within Safe-DEED) in the direction of low complexity primitives security analysis. We defer to Appendix A for the article's abstract. This deliverable's main objective can be found in Section 3. We review the literature of HHE, both regarding formal definitions (Section 3.2) as well as practical solutions (Section 3.4). In Section 3.3 and Section 3.5, we give our own definition of HHE respectively present a new benchmark platform including results for HHE.

2 Research on Low Complexity Symmetric Key Primitives

To improve the performance of Privacy Enhancing Technologies (PETs) like Homomorphic Encryption, Multi-Party Computation (MPC), and Zero-Knowledge Proofs (ZKPs), one can often apply (symmetric) ciphers. For example, this deliverable extensively describes how ciphers re-

duce communication complexity for homomorphic encryption schemes [32, 9, 17, 3]. Similar effects can be seen for MPC [3, 26, 2, 27, 1] and ZKPs [2, 4, 25].

In the last years, there was a considerable effort to design ciphers dedicated to the above PETs. This effort lead to significantly better performance for these PETs. However, due to their recent nature, the amount of security analysis done so far cannot be compared to AES. Hence, it is of great importance to further study the security of these dedicated ciphers.

Safe-DEED was involved in a research article [20] showing new attacks on cipher designs often used for PETs. In particular, the article focused on the cipher MiMC [2, 27]. MiMC is often used as a baseline for more recent algorithms exploring this design space (e.g., see D5.2). We have included the abstract of the article in Appendix A.

A similar design, GMiMC [1] (co-developed in Safe-DEED), is not directly affected by the attack. Nevertheless, the observation of the research article enhances our understanding of GMiMC as well. We will benefit from this new knowledge in two ways. First, it contributes to the confidence in the existing GMiMC design. In addition, it will be incorporated into a new version of GMiMC.

3 Hybrid Homomorphic Encryption: Guideline

This section is a summary of our research into the topic of hybrid homomorphic encryption done so far. Based on this material and further analysis, we plan to publish a research article online based on this material. In addition, we are aiming to submit this article to a conference or a journal.

3.1 Motivation and Contribution

Most of the literature about the practicality of HE schemes focuses on the computational overhead. However, the first practical issue is already before the actual computation. The question is, how do we send a homomorphic ciphertext from the client to the server in an efficient way because all known HE encryption schemes suffer from a significant expansion factor. The expansion factor is the ratio between the ciphertext and plaintext length. Consequently, we want it to be as close to one as possible. Usually, HE schemes have around a thousand-fold expansion. In other words, if one wants to send 1MB in raw data, we expect the homomorphic ciphertext's size to be around 1GB in size. Therefore, the transmission of the ciphertext between client and server is often prohibitive in terms of bandwidth.

Our goal is to apply our expertise for low complexity primitives for MPC protocols to HE protocols. As a first step, we review existing literature and formalize the definition of hybrid homomorphic encryption. Then, we do a careful analysis of all the benchmarks done so far. The analysis' conclusion is that the benchmarks' setups differ widely and cannot be compared. Therefore, we designed and implemented the first benchmark platform for hybrid homomorphic encryption and already tested several candidates. This resulted in new insights about the performance of the different low complexity primitives. In the future, we plan to test even more candidates and open our platform to the public so that everyone can benchmark their low complexity primitives.

3.2 Review: Formal Definitions

3.2.1 Preliminaries and Notation

The computational security parameter for all encryption schemes will be denoted by κ. Further, if we write SYM for a symmetric encryption scheme, we implicitly mean $\mathsf{SYM} = (\mathsf{SYM.KGen}, \mathsf{SYM.Enc}, \mathsf{SYM.Dec})$. In analogy, we write short HE for a homomorphic encryption scheme meaning $\mathsf{HE} = (\mathsf{HE.KGen}, \mathsf{HE.Enc}, \mathsf{HE.Dec}, \mathsf{HE.Eval})$.

We will start - in Section 3.2.2 - by reviewing the previous work done in the light of defining HHE. Our approach is to describe the literature that formally treated HHE chronically. Note none of the papers focused on defining HHE. Instead, most of the time, it was a by-product, either of some bigger applications or benchmarks concerning symmetric ciphers. Therefore, we also explain the main contribution - in our opinion - of the following papers. We do this in order to help the reader to understand the specific approach of each paper better. Afterward, in Section 3.2.3, we work out the differences as well as the similarities in the several approaches to define HHE. This discussion will naturally lead us to a new formal definition. This new definition - in Section 3.3 - aims at being more general than the ones so far.

3.2.2 Literature

The first time that the idea of HHE was mentioned is in the work of Lauter et al. [35] when talking about optimizing the communication with the cloud. The main idea of this paper is to discuss the practicals of homomorphic encryption. Therefore, they look at real-world applications from different sectors and show a proof-of-concept implementation of an HE scheme based on Ring-Learning with Errors (RLWE) [30]. Since RLWE construction suffers from very large ciphertexts, Lauter et al. proposed HHE as mitigation. Their main observation is that all the steps for the decryption of a symmetric cipher can be performed on HE-encrypted entries. More concretely, in addition to the message encrypted with AES, the client sends the HE encryption of this secret AES key to the cloud. The cloud then homomorphically decrypts the AES ciphertext. Thereby it gets a homomorphic ciphertext of the original message. Their description covered all essentials of HHE, but the authors do not make an attempt to formalize or generalize their idea.

In 2014, Brakerski and Vaikuntanathan [8] generalized a construction for efficient Private Information Retrieval (PIR) [15] used in an earlier version of the paper [7]. This can be seen as the first attempt to formalize what they dubbed HHE. We have depicted Brakerski and Vaikuntanathan construction in Scheme 1. The main idea follows Lauter et al.'s practical description.

We now want to analyze some of their algorithms in a more detailed way. First, it has a flaw from a technical viewpoint because the encryption algorithm uses the secret symmetric key sk_{sym}. However, the key is not an input parameter of the algorithm and, therefore, can not be used. Adding the secret symmetric key as a parameter to the encryption algorithm would turn the construction into a private-key HE scheme. Secondly, they mention that their decryption algorithm can only handle evaluated ciphertext. Therefore, before we do the decryption, we have to check whether the ciphertext is still symmetric. If this is the case, they suggest to evaluate the ciphertext on the identity function first and then to decrypt the message. Third, note that the decryption circuit is built into the evaluation algorithm. Therefore it can only be called on symmetric ciphertexts. This is effectively limiting the number of evaluations to one.

On the theoretical side, Brakerski and Vaikuntanathan were already able to prove two natural and important statements. At first, they showed that the cryptographic construction in Scheme 1 is indeed again an HE scheme. In particular, if both the underlying private-key encryption scheme as well as the HE scheme is semantically secure, then they proved that their new HE scheme is also semantically secure. In other words, it inherits the property. To sum up, the first formal description of HHE is already quite evolved but has some technical issues.

Let HE be an homomorphic encryption scheme and SYM be a private-key encryption scheme with message space $\mathcal{M}$, and $m \in \mathcal{M}$. Then the hybrid encryption scheme HHE is defined as follows:

$\mathsf{HHE.KGen}(1^\kappa)$

$\mathsf{sk}_{sym} \leftarrow_\$ \mathsf{SYM.KGen}(1^\kappa)$
$(\mathsf{pk}, \mathsf{evk}, \mathsf{sk}) \leftarrow_\$ \mathsf{HE.KGen}(1^\kappa)$
$c_{sym} \leftarrow_\$ \mathsf{HE.Enc}_{\mathsf{pk}}(\mathsf{sk}_{sym})$
return $(\mathsf{pk}, (\mathsf{evk}, c_{sym}), \mathsf{sk})$

$\mathsf{HHE.Enc}(\mathsf{pk}, m)$

$c \leftarrow_\$ \mathsf{SYM.Enc}_{\mathsf{sk}_{sym}}(m)$
return c

$\mathsf{HHE.Eval}(f, \mathsf{evk}, (c_1, \ldots, c_t))$

$c^* \leftarrow_\$ \mathsf{HE.Eval}_{\mathsf{evk}}(\mathsf{SYM.Dec}_{c_{sym}}(c_1), \ldots, \mathsf{SYM.Dec}_{c_{sym}}(c_t))$
return c^*

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

if c is symmetric **then**
  $c \leftarrow_\$ \mathsf{HE.Eval}(id, \mathsf{evk}, c)$
fi
$m^* \leftarrow_\$ \mathsf{HE.Dec}_{\mathsf{sk}}(c)$
return m^*

Scheme 1: Hybrid Homomorphic Encryption [8]

Shortly thereafter, Gentry et al. came up with their own HHE construction. They refer to it at some point as homomorphic hybrid encryption and apply it to non-interactive zero-knowledge proofs. Further, they propose it as a way to minimize communication in general secure computation. Looking at their construction - Scheme 2 - we can immediately notice the use of a pseudorandom generator (PRG). Since a PRG is an idealization of a stream cipher, Scheme 2 is applicable only for a subset of all symmetric ciphers. A further limitation is that the decryption circuit is built into the evaluation algorithm. We have already seen the implications of this. Gentry et al. [8] showed a similar result about propagating the semantic security from the underlying HE scheme to the hybrid construction.

The work from Canteaut et al. [9] was probably the first one solely dedicated to the problem of transmitting a homomorphic ciphertext to an evaluator. The contribution to the formal understanding of HHE (compressed encryption in their words) is a generic design for efficient compression. So their approach - illustrated in Scheme 3 - is rather practical. As one can, for example, see due to the fact that within their definition, they specifically work with additive stream ciphers. In contrast to the last two approaches, the homomorphic decryption of symmetric ciphertexts is done in a separate algorithm - not built into the evaluation. This has the huge benefit of being able to use the evaluate algorithm as many times as we want. The issue was solved by implicitly assuming that the HE scheme is able to add plain constants without

Let HE be a fully homomorphic bit-encryption scheme, G a pseudorandom generator, and $m \in \{0,1\}^*$. Then they construct a fully homomorphic hybrid encryption scheme in the following way:

$\mathsf{HHE.KGen}(1^\kappa)$

$(\mathsf{sk},\mathsf{pk}) \leftarrow_\$ \mathsf{HE.KGen}(1^\kappa)$
return $(\mathsf{sk},\mathsf{pk})$

$\mathsf{HHE.Enc}(\mathsf{pk},m)$

$s \leftarrow_\$ \{0,1\}^\kappa$
$u = m \oplus G(s,|m|)$
$\bar{s} \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}}(s)$
return $c = (\bar{s},u)$

$\mathsf{HHE.Dec}(\mathsf{sk},v)$

return $\mathsf{HE.Dec}(v)$

$\mathsf{HHE.Eval}(C,\mathsf{pk},c_1,\ldots,c_n)$

Parse $c_i = (\bar{s}_i,u_i)$
$C_u \leftarrow f(1^\kappa,C,u_1,\ldots,u_n)$
return $v \leftarrow \mathsf{HE.Eval}_{\mathsf{pk}}(C_u,\bar{s}_1,\ldots,\bar{s}_n)$

where $f(1^\kappa,C,u_1,\ldots,u_n)$ returns a circuit
C_u s.t. for all $s_1,\ldots,s_n \in \{0,1\}^\kappa$
$C_u(s_1,\ldots,s_n) = C(u_1 \oplus G(s_1,|u_1|),\ldots,u_n \oplus G(s_n,|u_n|))$

Scheme 2: Hybrid Homomorphic Encryption [23]

encrypting them first. They did not specify any algorithms for the homomorphic evaluation or the decryption. Nevertheless, it can be safely assumed that it is fine to take both algorithms from the underlying HE scheme.

It is worthwhile to note that for performance reasons, they suggest distinguishing between three phases of HHE.

1. An **offline key-setup** phase which only depends on the public key and has to be performed only once.

2. An **offline decompression** phase which can be performed only based on some plaintext-independent material found in the compressed ciphertext.

3. An **online decompression** phase which aggregates the result of the offline phase with the plaintext-dependent part of the compressed ciphertext and recovers the decompressed ciphertext.

They shortly discuss that compressed homomorphic encryption is just hybrid encryption and relates to the generic Key-Encapsulation Mechanism/Data-Encapsulation Mechanism (KEM-DEM) framework. Therefore they claim Indistinguishability under Chosen Plaintext Attack (IND-CPA) of their generic construction as long as the HE is a semantic KEM and any general-purpose IND-CPA secure DEM. We see a similar argumentation in [31]. However, they mention that the notions of security of this framework - with respect to HE - have not been particularly investigated.

Let HE be a homomorphic encryption scheme with plaintext space $\{0,1\}^*$ and $m \in \{0,1\}^*$, an expansion function G and F a fixed-size parametrized function F with input size l_x, parameter size l_k with output size N.

$\mathsf{HHE.KGen}(1^\kappa)$

Transmit IV to both parties
$(\mathsf{sk},\mathsf{pk}) \leftarrow_\$ \mathsf{HE.KGen}(1^\kappa)$
return $(\mathsf{sk},\mathsf{pk})$

$\mathsf{HHE.Enc}(\mathsf{pk},m)$

$t \leftarrow \lceil |m|/N \rceil$
$(x_1,\ldots,x_t) \leftarrow G(IV,tl_x)$
$k \leftarrow_\$ \{0,1\}^{l_k}$
$z_i \leftarrow F_k(x_i)$ for $1 \le i \le t$
$kstream \leftarrow |m|$ leftmost bits of $z_1||\cdots||z_t$
return $c' \leftarrow (\mathsf{HE.Enc}_{\mathsf{pk}}(k), m \oplus kstream)$

$\mathsf{HHE.Eval}(C,\mathsf{pk},c_1,\ldots,c_n)$

return $\mathsf{HE.Eval}_{\mathsf{pk}}(C,c_1,\ldots,c_n)$

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

$t \leftarrow \lceil |m|/N \rceil$
$(x_1,\ldots,x_t) \leftarrow G(IV,tl_x)$
$\mathsf{HE.Enc}_{\mathsf{pk}}(z_i) \leftarrow \mathcal{C}_F(\mathsf{HE.Enc}_{\mathsf{pk}}(k),x_i)$ for $1 \le i \le t$
$\mathsf{HE.Enc}_{\mathsf{pk}}(kstream) \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}}(z_1),\ldots,\mathsf{HE.Enc}_{\mathsf{pk}}(z_t)$
return $c \leftarrow \mathcal{C}_\oplus(\mathsf{HE.Enc}_{\mathsf{pk}}(kstream), m \oplus kstream)$

Scheme 3: Compressed Homomorphic Encryption [9]

In terms of formalizing HHE [33] (and also the thesis [31]) are able to extend improve on the work of Canteaut et al. Most important, their framework to reduce the data complexity uses a generic symmetric cipher. By also dividing their construction Scheme 4 into different phases, they are able to show that the approach of Canteaut et al. is also applicable to generic symmetric ciphers. They even go a step further and include the evaluation and sending back - which is done after a ciphertext reduction - into their phases. Note that they state that the decompression algorithm gets way easier if the evaluation algorithm can handle symmetric ciphertexts.

For completeness, we depicted the formal definition of [22] in Scheme 5. The aim of this paper is to minimize the cost of the decryption algorithm's homomorphic evaluation of lattice-based symmetric encryption schemes.

3.2.3 Conclusion

In this section, we want to discuss the significant differences in the above approaches to defining HHE formally. Also, we want to give reasons which options are preferable to our viewpoint.

Maybe the most substantial difference is how homomorphic decryption of the symmetric ciphertexts is performed. We can see two methods to do homomorphic decryption

- built into the evaluation algorithm of the HHE [8, 23].
- a separate algorithm [33, 9, 22]

We prefer the latter approach for two reasons. First, it seems more general and also more intuitive to separate the evaluation algorithm and the algorithm for homomorphic decryption. The

Let SYM be a symmetric scheme and HE a homomorphic scheme (no specific requirements)

$\underline{\mathsf{HHE.KGen}(1^\kappa)}$

$(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{HE.KGen}(\kappa)$
$k \leftarrow \mathsf{SYM.KGen}(\kappa)$
$c^H \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}}(k)$
return $(\mathsf{sk},\mathsf{pk},k,c^H)$

$\underline{\mathsf{HHE.Enc}(\mathsf{pk},m)}$

$c^S \leftarrow \mathsf{SYM.Enc}_k(m)$
return c^S

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

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

$\underline{\mathsf{HHE.Eval}(C,\mathsf{pk},c_1,\ldots,c_n)}$

return $\mathsf{HE.Eval}(C,\mathsf{pk},c_1,\ldots,c_n)$

$\underline{\mathsf{HHE.Decomp}(c^H,c^S,\mathsf{pk})}$

return $\mathsf{HE.Eval}_{\mathsf{pk}}(C_{\mathsf{SYM.Dec}_{c^H}},\mathsf{HE.Enc}_{\mathsf{pk}}(c^S))$

$\underline{\mathsf{HHE.Red}(c^H)}$

return $\mathsf{HE}.Comp(c^H)$

Scheme 4: Hybrid Homomorphic Encryption Framework [33]

second reason is motivated by a real-world example. Think of a client sending homomorphically encrypted data to the server together with a function to evaluate. So far, both methods would work. What if, at some later point in time, the client asks the server to perform additional computations on the same data (without sending the encrypted data again). Then, this is only possible if the server is still able to perform computations on the encrypted data, which does not work if the decryption circuit is built into the evaluation algorithm. For the above reasons, we will opt for the second method and call the separated algorithm *Decompression* and write Decomp. A point that was not seriously discussed in any of the above work is the underlying homomorphic encryption scheme's plaintext space. We could find the following plaintext spaces:

- $\mathbb{Z}_2$ [9, 23, 8], and
- no restriction: [22, 33].

Obviously a general definition of HHE should not restrict the plaintext space of the underlying HE scheme. For example, this would already exclude on of the most popular HE libraries SEAL. At last, we want to pick up the idea of reducing the homomorphic ciphertexts before sending it back as it is mentioned in [35, 33]. While this is perfectly fine and can be an advantage - in terms of data complexity - we think this should not be a part of an HHE definition. Due to the fact that such a reduction only depends on the properties of the underlying HE scheme.

3.3 Formal Definition

This section aims to come up with a generally valid definition of HHE. The definition will be highly influenced by the literature and based on the conclusion drawn from the different approaches. Before defining HHE, we want to formally define its two components, private-key

Let HE be a fully homomorphic encryption scheme and SYM be a private-key encryption scheme and m a message. Then the hybrid encryption scheme HHE is defined as follows:

$\mathsf{HHE.KGen}(1^\kappa)$

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

$\mathsf{HHE.Enc}(\mathsf{pk}, m)$

$k \leftarrow \mathsf{SYM.KGen}(1^\kappa)$
$c_1 \leftarrow_\$ \mathsf{HE.Enc}_{\mathsf{pk}}(k)$
$c_2 \leftarrow_\$ \mathsf{SYM.Enc}_k(m)$
return (c_1, c_2)

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

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

$\mathsf{HHE.Eval}(f, \mathsf{evk}, c)$

$c^* \leftarrow \mathsf{HE.Eval}_{\mathsf{evk}}(f, c)$
return c^*

$\mathsf{HHE.Decomp}(\mathsf{evk}, c_1, c_2)$

$x \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}}(c_2)$
$c^* \leftarrow \mathsf{HE.Eval}_{\mathsf{evk}}(\mathsf{SYM.Dec}_{c_1}, x)$
return c^*

Scheme 5: Hybrid Homomorphic Encryption [22]

encryption and HE in Section 3.3.1. After the definition of HHE (Section 3.3.2), we will take a closer look at the relation between HHE and the KEM-DEM mechanism (Section 3.3.3). This discussion will enable us to understand HHE constructions' security, depending on the security of its two components (Section 3.3.4).

3.3.1 Preliminaries

As we have seen in Section 3.2, there were different ways of modeling the symmetric as well as the homomorphic part of HHE. In order to not exclude certain constructions, we are aiming to formulate the building blocks as general as possible. We start with what is commonly referred to as symmetric encryption.

Definition 3.1 (Private-Key Encryption, [28])**.** A private-key encryption scheme is a triple of probabilistic polynomial-time algorithms $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ such that:

1. The algorithm KGen takes as input 1^κ and outputs a key k.
2. The algorithm Enc takes as input a key k and a plaintext message $m \in \{0,1\}^*$, and outputs a ciphertext c.
3. The algorithm Dec takes as input a key k and a ciphertext c and outputs a message m or an error $\perp$.

After this well-established definition in cryptography, we are going to define HE. Due to the relative novelty of HE, there is no well-established book that defines HE. Therefore, we choose to follow the definition of [8]. The only adaption we make is to change the plaintext spaces from $\mathbb{Z}_2$ to arbitrary messages.

Definition 3.2 (Public-Key Homomorphic Encryption)**.** A homomorphic public key encryption scheme is a quadruple of probabilistic polynomial-time algorithms algorithms $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ that satisfy the following

1. The algorithm KGen takes as input the security parameter 1^κ and outputs a triple of keys $(\mathsf{pk}, \mathsf{sk}, \mathsf{evk})$, called the public key, the secret key and the evaluation key.
2. The algorithm Enc takes as input a public key pk and a message m from some underlying plaintext space. It outputs a ciphertext c.
3. The algorithm Dec takes as input a secret key sk and a ciphertext c, and outputs a message m or a symbol $\perp$ denoting failure.
4. The algorithm Eval takes as input a evaluation key evk, a l-ary function f, and a set of ciphertexts $c_1, \ldots, c_l$ and outputs a ciphertext c_f.

The first thing to consider after defining a new cryptographic primitive is correctness. In the case of homomorphic encryption, correctness is, in a way, incorporated into the ability to perform homomorphic computations.

Definition 3.3 (Correctness)**.** Let $m \in \mathcal{M}$ be an arbitrary message and let $f : \mathcal{M} \to \mathcal{M}$ be any function. A public-key homomorphic encryption scheme HE is correct if the following probability is negligible:

$$\Pr\left[\mathsf{HE.Dec}_{\mathsf{sk}}(\mathsf{HE.Eval}_{\mathsf{evk}}(f, c)) \neq f(m)\right], \quad \text{where } \mathsf{HE.Enc}_{\mathsf{pk}}(m) = c. \tag{1}$$

3.3.2 Hybrid Homomorphic Encryption

We are now ready to give our definition of HHE.

Definition 3.4 (Public-Key Hybrid Homomorphic Encryption (HHE))**.** Let HE be a public-key homomorphic encryption scheme and SYM a private-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 $\mathsf{HE.Dec}(\mathsf{sk}, c)$

$\mathsf{HHE.Eval}(f, \mathsf{evk}, c_1, \ldots, c_n)$

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

$\mathsf{HHE.Decomp}(c, \hat{k}, \mathsf{evk})$

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

return $\hat{c}$

Scheme 6: 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 (see Definition 3.3). Note that the relation between ciphertext and message in Equation (1) changes to $\hat{c} = \mathsf{HHE.Decomp}_{\mathsf{evk}}(\mathsf{HHE.Enc}_{\mathsf{pk}}(m))$.

Lemma 3.1. *Let* HE *be a correct public-key homomorphic encryption scheme and* SYM *a correct private-key encryption scheme. Then the resulting hybrid homomorphic encryption scheme* $\mathsf{HHE} = (\mathsf{HHE.KGen}, \mathsf{HHE.Enc}, \mathsf{HHE.Dec}, \mathsf{HHE.Eval})$ *is as well.*

Proof. 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 $\mathsf{HHE.Eval} = \mathsf{HE.Eval}$. 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 decomposition 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. □ □

3.3.3 Relation between HHE and KEM-DEM paradigm

The high-level idea of HHE is related to the KEM-DEM paradigm. The KEM-DEM paradigm traditionally aims to do as few public-key operations as possible because they are expensive. The public-key encryption scheme is exclusively used for agreeing on a common private key. In the homomorphic encryption setting, we have a similar objective. We want to reduce the high communication complexity, which is due to the huge ciphertext expansion inherent to HE schemes. Therefore, we saw that we only homomorphically encrypt the symmetric key instead of the actual possible data.

Due to the apparent relation between HHE and the KEM-DEM paradigm, it is valid to ask if all the KEM-DEM paradigm statements automatically apply to HHE. [9] only discussed this aspect informally, and [31] 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 a rigorous analysis of the connection between HHE and KEM-DEM and possible implications for HHE schemes' security. We start by recalling the definition of the KEM-DEM paradigm.

Definition 3.5 (KEM, [28])**.** A key-encapsulation mechanism (KEM) is a tuple of probabilistic polynomial-time algorithms algorithm $(\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$ such that:

1. Algorithm KGen takes as input the security parameter 1^κ and outputs the key public-/private-key pair $(\mathsf{pk}, \mathsf{sk})$.
2. Algorithm Encaps takes as input a public key pk and the security parameter 1^κ. It outputs a ciphertext c and a key $k \in \{0,1\}^{l(\kappa)}$, where $l(\kappa)$ is the key length.
3. Algorithm Decaps takes as input a private key sk and a ciphertext c, and outputs a key k or a special symbol $\perp$ denoting failure.

It is required that with all but negligible probability over $(\mathsf{sk}, \mathsf{pk})$ output by $\mathsf{KGen}(1^\kappa)$, if $\mathsf{Encaps}_{\mathsf{pk}}(1^\kappa)$ outputs (c,k), then $\mathsf{Decaps}_{sk}(c)$, then $\mathsf{Decaps}_{\mathsf{sk}}(c)$ outputs k.

It is easy to see that any public-key encryption scheme gives a KEM by choosing a random key k and encrypting it. In particular, any public-key homomorphic encryption scheme gives a KEM. A KEM is usually used together with a data encapsulation mechanism. Together they form the KEM/DEM paradigm depicted in Scheme 7.

Let $\Pi = (\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a KEM with key length κ, and let $\Pi' = (\mathsf{KGen}', \mathsf{Enc}', \mathsf{Dec}')$ be a private-key encryption scheme. Construct a public-key encryption scheme $\Pi^{\text{hy}} = (\mathsf{KGen}^{\text{hy}}, \mathsf{Enc}^{\text{hy}}, \mathsf{Dec}^{\text{hy}})$ as follows:

$\mathsf{KGen}^{\text{hy}}(1^\kappa)$	$\mathsf{Enc}^{\text{hy}}(\mathsf{pk}, m)$	$\mathsf{Dec}^{\text{hy}}(\mathsf{sk}, (c,c'))$
1 : **return** $(\mathsf{pk},\mathsf{sk}) \leftarrow_\$ \mathsf{KGen}(1^\kappa)$	$(c,k) \leftarrow_\$ \mathsf{Encaps}_{\mathsf{pk}}(1^\kappa)$ $c' \leftarrow_\$ \mathsf{Enc}'_k(m)$ **return** (c,c')	$(k) \leftarrow_\$ \mathsf{Decaps}_{\mathsf{sk}}(c)$ $m \leftarrow_\$ \mathsf{Dec}'_k(c')$ **return** m

Scheme 7: KEM/DEM paradigm, [28]

A naive first attempt to get a formal relation between HHE and KEM-DEM would be to instantiate the KEM-DEM paradigm with a public-key HE scheme. This is perfectly fine because, as we saw, every public-encryption scheme is sufficient for a KEM. We have worked out the details of such a construction in Scheme 8.

$\mathsf{KGen}^{\text{hy}}(1^\kappa)$	$\mathsf{Enc}^{\text{hy}}(\mathsf{pk}, m)$	$\mathsf{Dec}^{\text{hy}}(\mathsf{sk}, (c,c'))$
1 : $(\mathsf{pk},\mathsf{evk},\mathsf{sk}) \leftarrow_\$ \mathsf{HE.KGen}(1^\kappa)$ 2 : **return** $(\mathsf{pk},\mathsf{evk},\mathsf{sk})$	$k \leftarrow_\$ \{0,1\}^{l(\kappa)}$ $c \leftarrow_\$ \mathsf{HE.Enc}_{\mathsf{pk}}(k)$ $c' \leftarrow_\$ \mathsf{Enc}'_k(m)$ **return** (c,c')	$k \leftarrow_\$ \mathsf{HE.Dec}_{\mathsf{sk}}(c)$ $m \leftarrow_\$ \mathsf{Dec}'_k(c')$ **return** m

Scheme 8: Homomorphic Encryption KEM-DEM mechanism

If we compare this construction to Definition 3.4, we immediately see that the encryption algorithm is identical. In contrast, the decryption algorithm is neither identical to the decryption

Table 1: Metrics

	Metric	[24]	[11]	[34]	[16]	[19]	[29]	[3]	[18]	[22]	[33]	[9]	[17]	[32]	[10]
Performance	Latency	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Throughput	✓	✓	(✓)	✓	✓	✓	✓	✓	✓	✓	✓	(✓)	(✓)	✓
	Noise						✓							✓	✓
	Memory	(✓)						✓							
	Amortization Complexity														✓
Cipher char.	AND Depth					✓		✓		✓	✓		✓	✓	
	#AND-Gates											✓	✓		
	#minAND-Gates												✓		
	#XOR-Gates											✓			
	Algebraic Degree					✓									
	Key Generation	✓			✓		✓								
	Public Key Size				✓	✓			✓						

algorithm nor the decompression algorithm of the HHE scheme. Worse still, 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 strongly argue that HHE is not just an application of the KEM-DEM mechanism.

3.3.4 Security

Does this mean that we have to do the security analysis of HHE from scratch? We want to remind us why we are interested in finding a connection to the KEM-DEM paradigm. In the end, we would like to apply the statements about security that already exist plentiful for the KEM-DEM paradigm. Therefore, we now take a closer look at semantic security. First, recall, a public-key HE scheme is semantically secure if it is secure as a public-key encryption scheme [8]. Secondly, the proofs of the KEM-DEM construction's semantic security only depend on the encryption and encapsulation algorithm. We already saw that in those two algorithms HHE and KEM-DEM are identical. 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. A proof can be found in [28].

Theorem 3.2. *Let* HE *be a IND-CPA-secure public-key homomorphic encryption scheme and* SYM *a IND-CPA-secure private-key encryption scheme. Then* HHE *is a IND-CPA-secure public-key HE scheme.*

3.4 Review: Benchmarks

3.4.1 Metrics

The general meaning of most of the metrics in Table 1 is clear to a researcher in cryptography. Nevertheless, we want to specify what we understand under the given notions in the context of hybrid homomorphic encryption. For this reason, we have grouped them into the following categories: Performance of the algorithm *Decomp*, characteristics of the ciphers, performance of the hybrid homomorphic encryption algorithm, and miscellaneous

The performance of the homomorphic decryption of a symmetric cipher (*Decomp* in Scheme 6) was so far evaluated in five different metrics. Every paper that we reviewed measured the latency. Latency corresponds to the runtime of the algorithm. In other words, it is the total time

that it takes to perform the decryption circuit of a given cipher homomorphically. Latency alone can be misleading, due to the difference in the number of output bits of each cipher and the chosen packing variant (see Section 3.4.4). Therefore, it is also of interest to look at the throughput of a cipher. The throughput is defined as the amount of information per time unit that can be achieved. Lately, [10] introduced a new metric by evaluating how many ciphertexts are needed to achieve the optimal throughput. Only two papers so far looked explicit at the memory consumption of *Decomp*. Sometimes it was given implicitly by reporting the specifications of the machine, where the benchmarks were done. A further metric that was studied was the amount of noise the homomorphic decryption produces. Put differently we are interested in how much noise budget is still left after *Decomp*.

The second big category of metrics is the characteristics of symmetric ciphers. We want to highlight that in contrast to the above metrics - which have to be obtained by doing benchmarks - the metrics in this category can be exactly calculated. The most studied property of the ciphers with respect to homomorphic encryption is the multiplicative depth (often called the AND-gates depth). For better comparability between different cipher designs, [17] also calculated the minimal number of AND gates that are needed to produce at least one output bit. In addition, two papers looked at the total number of AND-gates and one of them also at the total number of XOR-gates. There was one paper that looked into the algebraic degree.

Performance of the encryption algorithm - in the setting of hybrid homomorphic encryption - was only measured by three papers. Even these papers restricted their benchmarks to the generation of the homomorphic encryption of the key for the symmetric cipher. Another unrelated metric that can be mostly found by earlier papers is the size of the public key of the underlying homomorphic encryption scheme.

3.4.2 Ciphers

A wide range of symmetric ciphers has been proposed to reduce ciphertext expansion in homomorphic encryption. In the beginning, most papers [24, 11, 34, 16, 18] looked at AES. They often choose a dedicated representation of the AES circuit that is best suited to be evaluated homomorphically. In the next phase, researchers looked at how well two lightweight ciphers - PRINCE [19] and Simon [29] perform in homomorphic evaluations. Shortly thereafter, we could see the first ciphers dedicated to privacy-preserving computation in general or even focused on homomorphic encryption. These new ciphers can be roughly grouped into two categories. On the one hand, we have ciphers where the security is related to some lattice problem (R,M)-LPN-C [22], (R)-LWR-SYM [22], and recently LWE [10]. On the other hand, we have more traditional symmetric designs as LowMC [3], FLIP [33], Trivium [9], Kreyvium [9], Rasta [17], and FiLIP[32].

3.4.3 Homomorphic Encryption Libraries

Not only have we seen many different designs of symmetric ciphers, but they were also evaluated with several different homomorphic encryption libraries. Most homomorphic encryption libraries were not used in more than two papers as BGV [24], DGHV [11, 16], LTV [19, 18], FV [29, 9], YASHE [29], GSW [33]. This is mainly due to the fact that until today the development of homomorphic encryption libraries is a highly active research area. Only the HELib was used frequently by different papers and, therefore, also different cipher designs

Table 2: Packing Techniques

	[24]	[11]	[34]	[16]	[19]	[29]	[3]	[18]	[22]	[33]	[9]	[17]	[32]	[10]
Packed	✓		✓		?			?	✓	?	✓		?	?
State-wise		✓		✓		✓								
Byte-Sliced	✓	✓	✓											
Bit-Sliced	✓						✓		✓			✓		

[34, 3, 22, 33, 9, 17, 32]. It is noteworthy that there is one rather recent proposal that claims to be independent of the specific homomorphic library [10].

3.4.4 Packing

Packing is another factor that has a considerable impact on the benchmarks. Especially the throughput can be significantly improved if one chooses the right packing technique for the cipher homomorphic library combination at hand. Comparing the packing technique used in the literature is a bit problematic because of the variety of the different homomorphic libraries and the symmetric ciphers' design. Besides, a few papers do not explicitly discuss which packing technique they are using in their implementation (depicted with a question mark). Nevertheless, in Table 1, we tried to group the packing techniques into four different categories.

Packed means that one homomorphic ciphertext is filled with as many as possible symmetric ciphertexts. In contrast, the remaining three packing categories split a single symmetric ciphertext into many homomorphic ciphertexts. The idea behind is to use vector processing in order to accelerate the homomorphic computations. This slicing can be done to various extend. State-wise stores each state of the giving symmetric cipher in an extra homomorphic ciphertext. Byte-Sliced, respectively Bit-Sliced, go even further. They store each byte respectively bit in a separate homomorphic ciphertext. All three slicing techniques have in common that they put parts of different ciphertexts adjacent to each other in a single homomorphic ciphertext. This aims at increasing the throughput.

3.4.5 Security Parameters

Comparing different ciphers only makes sense if we compare them for the same security level. Homomorphic encryption is a relatively new primitive, so are the methods to estimate the bit-security of homomorphic libraries with specific parameters. Even worse, different papers used different methods to claim the security of their homomorphic library in use. This is one of the reasons why we found ten security levels - ranging from 42-bit to 256-bit - in the reviewed literature. At least in the more recent literature, one can observe that the aim is to go for 80-bit security and provide figures for 128-bit security.

3.4.6 Discussion

To sum up, for the various reasons outlined above one can not determine the best cipher for hybrid homomorphic encryption so far. Our goal is to change this by making the first comparable benchmarks.

We decided to focus on two key metrics. Firstly, as every paper before, we will benchmark the latency, i.e., the runtime of the *Decomp* algorithm. Latency implicitly benchmarks the

throughput, which can be calculated by the latency and the specifics of a given cipher. In addition, we propose a new way to evaluate the cipher's impact on the noise parameter of HE schemes. We will do a matrix multiplication after the *Decomp* algorithm. The idea behind this is two-fold. On the one hand, this approach is more independent of the HE library than previous noise analysis. On the other hand, matrix multiplications are basics building blocks for more complex computations like neural networks.

3.5 Benchmarks

For this deliverable, we focus on benchmarking the ciphers LowMC [3], Kreyvium [9], Rasta [17], and FiLIP [32]. While Kreyvium, Rasta, and FiLIP are considered to be state-of-the-art in minimizing the AND-depth of a secure symmetric cipher, LowMC was the first cipher designed to reduce the number of AND-gates and provides a good baseline for the benchmarks.

3.5.1 Cipher Parameters and Modes of Operation

In this section, we discuss the benchmarked ciphers, and how we used them to encrypt data of arbitrary length. In general, we chose the instance of each cipher which has the smallest AND-depth for $\kappa = 128$ bit computational security. In Table 3 we summarize the parameters of the ciphers in their respective modes of operations.

LowMC LowMC is a very parameterizable blockcipher, designed to minimize the number of AND gates required for secure encryption. In this project, we benchmark LowMC($n = 256$, $k = 128$, $m = 63$, $r = 14$, $d = 128$), the LowMC instance proposed for HHE in the original publication [3]. To encrypt plaintext of arbitrary length, we use LowMC in the Counter (CTR) mode of operation.

Rasta Rasta is a family of stream ciphers, in which the permutation is different for each encrypted block. In this project, we benchmark Rasta($n = 525$, $r = 5$) [17]. 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 Agrasta is an aggressive instantiation of the Rasta stream cipher with no security margin. The lack of security margin allows to further reduce the AND-depth compared to more conservative parameter sets. In this project we benchmark Agrasta($n = 129$, $r = 4$) [17].

Kreyvium Kreyvium is a 128 bit security variant of the low-depth stream cipher Trivium. In Kreyvium, the AND-depth of the cipher grows with the number of required keystream bits, which is undesired in HHE. 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. In this project, we benchmark Kreyvium-12 [9].

FiLIP 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. However, to better compare it with the other ciphers, we define a block size of 64 bit and benchmark FiLIP in counter mode. In this project, we benchmark FiLIP-1216 [32].

Table 3: Parameters of the benchmarked ciphers in their respective modes of operations in bits.

Cipher	Blocksize	Keysize	AND-depth
LowMC ($n = 256$, $k = 128$, $m = 63$, $r = 14$, $d = 128$)	256	128	14
Rasta ($n = 525$, $r = 5$)	525	525	5
Agrasta ($n = 129$, $r = 4$)	129	129	4
Kreyvium-12	42	128	12
FiLIP-1216 ($n = 64$)	64	16384	3

3.5.2 FHE Libraries

In this project, we benchmark HHE for two different homomorphic encryption libraries with HE parameters which provide $\kappa = 128$ bit computational security. We discuss the two libraries in the following:

SEAL [36] SEAL is a state-of-the-art, actively developed, homomorphic encryption library maintained by Microsoft Research. It implements the BFV [5, 21] and CKKS [12] homomorphic encryption schemes. In this project, we implement and benchmark the ciphers for the BFV cryptosystem. In BFV, plaintexts can be elements in $\mathbb{Z}_t$, however to support evaluations of the benchmarked symmetric ciphers, we are limited to plaintexts in $\mathbb{Z}_2$. Like most modern HE schemes [5, 21, 12, 6], BFV is based on the RLWE [30] hardness assumption, in which random noise is added to ciphertexts for security. This noise grows for every homomorphic operation, negligible for homomorphic additions, but significantly for homomorphic multiplications. Once the noise becomes too large, the ciphertexts cannot be decrypted correctly anymore. Therefore, the most limiting metric in BFV is the number of consecutive multiplications, i.e. the AND-depth in $\mathbb{Z}_2$. Furthermore, multiplications are far more expensive in terms of computational effort then additions.

TFHE [14] TFHE implements the FHE scheme of the same name (TFHE [13]). TFHE only allows to encrypt boolean values (i.e., plaintexts are in $\mathbb{Z}_2$) and specializes on fast gate bootstrapping. This basically means, that the noise in the ciphertexts is reset after each homomorphically evaluated boolean gate. As a consequence, contrary to most other modern homomorphic encryption schemes, the AND-depth of a cipher is no relevant metric in TFHE. However, each homomorphically evaluated gate requires the same computational effort to evaluate, thus additions are not considered free as in the BFV cryptosystem. The most relevant metric for TFHE is, therefore, the total number of gates.

3.5.3 Benchmark Platform

We run all benchmarks on a Linux Server with a Intel Xeon E5-2669 v4 CPU (2.2 GHz, turbo-boost up to 3.6 GHz) and 512 GB RAM available. Each of the individual benchmarks has only access to one thread.

3.5.4 Benchmarked Applications

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, that the server wants to compute $\vec{r} = M \cdot \vec{v} + \vec{b}$, where $\vec{r}, \vec{v}, \vec{b} \in \mathbb{Z}_{2^{16}}^4$ and $M \in \mathbb{Z}_{2^{16}}^{4\times 4}$, i.e., a 4×4 matrix-vector multiplication of 16-bit integers. The matrix M, and the vector $\vec{b}$ is, thereby private data owned by the server, and $\vec{v}$ 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 result. Since all benchmarked ciphers operate over $\mathbb{Z}_2$, we have to implement binary circuits for addition and multiplication to perform the matrix multiplication over integers in $\mathbb{Z}_{2^{16}}$.

3.5.5 SEAL Benchmarks

In SEAL, the available noise budget, (i.e. how much further noise can be introduced until decryption will fail) depends on the degree N of the cyclotomic reduction polynomial. N is always a power of two and has a severe impact on the performance of the HE scheme. While a bigger N comes the with a bigger noise budget, it exponentially increases the runtime of homomorphic operations.

In Table 4 we present the benchmarks for the SEAL library, for homomorphically decrypting only one block, and for the complete HHE use case. For both benchmark we give timings for homomorphically encrypting the symmetric key and homomorphically decrypting the symmetric ciphertexts for the smallest N providing enough noise budget for correct evaluation. For the HHE use case we additionally give the runtime for the matrix multiplication. In Table 5 we additionally give the remaining noise budget after encrypting the symmetric key, homomorphically decrypting the symmetric ciphertexts, and performing the matrix multiplication.

Table 4: Benchmarks for the SEAL library.

Cipher	1 Block			HHE use case			
	N	Enc. Key	HE decrypt	N	Enc. Key	HE decrypt	Matmul
		s	s		s	s	s
LowMC	16384	2.1	507.1	32768	7.4	2 384.9	999.7
Rasta	8192	2.6	166.9	32768	31.3	2 485.6	1 019.8
Agrasta	8192	1.1	26.8	16384	3.8	136.3	384.9
Kreyvium	8192	1.2	172.6	32768	13.3	7 908.0	1 686.7
FiLIP	8192	141.1	2 533.0	16384	407.4	6 549.4	193.6

Table 5: Noise budget after each operation in the SEAL lbirary.

Cipher	1 Block			HHE use case			
	N	Enc. Key bit	HE decrypt bit	N	Enc. Key bit	HE decrypt bit	Matmul bit
LowMC	16384	379	69	32768	815	489	215
Rasta	8192	165	46	32768	815	686	411
Agrasta	8192	165	82	16384	379	293	23
Kreyvium	8192	165	0	32768	815	624	350
FiLIP	8192	164	122	16384	379	333	79

3.5.6 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. In Table 6 we present the benchmarks for the TFHE library for homomorphically decrypting only one block, and for the complete HHE use case. We give timings for homomorphically encrypting the symmetric key, homomorphically decrypting one block and for the complete HHE use case.

Table 6: Benchmarks for the TFHE library.

Cipher	Enc. Key	1 Block	HHE use case	
		HE decrypt	HE decrypt	Matmul
	s	s	s	s
LowMC	0.003	6 767.4	6 574.6	119.7
Rasta	0.015	6 705.3	7 399.2	227.3
Agrasta	0.005	836.9	825.3	198.0
Kreyvium	0.005	538.7	1 028.6	192.9
FiLIP	0.490	1 678.4	1 689.8	133.6

3.5.7 Throughput

Since the ciphers we benchmark have different blocksizes, we also want to list the throughput of the ciphers. In Table 7 we list the throughput of the ciphers for the different parameter sets from the previous two sections. The numbers were obtained by dividing the runtime of homomorphically decrypting one block by the blocksize.

3.5.8 Discussion

In this section, we discuss the benchmarks and some further considerations.

Ciphers for SEAL In SEAL, on of the most important metric is the AND-depth of the evaluated circuit. The bigger the depth, the bigger noise budget is required, exponentially increasing

Table 7: Throughput of the ciphers.

	SEAL				TFHE
	N	Throughput bit/s	N	Throughput bit/s	Throughput bit/s
LowMC	16384	0.505	32768	0.108	0.038
Rasta	8192	3.146	32768	0.138	0.078
Agrasta	8192	4.813	16384	0.953	0.154
Kreyvium	8192	0.243	32768	0.010	0.078
FiLIP	8192	0.025	16384	0.005	0.038

the runtime of homomorphic operations. As can be seen in Table 4, most ciphers can be evaluated with a smalle N (i.e., small noise buidget). However, if we additionally want to evaluate a use case after decrypting ciphertexts, N has to be much larger. In terms of noise, the best choice for HHE in SEAL is FiLIP, closely followed by Agrasta. However, due to its large key size and small throughput, the runtime to homomorphically decrypt FiLIP ciphertexts is too hight to be competitive, leaving Agrasta as the best choice for HHE in SEAL.

Ciphers for TFHE In TFHE, only the total gate count matters for the performance of homomorphic operations. Therefore, Kreyvium intuitively looks like the best choice for HHE in TFHE due to its low gate count. Indeed, Kreyvium has the best runtime for single block decryption. However, due to its small blocksize it has a very small throughput, and depending on the use case, Agrasta might be the better choice.

Binary Circuits In TFHE plaintexts are limited to boolean gates (i.e., plaintexts are in $\mathbb{Z}_2$), therefore, every use case which includes arithmetic over $\mathbb{Z}_t$ has to implement binary circuits. On the other hand, plaintexts in SEAL can be arbitrary elements in $\mathbb{Z}_t$. However, since there is no known transformation from HE over $\mathbb{Z}_2$ to HE over $\mathbb{Z}_t$ without knowing the secret decryption key, using the benchmarked ciphers in HHE limits the use cases in SEAL to also operate over $\mathbb{Z}_2$. Therefore, a symmetric cipher over $\mathbb{Z}_t$ would enable HHE use case with arithmetics over $\mathbb{Z}_t$ without the requirement of implementing binary circuits. This would drastically speed up HHE use cases over $\mathbb{Z}_t$.

Further Considerations One of the most importatnt uses of HHE is outsourcing computations from small embedded devices to more powerful cloud services. The embedded devices, thereby, only have limited computing resources, limiting their capabilities. In HHE, therefore, homomorphically encrypting the symmetric key and encrypting plaintexts (without HE) are required to be efficient operations as well. This aspects, in general, disqualifies FiLIP-1216 as a suitable candidate for those use cases, due to its huge key. In future work, we intend to also investigate the usefulness of FiLIP-1280, another FiLIP instance, with smaller keysize, but bigger AND-depth.

4 Conclusion

The knowledge presented in this deliverable is a big step forward in the area of symmetric-key primitives for privacy-preserving computations. On the one hand, we were able to get new insights into low complexity, symmetric ciphers' security. This will lead to more confidence in these new cipher category and potentially improved designs. On the other hand, we created the first benchmark framework for hybrid homomorphic encryption. Even the first round of experiments lead to a better understanding of the different cipher's performance. In the future, we will extend the experiments so that we can find the best cipher for hybrid homomorphic encryption. The goal is to submit the extended experiments to a scientific venue.

5 References

[1] Martin R. Albrecht, Lorenzo Grassi, Léo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, Arnab Roy, and Markus Schofnegger. Feistel structures for mpc, and more. In Kazue Sako, Steve A. Schneider, and Peter Y. A. Ryan, editors, *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part II*, volume 11736 of *Lecture Notes in Computer Science*, pages 151–171. Springer, 2019.

[2] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 191–219, 2016.

[3] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 430–454. Springer, 2015.

[4] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. *IACR Trans. Symmetric Cryptol.*, 2020(3):1–45, 2020.

[5] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012.

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

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

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

[9] 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 *Lecture Notes in Computer Science*, pages 313–333. Springer, 2016.

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

[11] 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 *Lecture Notes in Computer Science*, pages 315–335. Springer, 2013.

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

[13] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *ASIACRYPT (1)*, volume 10031 of *Lecture Notes in Computer Science*, pages 3–33, 2016.

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

[15] 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.

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

[17] 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 *Lecture Notes in Computer Science*, pages 662–692. Springer, 2018.

[18] Yarkin Doröz, Yin Hu, and Berk Sunar. Homomorphic AES evaluation using the modified LTV scheme. *Des. Codes Cryptogr.*, 80(2):333–358, 2016.

[19] Yarkin Doröz, Aria Shahverdi, Thomas Eisenbarth, and Berk Sunar. Toward practical homomorphic evaluation of block ciphers using prince. In *Financial Cryptography Workshops*, volume 8438 of *Lecture Notes in Computer Science*, pages 208–220. Springer, 2014.

[20] Maria Eichlseder, Lorenzo Grassi, Reinhard Lüftenegger, Morten Øygarden, Christian Rechberger, Markus Schofnegger, and Qingju Wang. An algebraic attack on ciphers with low-degree round functions: Application to full mimc. *IACR Cryptol. ePrint Arch.*, 2020:182, 2020.

[21] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[22] Pierre-Alain Fouque, Benjamin Hadjibeyli, and Paul Kirchner. Homomorphic evaluation of lattice-based symmetric encryption schemes. In *COCOON*, volume 9797 of *Lecture Notes in Computer Science*, pages 269–280. Springer, 2016.

[23] 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. Cryptology*, 28(4):820–843, 2015.

[24] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012.

[25] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. Cryptology ePrint Archive, Report 2019/458, 2019. `https://eprint.iacr.org/2019/458`.

[26] Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. On a generalization of substitution-permutation networks: The HADES design strategy. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 674–704. Springer, 2020.

[27] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. Mpc-friendly symmetric key primitives. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 430–443. ACM, 2016.

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

[29] Tancrède Lepoint and Michael Naehrig. A comparison of the homomorphic encryption schemes FV and YASHE. In *AFRICACRYPT*, volume 8469 of *Lecture Notes in Computer Science*, pages 318–335. Springer, 2014.

[30] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010.

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

[32] 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 *Lecture Notes in Computer Science*, pages 68–91. Springer, 2019.

[33] 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 *Lecture Notes in Computer Science*, pages 311–343. Springer, 2016.

[34] Silvia Mella and Ruggero Susella. On the homomorphic computation of symmetric cryptographic primitives. In *IMA Int. Conf*, volume 8308 of *Lecture Notes in Computer Science*, pages 28–44. Springer, 2013.

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

[36] Microsoft SEAL (release 3.5). `https://github.com/Microsoft/SEAL`, April 2020. Microsoft Research, Redmond, WA.

A MiMC Attack

An Algebraic Attack on Ciphers with Low-Degree Round Functions: Application to Full MiMC

(Full Version)

Maria Eichlseder[1], Lorenzo Grassi[1,2], Reinhard Lüftenegger[1], Morten Øygarden[3], Christian Rechberger[1], Markus Schofnegger[1], and Qingju Wang[4]

[1] IAIK, Graz University of Technology (Austria)
[2] Digital Security Group, Radboud University, Nijmegen (The Netherlands)
[3] Simula UiB (Norway)
[4] SnT, University of Luxembourg (Luxembourg)
firstname.lastname@iaik.tugraz.at
lgrassi@science.ru.nl
morten.oygarden@simula.no
qingju.wang@uni.lu

Abstract. Algebraically simple PRFs, ciphers, or cryptographic hash functions are becoming increasingly popular, for example due to their attractive properties for MPC and new proof systems (SNARKs, STARKs, among many others).

In this paper, we focus on the algebraically simple construction MiMC, which became an attractive cryptanalytic target due to its simplicity, but also due to its use as a baseline in a competition for more recent algorithms exploring this design space.

For the first time, we are able to describe key-recovery attacks on all full-round versions of MiMC over $\mathbb{F}_{2^n}$, requiring half the code book. In the chosen-ciphertext scenario, recovering the key from this data for the n-bit full version of MiMC takes the equivalent of less than $2^{n-\log_2(n)+1}$ calls to MiMC and negligible amounts of memory.

The attack procedure is a generalization of higher-order differential cryptanalysis, and it is based on two main ingredients. First, we present a higher-order distinguisher which exploits the fact that the algebraic degree of MiMC grows significantly slower than originally believed. Secondly, we describe an approach to turn this distinguisher into a key-recovery attack without guessing the full subkey. Finally, we show that approximately $\lceil \log_3(2 \cdot R) \rceil$ more rounds (where $R = \lceil n \cdot \log_3(2) \rceil$ is the current number of rounds of MiMC-n/n) can be necessary and sufficient to restore the security against the key-recovery attack presented here.

The attack has been practically verified on toy versions of MiMC. Note that our attack does not affect the security of MiMC over prime fields.

Keywords: Algebraic attack · MiMC · Higher-order differential