Symmetrical Disguise: Realizing Homomorphic Encryption Services from Symmetric Primitives (extended version)*

Alexandros Bakas, Eugene Frimpong, and Antonis Michalas

Tampere University, Tampere, Finland
{alexandros.bakas,eugene.frimpong,antonios.michalas}@tuni.fi

Abstract. Homomorphic Encryption (HE) is a modern cryptographic technique that allows direct computations on encrypted data. While relatively new to the mainstream debate, HE has been a solid topic in research for decades. However, and despite the technological advances of the past years, HE's inefficiencies render it impractical for deployment in realistic scenarios. Hence research in the field is still in its initial phase. To overcome certain challenges and bring HE closer to a realization phase, researchers recently introduced the promising concept of Hybrid Homomorphic Encryption (HHE) – a primitive that combines symmetric cryptography with HE. Using HHE, users perform local data encryptions using a symmetric encryption scheme and then outsource them to the cloud. Upon reception, the cloud can transform the symmetrically encrypted data to homomorphic ciphertexts without decrypting them. Such an approach can be seen as an opportunity to build new, privacy-respecting cloud services, as the most expensive operations of HE can be moved to the cloud.

In this work, we undertake the task of designing a secure cryptographic protocol based on HHE. In particular, we show how HHE can be used as the main building block of a protocol that allows an analyst to collect data from multiple sources and compute specific functions over them, in a privacy-preserving way. To the best of our knowledge, this is the first work that aims at demonstrating how HHE can be utilized in realistic scenarios, through the design of a secure protocol.

Keywords: Homomorphic Encryption · Hybrid Homomorphic Encryption · Multi-Client · Storage Protection

1 Introduction

Cloud computing has become an integral part of our lives. It has not only impacted our daily functions but also how businesses and organizations manage their data and customers. The wide use of cloud-services has, as expected, raised

* This work was partially funded by the `HARPOCRATES` project, Horizon Europe and the Technology Innovation Institute (TII), Abu Dhabi, United Arab Emirates, for the project `ARROWSMITH: Living (Securely) on the edge`.

a plethora of challenging security and privacy problems. One of the main security concerns related to cloud computing has to do with so-called internal attacks. This is, a corrupted cloud service provider (CSP) exploiting customer data for its own benefit, e.g. sharing customer data with third parties. To alleviate these concerns CSPs have introduced support for data encryption. However, the problem of creating real privacy-respecting cloud services is not as easy as applying encryption on the stored data. For this reason, the research community has started looking into solutions that are not based on traditional encryption and can successfully protect user data from internal attacks without jeopardizing the main benefits of cloud computing. One of the most common solutions is Structured Encryption (SE) [30], where data is encrypted locally with a key that is unknown to the CSP. Hence, the CSP, which does not have access to the encryption key cannot learn anything about the content of user data. Furthermore, whenever a user wishes to access her files, she can search directly over the encrypted data for specific keywords. While this approach solves part of the problem, (i.e. users do not have to download and decrypt the whole database), ciphertexts remain "useless" in the sense that one can not operate on them as if as they were plaintexts. With a view to addressing this issue, a number of approaches to make ciphertexts "more useful" and operate on encrypted data have been developed. The most promising solutions is Homomorphic Encryption (HE) [37] and Functional Encryption [28] – two modern encryption techniques that allow authorized entities (i.e. users, the cloud or third parties) to perform computations on the encrypted data without accessing their contents.

Homomorphic encryption. Often dubbed as *"the holy grail of cryptography"*. In an HE scheme a user first generates a public/private key pair $(\mathsf{pk}, \mathsf{sk})$ and an evaluation key evk[1]. Then, given two ciphertexts c_1, c_2 encrypting messages x_1 and x_2 respectively and the evaluation key evk, it is possible to compute $f(c_1, c_2)$, where f is a function associated either with addition or multiplication. Moreover, what is fascinating about HE, is that in computing $f(c_1, c_2)$ there are no leaks about the underlying plaintexts x_1 and x_2 while decrypting the result in only feasible by possessing the secret key sk. Naturally, this opens up tremendous possibilities as, for the first time ever, it becomes possible to not only outsource data, applications and services but also *computations* to the cloud, in a privacy-preserving manner. However, despite its advantages, HE is unfortunately characterized by its inefficiency. Homomorphically encrypting big loads of data requires powerful machines and is time-consuming. As a result, to this day HE is a topic of interest mainly among members of the academic community. To address these inefficiencies however, researchers recently turned their attention to Hybrid Homomorphic Encryption (HHE) [22].

Hybrid Homomorphic encryption. In an HHE scheme, a user encrypts data locally using a symmetric key K of a symmetric-key encryption scheme SKE. Subsequently, K is encrypted under HE's public key pk and is outsourced to the cloud

[1] Sometimes, in literature, the evaluation key is part of the public key.

along with the ciphertexts and the evaluation key evk. Upon reception, the CSP can transform the symmetrically encrypted data to homomorphic ciphertexts and hence operate on them. This promising approach significantly reduces computation costs on the client side by moving the most expensive computations on the cloud, where powerful machines are used traditionally for the processing and storage of the data. In this work, while we do not design a novel HHE scheme, we design a detailed protocol that aims at showing the applicability and functionality of HHE in real-world scenarios.

Contributions: While multiple different HHE schemes have been proposed over the past few years, to the best of our knowledge, none of these describe in detail how HHE can be used as the main building block of a secure protocol. We believe this is an important step forward that can bridge the gap between theoretical cryptographic concepts and security engineering and can pave the way for the implementation of a vast amount of privacy-respecting cloud services. The core contributions of this work can be summarized as follows:

C1. We design a protocol that utilizes the concept of HHE and allows multiple users to securely store and process their data in the cloud.

C2. We provide an efficient and novel way of using HE to securely store and analyze data stored in a remote location. More precisely, our scheme can run in any device that can run a typical symmetric encryption algorithm.

C3. We prove the security of our protocol in the presence of a malicious adversary modelled after the Dolev-Yao adversarial model [24].

C4. Our theoretical evaluation, is coupled with extensive experimental results that prove our protocol's efficiency and applicability.

2 Related Work

- ***Homomorphic Encryption*:** While HE has attracted a lot of attention in the recent years, it was first mentioned by Rivest et al. in 1978 [37]. However, the first HE constructions allowed only for one specific operation on encrypted data. The operation could either be addition, using the Paillier cryptosystem [36], or multiplication, under RSA [38]. It was not until 2009 and the work of Gentry that the first fully homomorphic encryption (FHE) scheme was developed [26]. This was a major breakthrough in the field of cryptography as, in theory, by using a FHE scheme one can perform any operation directly on encrypted data. While fascinating, this work was unfortunately characterized by its inefficiency. However, it produced a series of publications in the field [14,25,15,19,18]. These works addressed the impracticalities of Gentry's work and lead to novel and more efficient schemes.
- ***Hybrid Homomorphic Encryption*:** HHE was first introduced as a concept in [35], but the first formal definition was presented very recently in [23]. The first approaches for the design of HHE schemes, relied on existing and well-established symmetric ciphers, like AES [27,17,21]. However, AES was

not a good suitor for building HHE schemes, mainly due to its large multiplicative depth. Thus, research on the field of HHE took a new turn where the main focus has been shifted to the design of symmetric ciphers with different optimization criteria, depending on the use-case each work addresses [20,5,16,22,32,29,23]

- ***Different Approaches*:** Another emerging cryptographic primitive that can be used to outsource computations in a privacy-preserving is Functional Encryption (FE). In FE, each decryption key sk_f is connected to a function f. Unlike traditional public-key cryptography, the use of sk_f on a ciphertext $\mathsf{Enc}(x)$ does *not* recover x but a function $f(x)$. In this way, the actual value x remains private. A more recent work [28] has introduced the general and promising notion of multi-input FE (MIFE). Here, when ciphertexts $\mathsf{Enc}(x_1), \ldots, \mathsf{Enc}(x_n)$ are provided, sk_f can be used to recover $f(x_1, \ldots, x_n)$. To this day, most works in FE revolve around designing schemes for sums [13,8,11], inner products [1,3,2], as well as quadratic functions [40]. Unfortunately, except a handful works that rely on Multi-Party Computation (MPC) [11,12], most of FE schemes are highly centralized and require the existence of a fully trusted central authority.
- ***Provable Secure Protocols for Cloud Security using HHE*:** Designing provable secure protocols that utilize modern cryptography is not novel. Indeed, multiple solutions have been proposed [7,34,6,33], based on Attribute-Based Encryption (ABE) [39,4], as well as on Symmetric Searchable Encryption (SSE) [9,10]. However, to the best of our knowledge, this is the first work that aims at designing a provable secure protocol leveraging the functionality of HHE.

3 Preliminaries

Notation If $\mathcal{Y}$ is a set, we use $y \xleftarrow{\$} \mathcal{Y}$ if y is chosen uniformly at random from $\mathcal{Y}$. Concatenation of two strings x, y is denoted by $x \| y$. A probabilistic polynomial time (PPT) adversary $\mathcal{ADV}$ is a randomized algorithm for which there exists a polynomial $p(z)$ such that for all input z, the running time of $\mathcal{ADV}(z)$ is bounded by $p(|z|)$. A function $negl(\cdot)$ is called negligible if $\forall\, c \in \mathbb{N}, \exists\, \epsilon_0 \in \mathbb{N}$ such that $\forall\, \epsilon \geq \epsilon_0 : negl(\epsilon) < \epsilon^{-c}$.

Definition 1 (Homomorphic Encryption). *A (public-key) homomorphic encryption scheme is a quadruple of PPT algorithms* $\mathsf{HE} = (\mathsf{HE.KeyGen}, \mathsf{HE.Enc}, \mathsf{HE.Dec}, \mathsf{HE.Eval})$ *such that:*

- ***Key Generation:*** *The Key Generation algorithm* $(\mathsf{pk}, \mathsf{evk}, \mathsf{sk}) \leftarrow \mathsf{He.Keygen}$ $(1^{\breve{}})$ *takes as input a unary representation of the security parameter* λ*, and outputs a public key* pk*, a public evaluation key* evk *and a secret decryption key* sk*.*
- ***Encryption:*** *This algorithm* $c \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, m)$ *takes as input the public key* pk *and a message* m *and outputs a ciphertext* c*.*

- ***Decryption:*** *This algorithm* $m \leftarrow \mathsf{HE.Dec}(\mathsf{sk}, c)$*, takes as input the secret key* sk *and a ciphertext* c*, and outputs a plaintext* m*.*
- ***Homomorphic Evaluation:*** *This algorithm* $c_f \leftarrow \mathsf{He.Eval}(\mathsf{evk}, f, c_1, \ldots, c_n)$ *takes as input the evaluation key* evk*, a function* f*, and a set of* n *ciphertexts, and outputs a ciphertext* c_f*.*

Correctness: An HE scheme is said to be correct if and only if:

$$\Pr[\mathsf{HE.Dec}_{\mathsf{sk}}(\mathsf{HE.Eval}_{\mathsf{evk}}(f, c)) \neq f(m) | \mathsf{HE.Enc}_{\mathsf{pk}}(m) = c] \leq negl(\lambda) \quad (1)$$

Before we proceed with the formal definition of HHE, we discuss its functionality at a high-level. An HHE scheme is built on top of a traditional HE scheme as well as a symmetric cipher SKE. The *Key Generation* algorithm of HHE invokes the corresponding algorithms of both the HE and SKE and outputs $(\mathsf{pk}, \mathsf{sk}, \mathsf{evk})$ for the HE scheme, and K for the SKE scheme. As a next step, the *Encryption* algorithm takes as input a message m, HE's public key pk, and K. The message m will be encrypted symmetrically using K, resulting to a ciphertext c. Moreover, the symmetric key K will be homomorphically encrypted under pk, resulting to another ciphertext c_{K}. These two ciphertexts will then be given as input, along with the decryption function of SKE, to HHE's *Decompression* algorithm. This algorithm homomorphically performs the symmetric decryption circuit to transform the symmetric ciphertext c into a homomorphic ciphertext c', by invoking the evaluation algorithm of the HE scheme. Finally, the evaluation and decryption algorithms of HHE, are identical to those of the HE scheme.

Definition 2 (Hybrid Homomorphic Encryption). *Let* HE *be a Homomorphic Encryption scheme and* $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a symmetric-key encryption scheme. Moreover, let* $\mathcal{M} = (m_1, \ldots, m_n)$ *be the message space and* λ *the security parameter. An* HHE *scheme then consists of five PPT algorithms such that* $\mathsf{HHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Decomp}, \mathsf{Eval}, \mathsf{Dec})$ *and it is constructed as follows:*

```
$\mathsf{HHE.KeyGen}(1^\lambda)$:
$(\mathsf{pk}, \mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{HE.KeyGen}(1^\lambda)$
Return $(\mathsf{pk}, \mathsf{sk}, \mathsf{evk})$

$\mathsf{HHE.Enc}$:
$\mathsf{K} \leftarrow \mathsf{SKE.Gen}(1^\lambda)$
$c_{\mathsf{K}} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, \mathsf{K})$
$c \leftarrow \mathsf{SKE.Enc}(\mathsf{K}, m)$
Return $(c_{\mathsf{K}}, c)$

$\mathsf{HHE.Decomp}(\mathsf{evk}, c, c_{\mathsf{K}})$:
$c' \leftarrow \mathsf{HE.Eval}(\mathsf{evk}, \mathsf{SKE.Dec}, \mathsf{c_K}, c)$
Return $c'$

$\mathsf{HHE.Eval}(\mathsf{evk}, f, c'_1, \ldots, c'_n)$:
Return $\mathsf{HE.Eval}(\mathsf{evk}, f, c'_1, \ldots, c'_n)$

$\mathsf{HHE.Dec}(\mathsf{sk}, c')$ :
Return $\mathsf{HE.Dec}(\mathsf{sk}, \mathsf{c}')$
```

Fig. 1: Hybrid Homomorphic Encryption Scheme

The correctness of an HHE scheme follows directly from the correctness of the underlying public-key HE scheme.

For the security of HHE we rely on the following theorem that was first proved for the KEM/DEM paradigm in [31], and then later modified for HHE in [23]:

Theorem 1. *Let* HE *be an IND-CPA secure public-key homomorphic encryption scheme. Moreover, let* SKE *be an IND-CPA secure symmetric-key encryption scheme. Then the* HHE *scheme instantiated by* HE *and* SKE *is IND-CPA secure.*

4 Architecture

For the needs of our construction, we assume the existence of the following three entities:

- **Cloud Service Provider (CSP):** An honest-but-curious cloud service provider that is primarily responsible for gathering symmetrically encrypted data from multiple sources. The CSP undertakes the task of transforming the symmetrically encrypted data to homomorphic ciphertexts and, upon request, operate on them in a blind way.
- **Analyst (A):** The analyst is an entity that wishes to perform computations on the data of various users. **A** is the only entity in our construction that can perform the homomorphic decryption and thus, gain insights from user data.
- **Users ($\mathcal{U}$):** Users encrypt their data locally using a symmetric-key encryption scheme and outsource them to the CSP.

5 Symmetrical Disguise

Before we proceed with the formal construction of the scheme, we provide a high-level overview.

5.1 High-Level Overview

An analyst **A** generates $(\mathsf{pk}, \mathsf{sk}, \mathsf{evk})$ for the HHE scheme, outsources evk to the CSP and publishes pk. As a next step, each user u_i ($u_i \in \mathcal{U}$), can generate a symmetric key locally, encrypt their data, and outsource them to the CSP along with a homomorphic encryption of the symmetric key under A's public key. Upon reception, the CSP transforms the symmetric ciphertexts to homomorphic, and stores them online in its database. **A** can request the evaluation of a function f on the collection of the ciphertexts from the CSP. The CSP uses evk and outputs an encrypted result which then sends back to **A**. Finally, **A** decrypts the result using their secret key sk.

5.2 Formal Construction

We are now ready to present SD that constitutes the core of this paper's contribution. For the realization of our construction we rely on the following building blocks:

- An IND-CPA secure symmetric cipher $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.
- An IND-CPA secure homomorphic encryption scheme $\mathsf{HE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$.
- A CCA2 secure public-key encryption scheme $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$
- An EUF-CMA secure signature scheme $\mathsf{S} = (\mathsf{sign}, \mathsf{ver})$.
- A first and second pre-image resistant hash function H.

SD is built around three main protocols: $\mathsf{Setup}, \mathsf{Add}, \mathsf{Query}$ such that:

SD.Setup: Each entity from the described architecture generates a signing/verification key pair for an EUF-CMA secure signature scheme S and publishes its verification key while keeping the signing key private. Apart from that, the CSP, generates a public/private key pair $(\mathsf{pk}, \mathsf{sk})$ for a CCA2-secure public-key encryption scheme PKE. Finally, the analyst A runs $\mathsf{HHE.KeyGen}$ to generate the public, secret and evaluation keys for an IND-CPA secure homomorphic encryption scheme HE, and each user u_i runs $\mathsf{SKE.KeyGen}$ to generate a symmetric key K_i for an IND-CPA secure symmetric cipher SKE. Below we provide a list of the generated keys:

- Signing/Verification keys for each entity.
- $(\mathsf{pk}_{\mathsf{CSP}}, \mathsf{sk}_{\mathsf{CSP}})$: Public/private key pair of the CSP.
- $(\mathsf{pk}_{\mathsf{A}}, \mathsf{sk}_{\mathsf{A}}, \mathsf{evk}_{\mathsf{A}})$: Public/private/evaluation keys of A.
- K_{i}: Symmetric key for each user u_i.

Once the keys are generated, **A** outsources its evaluation key evk to the CSP via $m_1 = \langle t_1, \mathsf{Enc}(\mathsf{pk}_{\mathsf{CSP}}, \mathsf{evk}), \sigma_A(H(t_1||\mathsf{evk})))\rangle$, where t_1 is a timestamp, σ_A is a signature encrypted with **A**'s private key, and H is a hash function. Upon reception, the CSP verifies the signature of **A**, using **A**'s verification key, and the freshness of the message through the timestamp. If a verification fails, the CSP aborts the protocol and outputs $\perp$. Otherwise, the CSP stores $\mathsf{evk}_{\mathsf{A}}$.

SD.Add: This protocol is initiated by any user $u_i \in \mathcal{U}$ that wishes to outsource some data $\mathbf{x} = (x_1, \ldots, x_n)$ to the CSP. To do so, u_i first runs $c_i \leftarrow \mathsf{SKE.Enc}(\mathsf{K}_{\mathsf{i}}, \mathsf{x}_{\mathsf{i}})$. As a next step, u_i, homomorphically encrypts is symmetric key K_{i} under **A**'s public key, by running $c_{\mathsf{K}_i} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_{\mathsf{A}}, \mathsf{K}_i)$. Finally, the $(\mathbf{c}, c_{\mathsf{K}_i})$ pair is outsourced to the CSP via the following message:

$$m_2 = \langle t_2, \mathbf{c} = (c_i, \ldots, c_n), c_{\mathsf{K}_{\mathsf{i}}}\sigma_{u_i}(H(t_2\|\mathbf{c}\|c_{\mathsf{K}_{\mathsf{i}}}))\rangle.$$

Upon receiving m_2, the CSP verifies the freshness and integrity of the message. If the verifacation fails, CSP outputs $\perp$. Otherwise, it transforms the symmetric ciphertext c_i to a homomorphic one, by running $c'_i \leftarrow \mathsf{HHE.Decomp}(c_i, c_{\mathsf{K}_i}, \mathsf{evk}_{\mathsf{A}})$. Finally, the CSP stores c'_i in its database.

$\boxed{\textbf{SD.Query:}}$ The Query protocol is initiated by **A** whenever she wishes to issue a query to the encrypted data for a function f. To do so, A sends $m_3 = \langle t_3, \mathsf{Enc}(\mathsf{pk}_{\mathsf{CSP}}, f), \sigma_A(H(t_3 \| f))\rangle$ to the CSP. Upon reception, the CSP verifies both the integrity and the freshness of the message. If the verification fails, the CSP will abort the protocol and output the error symbol $\perp$. Otherwise, it runs $\mathsf{HHE.Eval}(f, \mathsf{evk}_{\mathsf{A}}, c'_1, \ldots, c'_n) \rightarrow c_{\mathsf{res}}$ to get an encrypted result c_{res}. Due to the homomorphic properties of the encryption scheme HE, the encrypted result c_{res} can be viewed as an encrypted version of $f(x'_1, \ldots, x'_n)$, where each x'_i corresponds to a ciphertext c'_i, and that can only be dercypted using **A**'s secret key sk. Subsequently, the CSP forwards c_{res} to A via $m_4 = \langle t_4, c_{\mathsf{res}}, \sigma_{CSP}(H(t_4 \| c_{\mathsf{res}}))\rangle$. Upon reception, A verifies both the integrity and the freshness of the message. If a verification fails, A aborts the protocol and outputs $\perp$. Otherwise, they run $\mathsf{HHE.Dec}(\mathsf{sk}, c_{res}) \rightarrow \mathsf{res}$ to retrieve the result res. Having acquired the result in plaintext, **A** can use it to perform statistics or data analysis, in a privacy-preserving manner, since she never got access to the actual plaintexts. Our protocol is illustrated in Figure 2.

6 Threat Model

In this section, we define the threat model under which we prove the security of SD. More specifically, we formalize the capabilities of the adversary $\mathcal{ADV}$ through the following set of possible attacks:

Attack 1 (Analyst Substitution Attack) *Let $\mathcal{ADV}$ be a malicious adversary. $\mathcal{ADV}$ successfully performs an Analyst Substitution Attack if she manages to convince the users that their data are processed for the needs of an analyst* $\mathbf{A}$*, while in reality they are processed for an analyst* $\mathbf{A}^{\mathcal{ADV}}$.

Attack 2 (Ciphertext Substitution Attack) *Let $\mathcal{ADV}$ be a malicious adversary. $\mathcal{ADV}$ successfully launches a Ciphertext Substitution Attack if she manages to replace the ciphertexts sent by users to the CSP in an indistinguishable way.*

Attack 3 (Query Substitution Attack) *Let $\mathcal{ADV}$ be a malicious adversary. $\mathcal{ADV}$ successfully launches a Query Substitution Attack if she manages to replace the query sent by A to the CSP, with another one of her choice, in an indistinguishable way.*

Attack 4 (Result Substitution Attack) *Let $\mathcal{ADV}$ be a malicious adversary. $\mathcal{ADV}$ successfully launches a Result Substitution Attack if she manages to replace the result sent by the CSP to the analyst A, in an indistinguishable way.*

In our threat model, we assume that the CSP cannot collude with the Analyst **A**. This is a valid assumption as otherwise we would be required to prove security in a setting where the decryption keys are publicly-available.

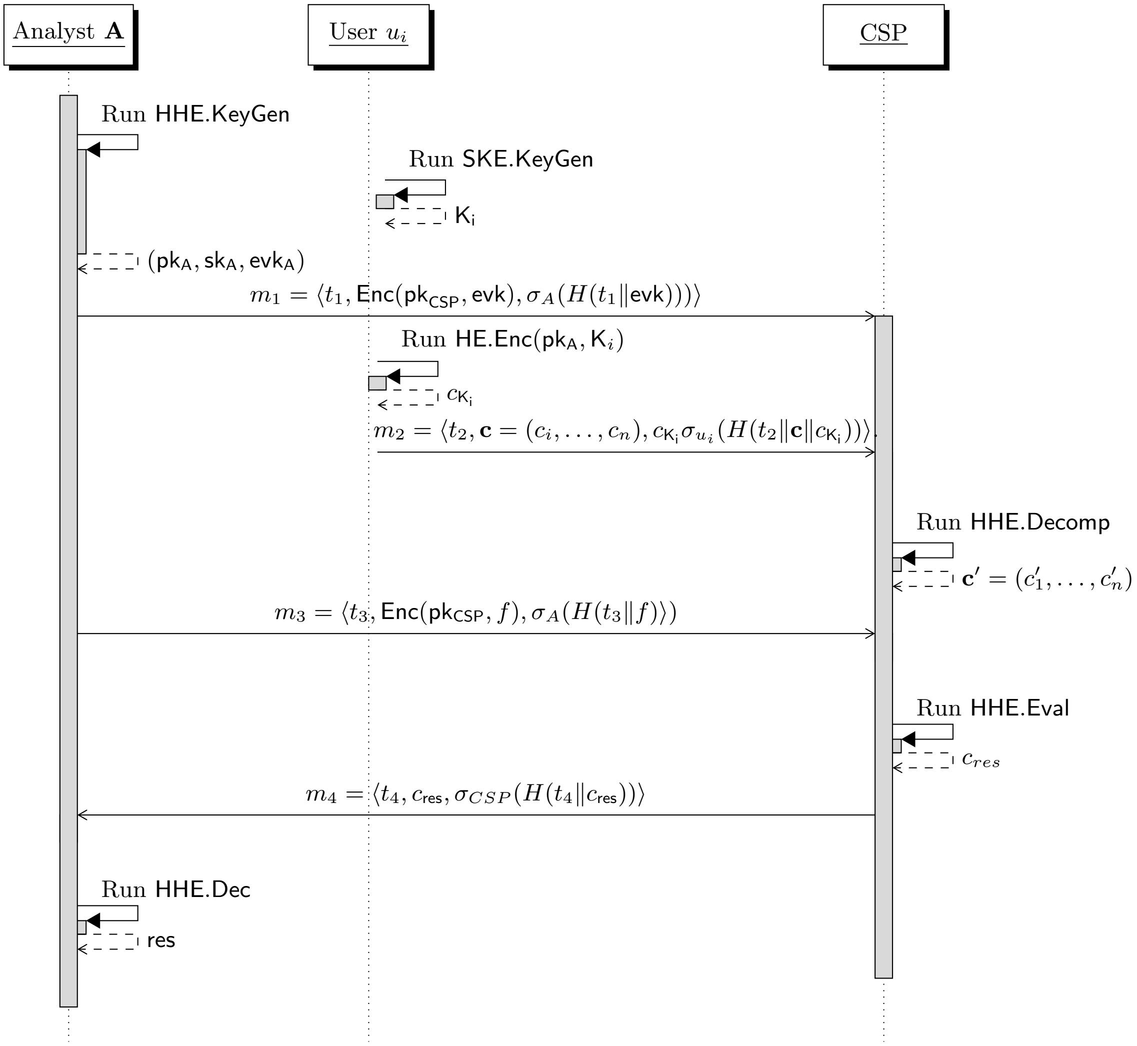
Analyst **A**
User u_i
CSP
Run HHE.KeyGen
Run SKE.KeyGen
$\mathsf{K_i}$
$(\mathsf{pk_A}, \mathsf{sk_A}, \mathsf{evk_A})$
$m_1 = \langle t_1, \mathsf{Enc}(\mathsf{pk_{CSP}}, \mathsf{evk}), \sigma_A(H(t_1 \| \mathsf{evk})))\rangle$
Run HE.Enc$(\mathsf{pk_A}, \mathsf{K}_i)$
$c_{\mathsf{K_i}}$
$m_2 = \langle t_2, \mathbf{c} = (c_i, \ldots, c_n), c_{\mathsf{K_i}} \sigma_{u_i}(H(t_2 \| \mathbf{c} \| c_{\mathsf{K_i}}))\rangle.$
Run HHE.Decomp
$\mathbf{c}' = (c'_1, \ldots, c'_n)$
$m_3 = \langle t_3, \mathsf{Enc}(\mathsf{pk_{CSP}}, f), \sigma_A(H(t_3 \| f))\rangle)$
Run HHE.Eval
c_{res}
$m_4 = \langle t_4, c_{\mathsf{res}}, \sigma_{CSP}(H(t_4 \| c_{\mathsf{res}}))\rangle$
Run HHE.Dec
res

Fig. 2: Complete run of our protocol with one user u_i. More users would behave exactly like u_i.

7 Security Analysis

We are now ready to prove the security of our construction assuming the threat model defined in section 6. In particular, we will prove the following theorem:

Theorem 2 (SD Security in the presence of Malicious Adversaries). *Let* PKE *be an INC-CPA secure public-key encryption scheme and* S *an EUF-CMA secure signature scheme with security parameter* λ*. Moreover, let* SKE *be an IND-CPA secure symmetric-key encryption scheme with security parameter* κ*. Finally, let* $\mathcal{ADV}$ *be a malicious adversary. Then, SD is secure against the threat model defined in section 6.*

Proof. To prove Theorem 2, we will start with a sequence of lemmas. Then, we will combine our results to derive a proof for the main theorem.

Lemma 1 (Analyst Substitution Attack Soundness). *Let* PKE *be an INC-CPA secure public-key encryption scheme. Moreover, let* S *be an EUF-CMA secure signature scheme and* $\mathcal{ADV}$ *a malicious adversary. In this case* $\mathcal{ADV}$ *cannot successfully launch an Analyst Substitution Attack against SD.*

Proof. $\mathcal{ADV}$ will successfully launch an Analyst Substitution Attack, by targeting either the $\mathsf{SD.Setup}$ or the $\mathsf{SD.Add}$ protocol. To this end, we distinguish the following cases:

C1: Attacking $\mathsf{SD.Setup}$: To perform an attack against $\mathsf{SD.Setup}$, $\mathcal{ADV}$ needs to swap the evaluation key of A, evk_A, with an evaluation key $\mathsf{evk}_{\mathbf{A}^{\mathcal{ADV}}}$, for an analyst $\mathbf{A}^{\mathcal{ADV}}$ such that $A \neq \mathbf{A}^{\mathcal{ADV}}$. To this end, ADV targets the $m_1 = \langle t_1, \mathsf{Enc}(\mathsf{pk}_{\mathsf{CSP}}, \mathsf{evk}), \sigma_A(H(t_1 \| \mathsf{evk})))\rangle$ message sent from A to the CSP and tries to swap evk_A with $\mathsf{evk}_{\mathbf{A}^{\mathcal{ADV}}}$. Generating a valid $\mathsf{Enc}_{\mathsf{pk}_{\mathsf{CSP}}}(\mathsf{evk}'_A)$ is straightforward for $\mathcal{ADV}$ as $\mathsf{pk}_{\mathsf{CSP}}$ is publicly known. However, swapping evk_A for evk'_A in the $\sigma_A(H(t_1||\mathsf{evk}))$ term, is equivalent with forging A's signature, and given the EUF-CMA security of the signature scheme S, this can only happen with negligible probability. More specifically, if λ is the security parameter of S, then the advantage ϵ_1 of $\mathcal{ADV}$ is successfully tampering with m_1 in an indistinguishable way is:

$$\epsilon_1 = negl(\lambda) \tag{2}$$

C2: Attacking $\mathsf{SD.Add}$: Another option for $\mathcal{ADV}$ is to target

$$m_2 = \langle t_2, \mathbf{c} = (c_i, \ldots, c_n), c_{\mathsf{K_i}} \sigma_{u_i}(H(t_2 \| \mathbf{c} \| c_{\mathsf{K_i}}))\rangle.$$

The motivation for this attack is to use user data for an analyst $\mathbf{A}^{\mathcal{ADV}}$ while the users believe that their data will be processed for an analyst A. Recall that $c_{\mathsf{K_i}}$ is generated as $c_{\mathsf{K}_i} \leftarrow \mathsf{HE.Enc}_{\mathsf{pk_A}}(\mathsf{K}_i)$. Hence, for $\mathcal{ADV}$ to successfully attack this protocol, they need to ***simultaneously*** satisfy the following three conditions:

(a) Guess the symmetric key K_i;

(b) Encrypt it with the public key of another analyst $\mathbf{A}^{\mathcal{ADV}}$;
(c) Tamper with m_2 in an indistinguishable way.

However, assuming that the symmetric cipher SKE is IND-CPA secure, the probability of correctly guessing the key (e.g. brute force attack) is negligible in the security parameter κ of SKE. Hence, if the advantage of $\mathcal{ADV}$ in guessing they key is ϵ_2:

$$\epsilon_2 = negl(\kappa) \tag{3}$$

Since condition (1) can never be fulfilled, except with negligible probability, there is no need to separately examine conditions (2) and (3).

Hence, we conclude that in every case, $\mathcal{ADV}$ can successfully launch an Analyst Substitution Attack with only negligible probability.

Lemma 2 (Ciphertext Substitution Attack Soundness). *Let* PKE *be an INC-CPA secure public-key encryption scheme. Moreover, let* S *be an EUF-CMA secure signature scheme and* $\mathcal{ADV}$ *a malicious adversary. Then* $\mathcal{ADV}$ *cannot successfully launch a Ciphertext Substitution Attack against SD.*

In contrast with the previous attack that aimed at processing real user data, this attack aims at substituting the actual ciphertexts $(c_1, \ldots, c_n)$ with a sequence of data $(c'_1, \ldots, c'_n)$ generated by $\mathcal{ADV}$. By succeeding in this attack, $\mathcal{ADV}$ can control the outcome of a query to the CSP and hence, manipulate the analyst $\mathbf{A}$.

Proof. Successfully performing a Ciphertext Substitution Attack, requires attacking the $\mathsf{SD.Add}$ protocol. More precisely, when a user u_i outsources their data to the CSP via $m_2 = \langle t_2, \mathbf{c} = (c_i, \ldots, c_n), c_{\mathsf{K_i}} \sigma_{u_i}(H(t_2 \| \mathbf{c} \| c_{\mathsf{K_i}}))\rangle$, $\mathcal{ADV}$ needs to substitute $\mathbf{c} = (c_i, \ldots, c_n)$ with $\mathbf{c}' = (c'_1, \ldots, c'_n)$. Apart from that $\mathcal{ADV}$ needs to generate a $c_{\mathsf{K}'}$ term where K' is the key used to encrypt $\mathbf{c}'$. More precisely, $\mathcal{ADV}$ needs to successfully:

1. Generate a symmetric key $\mathsf{K}_{\mathcal{ADV}}$;
2. Use K' to generate a sequence of ciphertexts $\mathbf{c}' = (c'_1, \ldots, c'_n)$;
3. Encrypt $\mathsf{K}_{\mathcal{ADV}}$ with pk_A to get $c_{\mathsf{K}_{\mathcal{ADV}}}$;
4. Tamper with m_2 in an indistinguishable way.

Conditions (1), (2) and (3) are trivial to achieve. Moreover, substituting $\mathbf{c}$ with $\mathbf{c}'$ and c_{K_i} with c_{K} in the first part of m_2 is straightforward. However, these terms are also included in the signature and hence, successfully substituting the terms is equivalent to forging u_i's signature. Given the EUF-CMA security of the signature scheme S, this can only happen with negligible probability in the security parameter λ of S. As a result, the advantage ϵ_2 in tampering with m_2 in an indistinguishable way is:

$$\epsilon_3 = negl(\lambda) \tag{4}$$

Lemma 3 (Query Substitution Attack Soundness). *Let* PKE *be an INC-CPA secure public-key encryption scheme. Moreover, let* S *be an EUF-CMA secure signature scheme and* $\mathcal{ADV}$ *a malicious adversary. Then* $\mathcal{ADV}$ *cannot successfully launch a Query Substitution Attack against SD.*

Proof. For $\mathcal{ADV}$ to successfully perform a Query Substitution Attack, they need to attack the $\mathsf{SD.Query}$ protocol. More precisely, when A sends $m_3 = \langle t_3, \mathsf{Enc}(\mathsf{pk}_{\mathsf{CSP}}, f), \sigma_A(H(t_3 \| f)) \rangle$ tries to substitute the function f with another function f' of their choice. Since f is encrypted with the public key of the CSP $\mathsf{pk}_{\mathsf{CSP}}$, $\mathcal{ADV}$ simply needs to encrypt f' under pk_{CSP} as well. However, f is also included in the signature part of m_3 and hence, tampering with m_3 requires forging $\mathbf{A}^{\mathcal{ADV}}$s signature. Given the EUF-CMA security of the signature scheme S this can only happen with negligible probability in the security parameter λ of S. As a result, $\mathcal{ADV}$'s advantage ϵ_4 is in tampering with m_3 in an indistinguishable way is:

$$\epsilon_4 = negl(\lambda) \tag{5}$$

Lemma 4 (Result Substitution Attack Soundness). *Let* PKE *be an INC-CPA secure public-key encryption scheme. Moreover, let* S *be an EUF-CMA secure signature scheme and* $\mathcal{ADV}$ *a malicious adversary. Then* $\mathcal{ADV}$ *cannot successfully launch a Result Substitution Attack against SD.*

Proof. The proof is identical to that of Lemma 3 with the main difference being that $\mathcal{ADV}$ targets m_4 instead of m_3. Hence, following the exact same reasoning as in the proof of Lemma 3, we conclude that the advantage ϵ_5 of $\mathcal{ADV}$ is in tampering with m_4 in an indistinguishable way is:

$$\epsilon_4 = negl(\lambda) \tag{6}$$

Having examined each possible attack separately, what remains to be done is to prove that the overall advantage ϵ_{total} of $\mathcal{ADV}$ is negligible. Given the security parameter λ and grouping up the results from equations 2- 6 we get that:

$$\begin{aligned} \epsilon_{total} &= \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4 + \epsilon_5 \\ &= 4 \cdot negl(\lambda) + negl(\kappa) \end{aligned} \tag{7}$$

However, it is a standard result in real analysis that the finite sum of negligible functions is still negligible and hence:

$$\epsilon_{total} = negl'(\lambda, \kappa), \tag{8}$$

where $negl'(\lambda, \kappa)$ is negligible function produced as a linear combination of $negl(\lambda)$ and $negl(\kappa)$ □

8 Evaluation

In this section, we evaluate the performance of the core algorithms of our protocol. Our primary testbed for these experiments was an Intel Core i7 laptop with 16GB RAM running an Ubuntu 20.04 operating system. For these experiments, we utilized the SEAL cryptographic library [41] for basic HE operations, PASTA library [23] to implement the secure symmetric cipher, and OpenSSL[2]. PASTA was chosen over more established Symmetric ciphers such as AES due to its low multiplicative index. All HE operations in this section were based on the BFV [14] scheme, with a polynomial modulus degree of 16384. We note that the choice of polynomial modulus degree impacted the efficiency of the implemented scheme and increased the size of the ciphertexts, however, this was necessary due to the complex operations involved. Finally, to provide a comprehensive overview of each algorithm's performance, each experiment was conducted 50 times with the average taken.

8.1 Performance of Core Protocols

In this phase of our evaluations, we focused on the performance of the $\mathsf{SD.Setup}$, $\mathsf{SD.Add}$, and $\mathsf{SD.Query}$ protocols.

$\boxed{\mathsf{SD.Setup:}}$ When evaluating the $\mathsf{SD.Setup}$ protocol, we first measured the time taken to generate an RSA public and private key pair (2048 bit long), which we used for both Signing/Verification and Encryption/Decryption, and the time taken to generate the HE keys for the Analyst (i.e., Public, Secret and Evaluation keys). On an average, it took 34.6 milliseconds to generate the RSA public and private keypair, and 88.4 milliseconds to generate the HE keys. Finally, we measured the time taken by a user to construct $m_1 = \langle t_1, \mathsf{Enc}(\mathsf{pk}_{\mathsf{CSP}}, \mathsf{evk}), \sigma_A(H(t_1\|\mathsf{evk})))\rangle$, and the time taken by the CSP to verify m_1 and decrypt $\mathsf{Enc}(\mathsf{pk}_{\mathsf{CSP}}, \mathsf{evk})$. Constructing m_1 took 1.478 milliseconds, while verifying and decrypting m_1 took 1.174 milliseconds (Table 2).

$\boxed{\mathsf{SD.Add:}}$ For the $\mathsf{SD.Add}$ protocol, we evaluated the cost of homomorphically encrypting the symmetric key ($\mathsf{HE.Enc}_{\mathsf{pk}}$), cost of symmetrically encrypting the user's data ($\mathsf{SKE.Enc}$), and cost of transforming the symmetric ciphertext to a homomorphic ciphertext ($\mathsf{HHE.Decomp}$). Additionally, we measured the time taken for a user to construct $m_2 = \langle t_2, \mathbf{c} = (c_i, \ldots, c_n), c_{\mathsf{K_i}}\sigma_{u_i}(H(t_2\|\mathbf{c}\|c_{\mathsf{K_i}}))\rangle$, and time taken for the CSP to verify m_2. Each experiment was run with a varying number of user data from 1 to 200. It is worth re-iterating that one of the primary advantages of SD is that irrespective of the amount of data being outsourced, $\mathsf{HE.Enc}_{\mathsf{pk}}$ is executed once. The cost of executing $\mathsf{HE.Enc}_{\mathsf{pk}}$ once was 18 milliseconds. When outsourcing one data value, it took 7 milliseconds to execute the $\mathsf{SKE.Enc}$ algorithm and 17.7 seconds to run the $\mathsf{HHE.Decomp}$

[2] https://github.com/openssl/openssl

algorithm on average. On the other hand, when outsourcing 200 data values, the SKE.Enc algorithm took 1.22 seconds to execute, while the HHE.Decomp algorithm took 3824 seconds to execute (Table 1). Constructing m_2 took 1.057 milliseconds, while verifying m_2 took 0.101 milliseconds (Table 2).

$\boxed{\mathsf{SD.Query:}}$ Meanwhile for the SD.Query protocol, we focused on the cost of executing the HHE.Eval and HHE.Dec algorithms. Once again, each experiment was run with a varying number of user data from 1 to 200. Additionally, for the HHE.Eval algorithm, we evaluated a simple squaring function. For a single data value, it took 91 milliseconds to execute the HHE.Eval algorithm, and 5 milliseconds to execute the HHE.Dec algorithm. While for 200 data values, the HHE.Eval algorithm took on average 16.9 seconds to execute, with the HHE.Dec algorithm taking 1.07 seconds to execute (Table 1). Furthermore, the analyst takes 1.098 milliseconds to construct $m_3 = \langle t_3, \mathsf{Enc}(\mathsf{pk}_{\mathsf{CSP}}, f), \sigma_A(H(t_3 \| f))\rangle$, while the CSP takes 1.56 milliseconds to verify m_3 and decrypt $\mathsf{Enc}(\mathsf{pk}_{\mathsf{CSP}}, f)$. Finally, the CSP took 1.118 milliseconds to construct $m_4 = \langle t_4, c_{\mathsf{res}}, \sigma_{CSP}(H(t_4 \| c_{\mathsf{res}}))\rangle$ (Table 2).

From our evaluations, it is quite obvious that the HHE.Decomp algorithm is the most computationally expensive function, which explains why it is executed by the CSP. We provide a complete overview of SD protocol measurements when outsourcing 200 data values in Table 2.

Data Values	SKE.Enc	HHE.Decomp	HHE.Eval	HHE.Dec
1	7 ms	17765 ms	91 ms	5 ms
50	0.31 s	990.2 s	4.916 s	0.275 s
100	0.61 s	1920.8 s	8.418 s	0.54 s
150	0.93 s	2832.1 s	12.592 s	0.81 s
200	1.22 s	3823.8 s	16.902 s	1.07 s

Table 1: Algorithm Execution

8.2 Comparison with plain BFV

To provide concrete evidence of the efficiency of SD, we compare the operations at its user side with that of a plain BFV scheme. To be more precise, we measured the performance of a plain BFV scheme where a user continuously encrypts each data value homomorphically before the data is outsourced to the CSP. We used the same encryption parameters as with SD. For these experiments, we compared the total cost of executing the $\mathsf{HE.Enc}_{\mathsf{pk}}$ and SKE.Enc algorithms of the SD.Add protocol, and the cost of continuously using HE encryption in the plain BFV.

As with the previous experiments, we vary the amount of data from 1 to 200. For a single data value, SD.Add takes 25 milliseconds to execute, while the plain

Sub-Protocol	Messages	Analyst Functions	User Functions	CSP Functions	Time (s)
SD.Setup	m_1	PKE Keygen HHE.KeyGen m_1 construction	PKE Keygen	PKE Keygen m_1 verification	0.19
SD.Add	m_2	-	HHE.Enc SKE.Enc m_2 construction	m_2 verification HHE.Decomp	3825
SD.Query	m_3, m_4	m_3 construction	-	m_3 verification HHE.Eval m_4 construction	16.91

Table 2: Protocol Measurements in the case of 200 data values

BFV scheme takes 21 milliseconds to perform one HE encryption. It is worth mentioning that for a single data value, the plain BFV scheme is marginally faster than SD at the user side. However, this is easily attributed to the fact that SD requires two operations (a symmetric encryption operation plus an HE encryption operation) at the user side, while the plain BFV scheme involves just the one HE encryption operation. When the number of data values is increased to 200, the SD.Add algorithm executes in 1.22 seconds, while the plain BFV scheme executes in 3.1 seconds. Figure 3 provides an overview of all the results obtained from this phase of our experiments. From these results, it is evident that SD considerably reduces the computational costs of the user and transfers majority of the computational costs to the CSP.

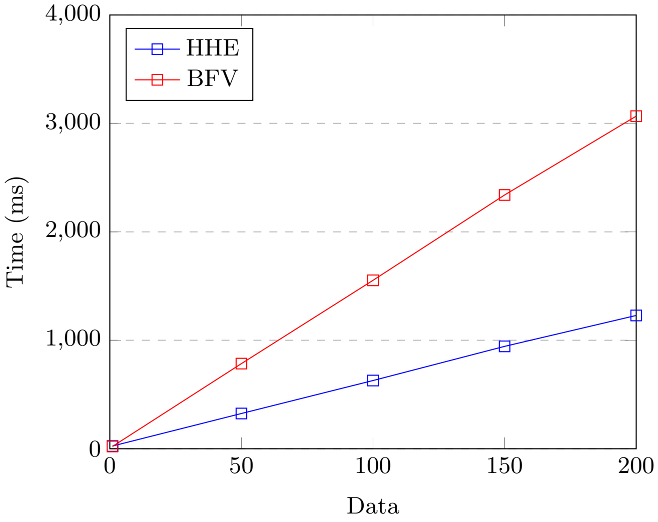

Fig. 3: Computation Time on the user's side

Science & Reproducible Research: To support open science and reproducible research, and provide other researchers with the opportunity to use, test, and hopefully extend our scheme, the source code used for the evaluations is publicly available online[3].

9 Conclusion

In this paper we presented SD; a secure cryptographic protocol based on Hybrid Homomorphic Encryption. The security and applicability of our construction have been demonstrated through a detailed security analysis and an extensive experimental evaluation. It is our firm belief that in the years to come, cloud storage services will rely less on traditional cryptographic primitives and more on modern cryptographic techniques allowing flexible computations over the encrypted data – such as HE. To this end, we believe it is vital to start designing realistic architectures based on HE in an attempt to demonstrate the feasibility and applicability of modern cryptography. We hope that our work will incentivize other researchers to look into the same direction. Most importantly, though, we hope it will help companies to create modern privacy-respecting cloud services.

References

1. Abdalla, M., , D., Fiore, D., Gay, R., Ursu, B.: Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings. In: Advances in Cryptology – CRYPTO 2018 (2018)
2. Abdalla, M., Bourse, F., De Caro, A., Pointcheval, D.: Simple functional encryption schemes for inner products. In: IACR International Workshop on Public Key Cryptography. pp. 733–751. Springer (2015)
3. Abdalla, M., Gay, R., Raykova, M., Wee, H.: Multi-input inner-product functional encryption from pairings. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer
4. Agrawal, S., Chase, M.: Fame: Fast attribute-based message encryption. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. p. 665–682. CCS '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3134014, `https://doi.org/10.1145/3133956.3134014`
5. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for mpc and fhe. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 430–454. Springer (2015)
6. Bakas, A., Dang, H.V., Michalas, A., Zalitko, A.: The cloud we share: Access control on symmetrically encrypted data in untrusted clouds. IEEE Access **8**, 210462–210477 (2020). https://doi.org/10.1109/ACCESS.2020.3038838
7. Bakas, A., Michalas, A.: Modern family: A revocable hybrid encryption scheme based on attribute-based encryption, symmetric searchable encryption and sgx. In: Chen, S., Choo, K.K.R., Fu, X., Lou, W., Mohaisen, A. (eds.) Security and Privacy in Communication Networks. pp. 472–486. Springer International Publishing, Cham (2019)

[3] https://github.com/iammrgenie/HHE-Protocol

8. Bakas, A., Michalas, A.: Multi-input functional encryption: efficient applications from symmetric primitives. In: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). pp. 1105–1112. IEEE (2020)
9. Bakas, A., Michalas, A.: Power range: Forward private multi-client symmetric searchable encryption with range queries support. In: 2020 IEEE Symposium on Computers and Communications (ISCC). pp. 1–7 (2020). https://doi.org/10.1109/ISCC50000.2020.9219739
10. Bakas, A., Michalas, A.: Nowhere to leak: A multi-client forward and backward private symmetric searchable encryption scheme. In: Barker, K., Ghazinour, K. (eds.) Data and Applications Security and Privacy XXXV. pp. 84–95. Springer International Publishing, Cham (2021)
11. Bakas, A., Michalas, A., Dimitriou, T.: Private lives matter: A differential private functional encryption scheme. In: Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy. p. 300–311. CODASPY '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3508398.3511514, https://doi.org/10.1145/3508398.3511514
12. Bakas, A., Michalas, A., Frimpong, E., Rabbaninejad, R.: Feel the quantum functioning: Instantiating generic multi-input functional encryption from learning with errors (extended version)? Cryptology ePrint Archive, Paper 2022/629 (2022), https://eprint.iacr.org/2022/629, https://eprint.iacr.org/2022/629
13. Bakas, A., Michalas, A., Ullah, A.: (f) unctional sifting: A privacy-preserving reputation system through multi-input functional encryption. In: Nordic Conference on Secure IT Systems. pp. 111–126. Springer (2020)
14. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Annual Cryptology Conference. pp. 868–886. Springer (2012)
15. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
16. Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., Sirdey, R.: Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. Journal of Cryptology **31**(3), 885–916 (2018)
17. Cheon, J.H., Coron, J.S., Kim, J., Lee, M.S., Lepoint, T., Tibouchi, M., Yun, A.: Batch fully homomorphic encryption over the integers. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 315–335. Springer (2013)
18. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 409–437. Springer (2017)
19. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)
20. Cid, C., Indrøy, J.P., Raddum, H.: Fasta – a stream cipher for fast fhe evaluation. Cryptology ePrint Archive, Report 2021/1205 (2021), https://ia.cr/2021/1205
21. Coron, J.S., Lepoint, T., Tibouchi, M.: Scale-invariant fully homomorphic encryption over the integers. In: International Workshop on Public Key Cryptography. pp. 311–328. Springer (2014)
22. Dobraunig, C., Eichlseder, M., Grassi, L., Lallemand, V., Leander, G., List, E., Mendel, F., Rechberger, C.: Rasta: a cipher with low anddepth and few ands per bit. In: Annual International Cryptology Conference. pp. 662–692. Springer (2018)

23. Dobraunig, C., Grassi, L., Helminger, L., Rechberger, C., Schofnegger, M., Walch, R.: Pasta: A case for hybrid homomorphic encryption. IACR Cryptol. ePrint Arch. **2021**, 731 (2021)
24. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2), 198–208 (1983). https://doi.org/10.1109/TIT.1983.1056650
25. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. **2012**, 144 (2012)
26. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first annual ACM symposium on Theory of computing. pp. 169–178 (2009)
27. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the aes circuit. In: Annual Cryptology Conference. pp. 850–867. Springer (2012)
28. Goldwasser, S., Gordon, S.D., Goyal, V., Jain, A., Katz, J., Liu, F.H., Sahai, A., Shi, E., Zhou, H.S.: Multi-input functional encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 578–602. Springer (2014)
29. Hebborn, P., Leander, G.: Dasta–alternative linear layer for rasta. IACR Transactions on Symmetric Cryptology pp. 46–86 (2020)
30. Kamara, S., Moataz, T., Ohrimenko, O.: Structured encryption and leakage suppression. In: Annual International Cryptology Conference. pp. 339–370. Springer (2018)
31. Katz, J., Lindell, Y.: Introduction to modern cryptography. CRC press (2020)
32. Méaux, P., Carlet, C., Journault, A., Standaert, F.X.: Improved filter permutators for efficient fhe: better instances and implementations. In: International Conference on Cryptology in India. pp. 68–91. Springer (2019)
33. Michalas, A., Bakas, A., Dang, H.V., Zalitko, A.: Abstract: Access control in searchable encryption with the use of attribute-based encryption and sgx. In: Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop. p. 183. CCSW'19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3338466.3358929, `https://doi.org/10.1145/3338466.3358929`
34. Michalas, A., Bakas, A., Dang, H.V., Zalitko, A.: Microscope: Enabling access control in searchable encryption with the use of attribute-based encryption and sgx. In: Askarov, A., Hansen, R.R., Rafnsson, W. (eds.) Secure IT Systems. pp. 254–270. Springer International Publishing, Cham (2019)
35. Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM workshop on Cloud computing security workshop. pp. 113–124 (2011)
36. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International conference on the theory and applications of cryptographic techniques. pp. 223–238. Springer (1999)
37. Rivest, R.L., Adleman, L., Dertouzos, M.L., et al.: On data banks and privacy homomorphisms. Foundations of secure computation **4**(11), 169–180 (1978)
38. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2), 120–126 (1978)
39. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) Advances in Cryptology – EUROCRYPT 2005. pp. 457–473. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)

40. Sans, E.D., Gay, R., Pointcheval, D.: Reading in the dark: Classifying encrypted digits with functional encryption. IACR Cryptology ePrint Archive **2018**, 206 (2018)
41. Microsoft SEAL (release 4.0). `https://github.com/Microsoft/SEAL` (Mar 2022), microsoft Research, Redmond, WA.