Faster Secure Arithmetic Computation Using Switchable Homomorphic Encryption

Hoon Wei Lim, Shruti Tople, Prateek Saxena, and Ee-Chien Chang

National University of Singapore

July 11, 2014

Abstract

Secure computation on encrypted data stored on untrusted clouds is an important goal. Existing secure arithmetic computation techniques, such as fully homomorphic encryption (FHE) and somewhat homomorphic encryption (SWH), have prohibitive performance and/or storage costs for the majority of practical applications. In this work, we investigate a new secure arithmetic computation primitive called *switchable homomorphic encryption* (or SHE) that securely switches between existing inexpensive partially homomorphic encryption techniques to evaluate arbitrary arithmetic circuits over integers. SHE is suited for use in a two-cloud model that is practical, but which makes stronger assumptions than the standard single-cloud server model. The security of our SHE solution relies on two non-colluding parties, in which security holds as long as one of them is honest. We benchmark SHE directly against existing secure arithmetic computation techniques—FHE and SWH—on real clouds (Amazon and Rackspace) using microbenchmarks involving fundamental operations utilized in many privacy-preserving computation applications. Experimentally, we find that SHE offers a new design point for computing on large data—it has reasonable ciphertext and key sizes, and is consistently faster by several (2–3) orders of magnitude compared to FHE and SWH on circuits involving long chain of multiplications. SHE exhibits slower performance only in certain cases, when batch (or parallel) homomorphic evaluation is possible, only against SWH schemes (which have limited expressiveness and potentially high ciphertext and key storage costs).

1 Introduction

Outsourcing of large-scale personal data from resource-constraint client devices, such as smartphones or laptops, to public clouds is prevalent from email to medical records management applications. In such services, users often want to utilize the cloud services for storage and computation without trusting it with the privacy of their data. Secure (or privacy-preserving) computation on encrypted cloud data has been a long-standing problem.

There have been considerable recent theoretical advances in cryptographic techniques for secure computation. In Gentry's seminal work [30], a primitive called fully homomorphic encryption (or FHE) was introduced and it offers ability to compute arbitrary circuits. Since then, continual improvement and progress has been made [53, 28, 27, 29]; however, these schemes are still prohibitively slow even for simple arithmetic operations. Another generic technique which offers faster circuit evaluation times is garbled circuits (or GC). GC-based computation techniques have witnessed considerable efficiency improvements [38, 51], especially for evaluation operations involving bitwise operations [37]. However, the size of garbled circuits grow linearly in the size of the data they compute on in general. Therefore, when computing on large-scale data, the costs of garbled circuit creation and upload to the cloud can be a prohibitive bottleneck in many practical applications. In contrast, partially homomorphic encryption (or PHE) [21, 54] that evaluates specific integer arithmetic operations has offered performance at least 2–3 orders of magnitude more efficient than FHE.

PHE techniques have been used in several practical research systems recently [57, 64]. Similarly, somewhat homomorphic encryption (or SWH) [53, 4] has received considerable attention for its better efficiency compared to FHE. However, it evaluates only low-degree polynomials on encrypted data, thereby having limited expressiveness.

In this paper, our goal is to develop an alternative primitive for secure arithmetic computation on encrypted data. Specifically, we aim to develop a primitive that supports general *expressiveness*, supporting arbitrary number of addition and multiplication over encrypted integers, with better *efficiency* for practical use. We aim to experimentally benchmark our primitive directly against existing primitives on the same benchmarks under the real cloud settings. In order to explore alternative primitives, we consider a *two-cloud* model that is practical to achieve, but requires a stronger trust assumptions than the traditional single server-setting. (We elaborate on this model and its practical feasibility in §2.1.)

Two-Cloud Model. The majority of the existing cryptographic techniques for secure computation are designed for a client-server model. Table 1 gives a summary of different variants of client-server models, ranging from one-client & one-server, *e.g.*, a naive download-and-compute approach[1] and FHE, to multi-client & multi-server, *e.g.*, general secure multi-party computation (MPC).

In this work, we consider a single-client and two non-colluding cloud model. We assume that the client outsources its data to two cloud providers (*e.g.*, Google, Microsoft, Amazon, and Rackspace), never revealing the decryption keys to any of them, but making some realistic assumptions about them not colluding. Such a model (relying on two non-colluding parties or multiple parties with a subset of them being honest) has also been used in other contexts, for example, initially in secure two-party computation (2PC) [22, 36, 14, 8, 62] and multi-party computation (MPC) [20, 12, 40], and more recently in private information retrieval (PIR) [2, 16, 4] and oblivious RAM (ORAM) [48, 61]. We consider a *non-interactive* model between a client and two servers, in the sense that the client is "offline", *i.e.*, outsourced computation on encrypted data can be performed without the client's participation. We further discuss the differences between previous works and ours in §6.

Model	$\leftrightarrow$	**Examples**
1 client & 1 server:		
– Download-and-compute	Yes	[5, 15]
– FHE	No	[30, 28, 29]
– SWH	No	[53, 4]
1 client & multi-servers:		
– Instantiation from MPC	No	[35, 62]
– *SHE (this work)*	*No*	
Multi-clients & 1 server:		
– Outsourced MPC	Yes	[22, 40]
– Outsourced MPC	No	[34]
– Threshold FHE	Yes	[30, 1]
– Multi-key FHE	No	[47]
Multi-clients & multi-servers:		
– Single-key MPC	Yes	[18, 12, 3]
– Multi-key MPC	No	[56]

Table 1: Models for secure computation. (Notation: $\leftrightarrow$ denotes interactivity between the client and the server during secure computation.)

Our Contributions. We demonstrate that there exist practical alternatives to FHE when working in the two-cloud model, which seems to be a promising, under-explored direction in the context of homomorphic

[1]Here a client downloads its encrypted data from online storage and locally performs the computation.

encryption. We realize this through a new cryptographic primitive called *switchable homomorphic encryption* (or SHE), which relies on two non-colluding clouds to jointly execute the client's computation on encrypted data without any of them having access to the decryption key. While we focus on two clouds in this paper, our approach can be generalized and extended to a n-cloud setting (*e.g.*, one server and $n-1$ proxies, where at least one of them is honest) in a straightforward manner.

Informally, our concept of SHE builds upon existing PHE techniques, such that it is possible to transform ciphertexts associated with additively homomorphic encryption (or ADD) to ciphertexts associated with multiplicatively homomorphic encryption (or MUL), and vice versa, and thus achieving fully homomorphic encryption. Our construction of SHE makes use of the well-studied Paillier ADD scheme [54] and a variant of the ElGamal MUL scheme [21] as building blocks. A major challenge in designing SHE based on the ElGamal MUL scheme is that a ciphertext leaks information whenever a plaintext value is zero. In our scheme, we make use of a simple, but novel, technique to address the zero-plaintext problem. Our technique enables an ElGamal ciphertext to be "operated" within a Paillier ciphertext, such that the encryption of zero is indistinguishable from the encryption of any other value. (We give an overview of our solution in §2.4.)

We implement and benchmark our SHE scheme based on two real cloud settings: Amazon and Rackspace. To compare against existing known primitives for secure arithmetic computation, we also implement and benchmark SWH based on the HElib library [45], FHE based on the Scarab Library (FHE) [46], and GC based on FastGC [37]. Our benchmarks are based on two fundamental operations in privacy-preserving computation: matrix multiplication and polynomial evaluation. These operations are commonly used in applications such as, set intersection [24], regular expression matching [55], private information retrieval [4], and so on. Our experimental results show that for encrypted polynomial evaluation, especially in applications with less data parallelism, our SHE scheme in the two-cloud model can be up to approximately $2,500$ times faster than SWH and FHE in the single-cloud setting; and roughly up to 35 times faster than FastGC. However, our scheme shows only slightly better performance as compared to SWH and GC approaches in the case of matrix multiplication. Thus, our scheme performs better than the existing solutions for applications with deeper circuit sizes, *i.e.*, many consecutive multiplications. When parallelization is used in SWH, our scheme is considerably slower. The dominant overhead comes from the communication cost between the two clouds. However, public key and ciphertext sizes in SWH can expand quickly (with polynomial or polylog overhead) as the evaluated circuit gets deeper. On the other hand, our approach enjoys constant sizes in both the public key and the ciphertext, and thus, has lower storage cost. In summary, our contributions are twofold:

- We demonstrate that indeed it is feasible to efficiently and securely switch between PHE schemes in the two-cloud model.
- We implement and evaluate secure arithmetic primitives on the same, real cloud platform (rarely done in previous works).

2 Problem & Approach

In this work, we are interested in looking for an alternative that is as expressive as existing FHE schemes, but with better efficiency.

2.1 Problem Definition

We consider the problem of secure computation of outsourced data generated by a client. In our setting, as illustrated in Figure 1, there exist two entities: nodes X and Y hosted by two separate cloud providers, respectively. We envision that these nodes can be hosted on two separate public clouds, such as Amazon and Rackspace connected by a high-speed network. These two clouds can be trusted not to overtly collude, while each being vulnerable to curious insiders. Encrypting data is still important to prevent curious insiders, such as database admins, who can be assumed to operate within the administrative domain of a single cloud

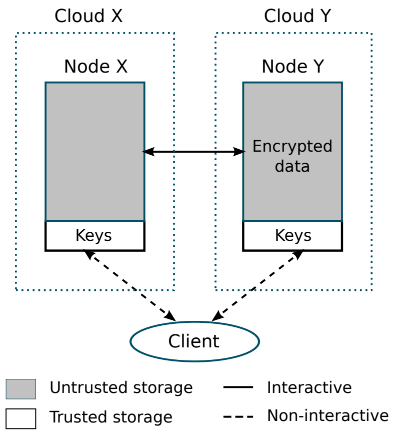

Figure 1: Secure outsourced computation via two clouds, X and Y.

provider and not able to collude outside of an organization. Our goal is then to protect data *confidentiality* against both the clouds. We do not consider non-malleability and verifiability of data computation.

A client interacts with both clouds, say via a web service, through a privacy-assisting browser extension. The client generates by itself some cryptographic keys, encrypts and stores its data on node Y. The client also shares some key material (different from the client's private keys) with X and Y, such that the key share at each side alone does not reveal any information about the encrypted data. Data computation is then performed within a virtual machine (VM) running on each cloud with the following constraints:

- Node Y is able to perform computation on the encrypted data on the client's behalf by interacting with node X over a high-speed network;
- The client can be offline during the computation, and upon completion, is able to retrieve the output from Y;
- The VMs run on an untrusted environment; that is, they are susceptible to snooping by curious insiders (*c.f.* [57]).
- Both X and Y are required to store only a small amount (less than a few KB) of cryptographic keys per user. All persistent ciphertext storage is accessible to service admins.

Adversarial Model. We assume that at least one of the two clouds is honest (*i.e.*, the other cloud can behave maliciously by deviating arbitrarily from our scheme). Security holds as long as at least one party is honest. If the server and cloud overtly collude by sharing the cryptographic keys, or if both act maliciously, no security can be guaranteed. We consider this to be a practical assumption. In the public cloud scenario, it is possible for well-reputed public cloud providers to overtly collude across organizations, but it risks their reputation. Our scheme can be extended to a n-cloud model where security holds as long as one party is honest, thereby making collusion difficult. Our trust assumptions are comparable to those used in multi-cloud PIR [48, 16] and ORAM [61].

We also assume that both nodes X and Y on the clouds are susceptible to malware installations by any outsider at the software stack, including the OS and the applications running on it. These may result in leakage of the client's encrypted data and key material, and any intermediate computation results stored on the VM. Hence, each cloud is assumed to further employ secure key distribution and storage mechanisms by leveraging TPM-based solutions, *e.g.*, KISS [69], to protect the key material. We assume that the client's key share never leaves a sealed storage component and it is accessible only via a trusted piece of software, *e.g.*, TrustVisor [49], and thus, decryption can be performed in an isolated, malware-free execution environment. A similar TPM-based trust model is also used in Autocrypt [64] for secure computation on encrypted web content in the single-server setting, where the client's (entire) decryption key is assumed to be securely stored on a hosting server. However, our model is stronger in the sense that the exposure of a key share does not reveal the plaintext corresponding to any encrypted data, unless the remaining key share is also known to the attacker.

Nevertheless, TPM-based solutions do not preclude all physical attacks, such as cold boot attacks. Hence, curious insiders, who have physical access to the node, may still learn the key material via a sophisticated hardware attack.[2] However, even if the insider gets the key material and has access to the encrypted data, it still cannot learn anything. It needs the other key share.

Our model also slightly differs from that used in many existing works on two-party computation (2PC), *e.g.*, [14, 35, 62, 67]. Their model allows the client's entire decryption key to be revealed to one of the two parties; the other party without the key has access to the client's encrypted data. That is, the model assumes that the encrypted data is never accessible to the party holding the decryption key. This seems to be a stronger assumption and may not be easy to realize in practice even with TPM-based solutions, since the ciphertext storage requirement could be large. Our model only assumes the protection of a small amount of key material on each node.

Note that it is not within our goal to prevent a malicious cloud provider from modifying ciphertexts stored on the cloud or launching a denial-of-service (DoS) attack to the other party. Also, we do not preserve the privacy of any function that is to be evaluated by the server, *i.e.*, it may infer some information about the client's encrypted input based on the function that it evaluates. Evaluating private functions [44, 42], verifying computation [25] and mitigating the impact of (DoS) attacks are orthogonal subjects of interest that are beyond the scope of this work.

2.2 A Simple but Inefficient Solution

Consider the following surprisingly simple known protocol [35, 62] that achieves the stated goal in the two-cloud model: store all the client's data encrypted with an additive homomorphic encryption (ADD) scheme, $\mathcal{E}^+$, on one cloud, Y, and store the corresponding decryption key on the other cloud, X. (Here, we require an ADD scheme, *e.g.*, the Paillier ADD scheme [54], that also allows multiplication of an encrypted value by a constant.) This way, Y can perform any arbitrary number of homomorphic additions on the encrypted data. To evaluate a multiplication operation on encrypted values $\mathcal{E}^+(a)$ and $\mathcal{E}^+(b)$, Y chooses two random values r_1 and r_2 and runs the following protocol with X:

$$\begin{aligned} Y \rightarrow X &: \mathcal{E}^+(a + r_1),\ \mathcal{E}^+(b + r_2) \\ X \rightarrow Y &: \mathcal{E}^+((a + r_1)(b + r_2)). \end{aligned}$$

In the first step, Y simply homomorphically blinds the target ciphertexts $\mathcal{E}^+(a)$ and $\mathcal{E}^+(b)$ with random values r_1 and r_2, respectively. X then, using the corresponding decryption key, recovers $(a+r_1)$ and $(b+r_2)$, performs the required multiplication, and returns the encryption of $(a + r_1)(b + r_2)$ to Y. Now what Y received from X is $\mathcal{E}^+(ab + ar_2 + br_1 + r_1r_2)$. However, clearly, Y is able to homomorphically remove all the "unneeded" terms (except ab) from the ciphertext, since it knows random values r_1 and r_2, and the ADD scheme allows multiplication of $\mathcal{E}^+(a)$ and $\mathcal{E}^+(b)$ with r_2 and r_1, respectively.

While the above simple solution works, it has two issues. First, X has the decryption key, and hence, if given access to any ciphertexts generated by the client (*e.g.*, those stored on Y), it would be able to recover the corresponding plaintexts. However, this is easy to fix. We can simply split the client's decryption key and give each party only one part of the key. This way, neither party can individually decrypt any ciphertext. Alternatively, we can "double-encrypt" the ciphertexts stored on Y, such that Y can decrypt the "outer" layer of encryption and X can decrypt the "inner" layer. (The outer-layer protects against curious insiders.)

The second issue is that the solution has poor efficiency. Each multiplication operation requires a "joint computation" between X and Y (which includes one encryption, two decryptions and roughly five modular

[2]There exist mechanisms for preventing hardware attacks, for example, by using a secure processor substrate and through memory cloaking. See Bastion [10] and SecureMe [11] for further details.

exponentiations if the Paillier ADD scheme is used). Therefore, the solution does not scale for applications requiring a large number of multiplications.

2.3 Challenges

Henceforth, we focus on using two partially homomorphic encryption (PHE) schemes (supporting addition and multiplication, respectively) as the building blocks in designing a practical FHE solution. There are numerous challenges in realizing this:

- **Compatibility**: To convert arbitrarily between ADD and MUL ciphertexts, we need a transformation function that is bijective, *i.e.*, one-to-one mapping between ADD and MUL ciphertexts. Moreover, such transformation function must preserve the homomorphism property of the ciphertexts. A homomorphic and bijective function that maps elements from one group to elements of another group is known as a *group isomorphism*. This may not be easy to realize, for example, if we work with the well-known Paillier ADD and the standard ElGamal MUL schemes [21]. The former uses an additive group with composite modulus and the later works in a multiplicative group with prime order. There seems to be no known efficient inverse of an isomorphic function that maps elements from a multiplicative group to those in an additive group.

- **Zero-Plaintext**: A fundamental, inevitable property of known MUL schemes, such as ElGamal and RSA, is that encrypting zero results a zero in the ciphertext. One natural way to address such issue is to encode a message with some "noise" before encrypting it. However, this typically destroys the multiplicative homomorphism of the scheme, for example in the case of RSA-OAEP [52]. Moreover, removing the accumulated noise in an encrypted domain after a series of computations may not be trivial at all.

- **Security**: The cloud providers performing computation on a client's encrypted data must not learn anything about the underlying plaintexts and the corresponding decryption keys. It is also critical to preserve the data confidentiality invariant before, during and after transformation of any ADD and MUL ciphertexts. We assume SHE switching logic runs in a TPM-protected environment.

- **Efficiency**: The overall computational and communication overhead incurred during transformation of ciphertexts should be reasonable for most real world application scenarios.

Why not instantiate with 2PC? To solve our outlined problem, one may consider using an existing 2PC protocol between the proxy and the server [37]. First, the communication complexity of generic 2PC is at least linear to the size of the garbled circuit, which in turn, is proportional to the size of the data (input wires). In contrast, our approach has communication complexity that is independent of the size of the encrypted data, but instead is linear to the number of ciphertext transformations between ADD and MUL schemes in the circuit. Second, many efficient solutions to 2PC, such as GC, require the circuit to be encrypted in advance by the client, unlike in our solution where the computed function can be dynamically picked by a third party. Finally, garbled circuits typically cannot be reused in order to preserve the secrecy of the relevant input and function. The client must reconstruct a new garbled circuit (proportional to the size of the dataset) and upload it to the server each time the client wants to evaluate a function.[3] This renders GC to be unsuitable for privacy-preserving outsourced computation, particularly for large datasets. Although circuit construction can be outsourced to the server, the computational and storage costs incurred could still be high for large datasets and large numbers of clients. We experimentally benchmark the efficacy of one of the recent GC solutions using its off-the-shelf implementation and compare it against our scheme [37].

[3]While there exist recent, promising results on reusable garbled circuits using function encryption by Goldwasser *et al.* [32], no practical implementation has been reported thus far.

2.4 Solution Overview

For ease of exposition, let us now call node X the *proxy* and node Y the *server*. In our solution, the client encrypts its data using either the ADD or the MUL scheme and stores the resulting ciphertexts on the server.

In our solution, the client stores integer or ASCII data encrypted under the MUL scheme. Circuits are in DNF form. They are constructed once and uploaded to the server, and can be reused on any subset of the data repeatedly. The client also provides the corresponding secret key shares to both the proxy and the server. Using our SHE schemes, the server can evaluate addition and multiplication operations on the encrypted data, and when necessary, switch between ADD and MUL ciphertexts. We now give some technical highlights of our approach.

Secret Key Shares. We split a user-chosen private (decryption) key into two parts. The proxy and the server each is given one part of the key, which in turn, can be used only to jointly decrypt a ciphertext during transformation of a ciphertext from ADD to MUL (and vice versa).

SHE. We work with a variant of the ElGamal MUL scheme that uses a composite modulus, such that it is "compatible" with the Paillier ADD scheme. However, a fundamental limitation with the ElGamal MUL scheme is that it does not handle a zero-plaintext.[4] Our SHE scheme addresses such an issue. We perform addition and multiplication of ElGamal ciphertexts in a homomorphic manner within an encrypted domain associated with the Paillier ADD scheme. Since the encryption of zero under the Paillier ADD scheme is indistinguishable from the encryption of a random plaintext under the same scheme, the ElGamal ciphertext will not leak any information even if the corresponding plaintext becomes zero. However, the catch is that all the client's initial input has to be encrypted with the ElGamal MUL scheme and the input must not contain zeros. We do not see this as a glitch. As pointed out by Jakobsson and Juels [39], we can represent an ElGamal encryption of zero with two ElGamal encryptions of two random values n_1 and n_2, respectively, such that $n_1 - n_2 = 0$. Using our technique, therefore, we can perform homomorphic addition or multiplication over any two ElGamal ciphertexts, or a Paillier ciphertext and an ElGamal ciphertext. It turns out that Boolean combinatorial circuits can be represented in DNFs (sum of products) [58]. This implies that we can evaluate arbitrary size disjuncts and sum up an arbitrary number of them. For certain applications, the DNF size may grow larger, but our evaluation in §5 explains that many interesting applications directly build on matrix multiplication and polynomial-arithmetic operations. Our scheme can also be used for statistical analysis applications such as, mean, covariance, and linear regression. These are well-suited for DNF.

Security. The security of our SHE solution is based on the assumptions that: (i) the underlying Paillier ADD and ElGamal MUL schemes are also secure, and (ii) the proxy and the server do not collude. We analyze the security of our SHE scheme under the scenario that either the proxy or the server is malicious, while the other remains honest. Our security proof is by reduction to the security of the Paillier ADD and ElGamal MUL schemes and the hardness of the Decisional Diffie-Hellman (DDH) problem. We show that our SHE scheme is CPA-secure [41] (in terms of indistinguishability of encryptions).

3 SHE with Two Clouds

In this section, we first give a definition of switchable homomorphic encryption (SHE). We also briefly recall two partially homomorphic encryption (PHE) schemes. We then present our SHE construction built upon the PHE schemes.

Notation. Henceforth, we use the operator $\circ \in \{+, \times\}$ to indicate if a parameter, an algorithm or a scheme is associated with either additive ($+$) or multiplicative ($\times$) homomorphic encryption, *i.e.*, ADD or MUL.

[4]In fact, technically speaking, any message $m \notin \mathbb{Z}_N^*$, for a composite N, encrypted under the ElGamal scheme leaks some information about m.

3.1 Definition

A switchable homomorphic encryption (SHE) scheme in the two-cloud setting comprises six algorithms defined as follows:

- $\mathsf{KeyGen}(1^n)$: On input a security parameter 1^n, the algorithm outputs a public and private key pair (pk°, sk°) for each operator $\circ \in \{+, \times\}$.
- $\mathsf{Enc}(pk^\circ, m)$: The algorithm takes as input a public key pk° and a message m. It outputs a ciphertext c°.
- $\mathsf{Dec}(sk^\circ, c^\circ)$: On input a private key sk° and a ciphertext c°, the algorithm outputs a message m.
- $\mathsf{KeyShaGen}(sk^\circ)$: The algorithm takes as input a private key sk° and outputs a pair of secret key shares (k_0°, k_1°). (Here secret key shares are key material used to *blindly* decrypt[5] a ciphertext.)
- $\mathsf{AddToMul}(k_0^+, k_1^+, c^+, pk^+, pk^\times)$: The algorithm takes as input the secret key shares (k_0^+, k_1^+) and an ADD ciphertext c^+. It outputs the corresponding MUL ciphertext $c^\times$ under public key $pk^\times$.
- $\mathsf{MulToAdd}(k_0^\times, k_1^\times, c^\times, pk^\times, pk^+)$: On input the secret key shares $(k_0^\times, k_1^\times)$ and an MUL ciphertext $c^\times$, the algorithm outputs the corresponding ADD ciphertext $c^\times$ under public key pk^+.

The first four algorithms described above are run by the client. The AddToMul and MulToAdd algorithms can be loosely regarded as isomorphic transformation functions (discussed in §2.3), which are performed by the proxy and the server to transform an ADD ciphertext to a MUL ciphertext, and vice versa.

3.2 PHE as Building Blocks

We work with the Paillier ADD scheme [54] and a variant of the ElGamal MUL scheme [21]. We choose to work with an ElGamal encryption scheme that uses a composite modulus [50], *i.e.*, $N = pq$ where p, q are large primes. Both the ADD and MUL schemes share the same modulus. (Note that this does not pose any security concern since the modulus is public information and a public-private key pair used in scheme has different structure and is independently chosen from that of the other scheme.) This is essential so that both schemes work on the same group structure, and thus, allowing ciphertexts to be transformed in a natural way (*i.e.*, such that bijective mapping exists.)

Paillier ADD. The Paillier encryption system, denoted by $\mathcal{E}^+$, is defined as follows:

- $\mathsf{KeyGen}(1^n, +)$: On input a security parameter 1^n, the algorithm outputs (N, p, q), where $N = pq$, and p and q are n-bit primes. The public key is $pk^+ := N$ and the corresponding private key is $sk^+ := \langle N, \phi(N)\rangle$, where $\phi(N) = (p-1)(q-1)$.
- $\mathsf{Enc}(pk^+, m)$: The algorithm takes as input a public key N and a message $m \in \mathbb{Z}_N$. It chooses a random $r \in \mathbb{Z}_N^*$ and outputs the ciphertext c^+ as
$$(1+N)^m \cdot r^N \bmod N^2.$$
- $\mathsf{Dec}(sk^+, c^+)$: On input a private key $\langle N, \phi(N)\rangle$ and a ciphertext c^+, the algorithm outputs the message m as
$$\frac{((c^+)^{\phi(N)} \bmod N^2) - 1}{N} \cdot \phi(N)^{-1} \bmod N.$$
(Recall that since $|\mathbb{Z}_{N^2}^*| = \phi(N^2) = N\phi(N)$, we have $r^{N\phi(N)} = 1 \bmod N^2$ for any $r \in \mathbb{Z}_N^*$.)

Clearly, the scheme is additively homomorphic.

[5] We use the term "blind decryption" to denote a decryption that results in a "partially" decrypted ciphertext or a blinded plaintext.

ElGamal MUL. The ElGamal encryption system, denoted by $\mathcal{E}^{\times}$, is defined as follows:

- $\mathsf{KeyGen}(1^n, \times)$: On input a security parameter 1^n, the algorithm chooses two large, safe primes p and q (as specified in [50]) and set $N = pq$. It also chooses a base $g = 16$ and a random odd[6] number x, and sets $h := g^x \bmod N$. The public key is then $pk^{\times} := \langle N, g, h \rangle$ and the corresponding private key is $sk^{\times} := \langle N, g, x \rangle$.
- $\mathsf{Enc}(pk^{\times}, m)$: The algorithm takes as input a public key $\langle N, g, h \rangle$ and a message $m \in \mathbb{Z}_N^*$. It chooses a random $r \in \mathbb{Z}_N^*$ and outputs the ciphertext $c^{\times} := \langle c_1^{\times}, c_2^{\times} \rangle$ as
$$\langle mh^r,\ g^r \bmod N \rangle.$$
- $\mathsf{Dec}(sk^{\times}, c^{\times})$: On input a private key $\langle N, g, x \rangle$ and a ciphertext $c^{\times}$, the algorithm outputs the message m as
$$(c_1^{\times})/(c_2^{\times})^x \bmod N.$$

It is easy to see that multiplicative homomorphism holds.
We discuss the security of the Paillier ADD and ElGamal MUL schemes in §4.

3.3 Our Scheme

We avoid the inherent zero-plaintext problem in the ElGamal scheme. Moreover, our scheme enables computation of arbitrary sum-of-subset expressions (and more) over encrypted integers, and thus is expressive as FHE. It is sufficient to show that such a scheme, in theory, evaluates arbitrary boolean combinatorial circuits, since any circuit can be represented in its DNF form (sum-of-subsets).

Intuition. Since the Paillier ADD scheme allows encryption of a zero-plaintext, but the ElGamal MUL scheme does not, one natural question to ask is: *can we manipulate ElGamal ciphertexts within the Paillier ciphertext space?* It turns out that we can, since both ElGamal and Paillier encryptions can operate within the group $\mathbb{Z}_N$. One can simply treat an ElGamal ciphertext, which is an element of the group $\mathbb{Z}_N^*$, as a "plaintext" in the Paillier encryption domain, where the message space is $\mathbb{Z}_N$. By slightly abusing the notation $\mathcal{E}^+$ and $\mathcal{E}^{\times}$, this can be visualized as
$$\mathcal{E}^+(\mathcal{E}^{\times}(m)) = (1+N)^{\mathcal{E}^{\times}(m)}.r^N$$
for a random r and where $\mathcal{E}^{\times}(m) = mh^{r_0} \bmod N$. (Note that strictly speaking, the above expression is incorrect since an ElGamal ciphertext has two components $\langle mh^{r_0},\ g^{r_0} \bmod N \rangle$. However, for exposition purpose, we simply assume for now that $\mathcal{E}^{\times}(m)$ refers to the first ciphertext component mh^{r_0}.) We observe that, interestingly, even if $\mathcal{E}^{\times}(m)$ is "encrypted" (as a Paillier ciphertext), the Paillier ADD scheme has the property that allows anyone without knowledge of the Paillier decryption key to manipulate the ElGamal ciphertext. That is, one can "decrypt" $\mathcal{E}^{\times}(m)$ by raising the encrypted ElGamal ciphertext to the power of $(h^{r_0})^{-1}$ (with knowledge of the ElGamal decryption key), or "re-encrypt" m by raising $\mathcal{E}^+(m)$ to the power of h^{r_1} for a randomly chosen r_1. This way, we can perform addition and multiplication on ElGamal and Paillier ciphertexts. Henceforth, when we say "an encrypted MUL ciphertext", we refer to encryption of the form $\mathcal{E}^+(\mathcal{E}^{\times}(m))$.

Take for example, given $\mathcal{E}^+(m)$ and $\mathcal{E}^{\times}(m') = mh^{r'}$, one can "homomorphically multiply" the ciphertexts to get $\mathcal{E}^+(mm')$. This can be done simply as follows:

1. raise $\mathcal{E}^+(m)$ to the power of $\mathcal{E}^{\times}(m')$ such that we have $\mathcal{E}^+(mm'h^{r'})$
2. remove the term $h^{r'}$ from $\mathcal{E}^+(mm'h^{r'})$ by raising it to the power of $(h^{r'})^{-1}$, assuming the decryption key of $\mathcal{E}^{\times}(mm')$ is known.

	$\mathcal{E}^+$	$\mathcal{E}^\times$
$\mathcal{E}^+$	$+$	$+, \times$
$\mathcal{E}^\times$	$+, \times$	$+, \times$

Table 2: Combination of ciphertexts and supported operations.

Table 2 summarizes possible combinations of ciphertexts and the corresponding supported arithmetic operations. Note that our scheme supports homomorphic addition and multiplication of any combination of $\mathcal{E}^+$and $\mathcal{E}^\times$ciphertexts, except that it does not allow homomorphic multiplication of two $\mathcal{E}^+$ciphertexts. Circuits prepared in DNF form will have no need to switch back from ADD to MUL. In DNF form, multiplications are performed first and followed by additions. If DNF form is not suitable for whatever reason, we can always combine SHE with the basic interactive multiplication approach described in § 2.2.

Construction. We are now ready to present our SHE scheme, which is based on the Paillier ADD scheme, $\mathcal{E}^+$, and the ElGamal MUL scheme, $\mathcal{E}^\times$described in §3.2. We note that since we operate MUL ciphertexts in an encrypted domain, the AddToMul algorithm is used to transform a message encrypted under $\mathcal{E}^+$ into an encrypted MUL ciphertext under $\mathcal{E}^+$; analogously, the MulToAdd algorithm transforms an encrypted MUL ciphertext under $\mathcal{E}^+$ into a plaintext encrypted under $\mathcal{E}^+$. Our SHE scheme is specified as follows:

- KeyGen(1^n): On input a security parameter 1^n, the algorithm first chooses (N, p, q), where $N = pq$, and p and q are n-bit primes. It then chooses two random odd numbers $x_0, x_1 \in \mathbb{Z}_N^*$, where $|x_0| \approx |x_1| < \frac{1}{2}|N|$, and the generator $g = 16$. It sets $x = x_0 x_1$ and $h = g^x$, and outputs the following public-private key pairs

$$(pk^+, sk^+) := (N,\ \langle N, \phi(N), p, q\rangle)$$
$$(pk^\times, sk^\times) := (\langle N, g, h\rangle,\ \langle N, g, x_0, x_1\rangle).$$

- Enc(pk°, m): For $\circ := +$, the algorithm runs the Enc algorithm of $\mathcal{E}^+$; otherwise if $\circ := \times$, it runs the Enc algorithm of $\mathcal{E}^\times$.
- Dec(sk°, c°): Similarly, for $\circ := +$, the algorithm runs the Dec algorithm of $\mathcal{E}^+$; otherwise if $\circ := \times$, it runs the Dec algorithm of $\mathcal{E}^\times$.
- KeyShaGen(sk^+): The algorithm sets both the secret key shares k_0^+ and k_1^+ to null (since no decryption of $\mathcal{E}^+(m)$ ciphertexts is required in our scheme).
- KeyShaGen$(sk^\times)$: The algorithm sets the secret key shares to be $k_0^\times := x_0$ and $k_1^\times := x_1$.
- AddToMul$(c^+, pk^\times)$: This algorithm is run locally by the server. Given an ADD ciphertext of the form

$$\mathcal{E}^+(m) := (1+N)^m \cdot (r')^N \bmod N^2$$

  and the MUL public key $pk^\times := \langle N, g, h\rangle$, the algorithm chooses a random $r \in \mathbb{Z}_N^*$ and outputs an encrypted MUL ciphertext

$$\mathcal{E}^+(\mathcal{E}^\times(m)) := \langle (1+N)^{mh^r} \cdot (r')^{Nh^r} \bmod N^2,\ g^r\rangle.$$

- MulToAdd$(c^+, k_0^\times, k_1^\times)$: This algorithm is jointly run by the server and the proxy. On input an encrypted MUL ciphertext of the form

$$\mathcal{E}^+(\mathcal{E}^\times(m)) := \langle (1+N)^{mh^r} \cdot (r')^{Nh^r} \bmod N^2,\ g^r\rangle,$$

[6]With an odd exponent x, it is almost certainly true that we will have $\gcd(x, \phi(N)) = 1$.

the server chooses a random $s \in \mathbb{Z}_N^*$, computes

$$c' := (g^{r+s})^{k_1^\times}, \ \ R := g^s$$

and forwards $\langle c^+, c', R \rangle$ to the proxy. (Here the random value s is used to blind the ElGamal ciphertext component g^r. This is to prevent the proxy from learning the corresponding h^r value using its key share.)

The proxy then, using its key share $k_0^\times$, computes $(c')^{k_0^\times} = h^{r+s}$ and its inverse $(h^{r+s})^{-1}$. It also computes and returns

$$\begin{aligned} c'' &:= ((1+N)^{mh^r} \cdot (r')^{Nh^r})^{(h^{r+s})^{-1}} \bmod N^2 \\ &:= (1+N)^{mh^{-s}} \cdot (r')^{Nh^{-s}} \bmod N^2 \\ &:= \mathcal{E}^+(mh^{-s}) \end{aligned}$$

and $R' = R^{k_0^\times}$ to the server.

Finally, the server computes $(R')^{k_1^\times} = h^s$ and recovers the corresponding ADD ciphertext $\mathcal{E}^+(m)$ by homomorphically removing h^{-s} from c''.

We analyze the security of the above construction in Section 4.

3.3.1 Discussion

Assumptions. In our SHE scheme, the client is required to generate and submit *only* ElGamal ciphertexts corresponding to *non-zero* plaintexts. As explained in Section 2.4, if encryption of zero is required in the computation, the client can construct a circuit that represents zero in the form of

$$\mathsf{MulToAdd}(\mathcal{E}^+(\mathcal{E}^\times(n_1))) \times \mathsf{MulToAdd}(\mathcal{E}^+(\mathcal{E}^\times(n_2)))^{-1} = \mathcal{E}^+(0)$$

for random n_1, n_2 and where $n_1 - n_2 = 0$. This prevents encryption of a zero-plaintext under $\mathcal{E}^\times$ at any point of the computation.

Moreover, we assume that: (i) at least either the proxy or the server does not reveal to any party its secret key shares and any local states, *e.g.*, random value s used in a ciphertext transformation; (ii) at least either the proxy or the server obediently executes the specified algorithms in the scheme.

Homomorphism. With the SHE scheme, it is trivial to evaluate homomorphic multiplication on two encrypted messages $\mathcal{E}^\times(m_1)$ and $\mathcal{E}^\times(m_2)$. To output the result in $\mathcal{E}^+$, the server simply encrypts $\mathcal{E}^\times(m_1 m_2)$ under $\mathcal{E}^+$ and runs $\mathsf{MulToAdd}(\mathcal{E}^+(\mathcal{E}^\times(m_1 m_2)))$. To homomorphically add $\mathcal{E}^\times(m_1)$ to $\mathcal{E}^+(m_3)$, the server first runs the $\mathsf{MulToAdd}(\mathcal{E}^+(\mathcal{E}^\times(m_1))) = \mathcal{E}^+(m_1)$ and then adds the result to $\mathcal{E}^+(m_3)$ to obtain $\mathcal{E}^+(m_1 + m_3)$. The steps involved in performing multiplication between $\mathcal{E}^\times(m_1)$ and $\mathcal{E}^+(m_3)$ has been sketched before. Lastly, homomorphic addition of two Paillier encrypted messages is done in the usual way.

Extending to n-cloud. Our scheme can be extending to work in the n-cloud setting. For example, instead of relying on one proxy, we can use $n-1$ non-colluding proxies to jointly perform the $\mathsf{MulToAdd}$ algorithm with the server. Each proxy is given a secret key share $k_{0,i}^\times$ for $i \in \{1, \ldots, n-1\}$ and the server holds the key share $k_{0,n}^\times$ such that $x = k_{0,1}^\times \cdots k_{0,n}^\times$. This way, the confidentiality of the client's data is protected so long as at least one of the n parties is honest.

4 Security Analysis

Our security goal is to protect data confidentiality, that is, ensuring no meaningful information is leaked through homomorphically encrypted data. This must hold before, during and after a ciphertext associated with one arithmetic operation is transformed to one that supports another arithmetic operation. However, as with any existing homomorphic encryption schemes, our schemes do not guarantee data integrity or correctness.[7] Moreover, our schemes do not prevent information leakage through the function to which the client's input is fed.

4.1 CPA Model

We work with the standard chosen-plaintext attack (CPA) model [41]. As usual, security is defined through simulating an *indistinguishability* security game between an adversary $\mathcal{A}$ (attack algorithm) and a challenger:

1. $\mathcal{A}$ plays the role of either a malicious server or a malicious proxy (but not both).
2. At the start of the game, $\mathcal{A}$ is given the public keys and secret key shares that would have been required by a real server or proxy to perform the Enc algorithms associated with $\mathcal{E}^+$ and $\mathcal{E}^\times$, and the AddToMul and MulToAdd algorithms for transformation of ciphertexts.
3. During the game, $\mathcal{A}$ is given access to the following oracles:
    - Enc oracle—capability to choose arbitrary plaintexts to be encrypted and obtain the corresponding ciphertexts;
    - AddToMul oracle—capability to transform $\mathcal{E}^+(m)$ to $\mathcal{E}^+(\mathcal{E}^\times(m))$;
    - MulToAdd oracle—capability to transform $\mathcal{E}^+(\mathcal{E}^\times(m))$ to $\mathcal{E}^+(m)$.
4. $\mathcal{A}$ is also required to choose two messages m_0, m_1 with the restriction that $|m_0| = |m_1|$. It also chooses what type of transformation (AddToMul or MulToAdd) on which it wishes to be challenged.
5. During the challenge phase, and given m_0, m_1, the challenger performs the following steps:
    - pick a random bit $b \in \{0, 1\}$;
    - if $\mathcal{A}$ wishes to be challenged on AddToMul, generate $\mathcal{E}^+(m_b)$ and run the AddToMul algorithm;
    - otherwise, generate $\mathcal{E}^+(\mathcal{E}^\times(m_b))$ and jointly run the MulToAdd algorithm with $\mathcal{A}$;
    - output both the ciphertexts before and after transformation as the challenge ciphertext; $\mathcal{A}$ is also given a copy of $\mathcal{E}^\times(m_b)$ (since all data is initially encrypted in such form in our SHE scheme).
6. $\mathcal{A}$ continues to have access to the aforementioned oracles.
7. At the end of the game, $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ for b. $\mathcal{A}$ wins the game if $b = b'$.

Note that since $\mathcal{A}$ has access to $\mathcal{E}^\times(m_b)$, we impose the restriction of $m_0 \neq m_1 \neq 0$ in the above game. (In the actual scheme, as explained, this requirement can be easily met.) Moreover, for simplicity, we consider transformation of a single ciphertext that is directly computed from one of the messages chosen by the adversary. However, it is straightforward to extend the above CPA security game to allow the adversary to submit two sets of messages $M_0 = \{m_0^1, m_0^2, \ldots, m_0^t\}$ and $M_1 = \{m_1^1, m_1^2, \ldots, m_1^t\}$, with the restriction that $|m_0^i| = |m_1^i|$ for all $1 \leq i \leq t$.

We say that the SHE scheme is CPA-secure if any probabilistic polynomial-time algorithm $\mathcal{A}$ attacking the scheme has only *negligible advantage* of winning the CPA security game.

[7]Hence, we work in a model which is comparable to a semi-honest model typically used in the MPC setting [31], but additionally, we allow one of the two parties to behave maliciously.

4.2 Security of PHE

The Paillier MUL scheme has been proven to have indistinguishable ciphertexts under the CPA model. The security proof is based on the assumption that the decisional composite residuosity problem is hard, relative to the hardness of factoring a composite number N that is the product of two primes [54].

On the other hand, McCurley [50] showed that, with careful selection of parameters, ElGamal encryption with a composite modulus can be secure against any adversary who could break Diffie-Hellman key exchange, or could factor the modulus, but not both. While there is no known proof of CPA-security, it is conjectured that the ElGamal MUL scheme over a composite modulus is semantically secure [23, 9].

4.3 Security of SHE

We now prove the security of our SHE scheme under the CPA model defined in § 4.1.

Theorem 1. *The SHE scheme is secure in the CPA model against any probabilistic polynomial-time adversary if the underlying ADD and MUL schemes are CPA-secure, the Decisional Diffie-Hellman (DDH) problem is hard, and provided at least the proxy or the server is honest.*

Proof of Theorem 1. Let ϵ_{she} be the advantage of $\mathcal{A}$ in breaking the SHE scheme. We simulate a CPA security game and consider the following cases.

Case 1 (Honest Server): If $\mathcal{A}$ is the proxy, it is given the public key $pk^{\times}$ and the key share $k_0^{\times}$ for the MUL scheme. Moreover, if $\mathcal{A}$ chooses to attack the AddToMul algorithm, it has access to

$$\langle \mathcal{E}^{\times}(m_b), \mathcal{E}^{+}(m_b), \mathcal{E}^{+}(\mathcal{E}^{\times}(m_b)) \rangle$$

as the challenge ciphertext. However, by the semantic security of the ADD scheme, $\mathcal{A}$ learns no information about m_b from $\mathcal{E}^{+}(m_b)$ and $\mathcal{E}^{+}(\mathcal{E}^{\times}(m_b))$ without the corresponding private key sk^{+}. Hence, it is sufficient to analyze if any information about m_b can be deduced from

$$\langle \mathcal{E}^{\times}(m_b) = (m_b h^r, g^r),\ pk^{\times} = (N, g, h),\ k_0^{\times} = x_0 \rangle.$$

Lemma 1. *Given $\langle g, h, x_0, g^r, m_b h^r \rangle$, any probabilistic polynomial-time adversary $\mathcal{A}$ has negligible advantage $\epsilon_{\mathcal{A}_1}$ in distinguishing $\mathcal{E}^{\times}(m_0)$ from $\mathcal{E}^{\times}(m_1)$, assuming that the DDH problem is hard.*

Proof of Lemma 1. By definition of the CPA-secure MUL scheme, $m_0 h^r$ is indistinguishable from $m_1 h^r$. This is because $m_b h^r$ contains no information about m_b for a randomly chosen r (*i.e.*, the distribution of the former is independent of that of the latter), since h^r is indistinguishable from a random group element under the DDH assumption. However, given knowledge of the key share $k_0^{\times} = x_0$, $\mathcal{A}$ is able to compute the values $h^{1/x_0} = (g^x)^{1/x_0} = g^{x_1}$ and $(m_b h^r)^{1/x_0} = m_b^{1/x_0} g^{r x_1}$, where x_1 is the other unknown key share. (Note that here $(m_b)^{1/x_0}$ is known to $\mathcal{A}$.) Hence, we need to show that given (g, g^r, g^{x_1}), $\mathcal{A}$ is unable to distinguish $g^{r x_1}$ from a random group element with non-negligible advantage, since the capability in doing so implies that $\mathcal{A}$ is able to distinguish $m_0^{1/x_0} g^{r x_1}$ from $m_1^{1/x_0} g^{r x_1}$ with non-negligible advantage.

Let consider $\tilde{m}_b = m_b^{1/x_0}$ as an encoded message of m_b. It is then straightforward to prove that $\tilde{m}_b g^{r x_1}$ leaks no information about $\tilde{m}_b$ under the standard DDH assumption. Suppose there exists an adversary $\mathcal{A}$ which can break the CPA security of the MUL scheme with advantage at least $\epsilon_{\mathcal{A}_1}$. We build an algorithm $\mathcal{B}$ which can solve the DDH problem. Let $\mathcal{B}$ take as input a DDH challenge (g, g^v, g^w, T) for random $v, w \in \mathbb{Z}_N^*$ and where T is either g^{vw} or a random group element of $\mathbb{Z}_N^*$. $\mathcal{B}$ then proceeds as follows. It picks a random x_0 and sets $\langle g, h = g^w, N \rangle$ as the public key of the MUL scheme and x_0 as the secret key share $k_0^{\times}$ for adversary $\mathcal{A}$. (This implicitly sets $g^w = g^{x_1}$.) $\mathcal{B}$ also provides Enc oracle access to $\mathcal{A}$ in the standard way.

During the challenge phase, $\mathcal{A}$ submits two messages m_0, m_1 of equal length. $\mathcal{B}$ chooses a random bit $b \in \{0, 1\}$ and returns $(m_b^{1/x_0} T, g^v)$ as the challenge MUL ciphertext, which by definition, is a valid MUL ciphertext since

$$(m_b^{1/x_0} T, g^v) = (\tilde{m}_b T, g^v)$$

where $\tilde{m}_b$ can be treated as an encoded message based on x_0, and $g^w = g^{x_1}$ is given to $\mathcal{A}$ as the public key.

$\mathcal{A}$ finally outputs a guess b' of b. If $b' = b$, $\mathcal{B}$ outputs 0 (indicating that $T = g^{vw}$); otherwise, it outputs 1 (indicating that T is a random element of $\mathbb{Z}_N^*$). We see that if T is a random group element, then $\mathcal{B}$ has probability $\frac{1}{2}$ in outputting $b = 0$. On the other hand, if $T = g^{vw}$, $\mathcal{B}$ has advantage at least $\epsilon_{\mathcal{A}_1}$ in solving the DDH problem. □

On the other hand, if $\mathcal{A}$ wishes to attack the MulToAdd algorithm, it has access to $\langle \mathcal{E}^\times(m_b), \mathcal{E}^+(\mathcal{E}^\times(m_b)), c', R\rangle$ instead during the challenge phase. Similarly, we turn our attention to analyzing what information can be deduced from non-ADD ciphertext components

$$\langle \mathcal{E}^\times(m_b) = (m_b h^r, g^r),\ c' = (g^{r+s})^{x_1},\ R = g^s,\ pk^\times = (N, g, h),\ k_0^\times = x_0\rangle.$$

From Lemma 1, we prove that the adversary does not learn anything about m_b from $\langle \mathcal{E}^\times(m_b), pk^\times, k_0^\times\rangle$. Hence, it is sufficient to show that no information about m_b is leaked through c' and R.

Lemma 2. *Given $\langle g, h, x_0, g^s, (g^{r+s})^{x_1}, m_b h^r\rangle$, any probabilistic polynomial-time adversary $\mathcal{A}$ has negligible advantage $\epsilon_{\mathcal{A}_2}$ in distinguishing $\mathcal{E}^\times(m_0)$ from $\mathcal{E}^\times(m_1)$, assuming that the DDH problem is hard.*

Proof of Lemma 2. Given $(g^{r+s})^{x_1}$ and x_0, $\mathcal{A}$ is able to compute $(g^{r+s})^{x_1 x_0} = (g^x)^{r+s} = h^{r+s}$, from which it can also compute $h^{r+s}/(m_b h^r) = (1/m_b)h^s$. Hence, we need to show that it is infeasible for the adversary to distinguish $(1/m_0)h^s$ from $(1/m_1)h^s$ with non-negligible advantage.

We can prove that $(1/m_b)h^s$ leaks no information about m_b using the same technique for proving Lemma 1. That is, we run an adversary $\mathcal{A}$ with advantage at least $\epsilon_{\mathcal{A}_2}$ against the MUL scheme as a subroutine within an algorithm $\mathcal{B}$, which is used to solve the DDH problem. Given a DDH challenge (g, g^v, g^w, T), we set $g^w = g^{x_1}$ and embed g^v in the challenge ciphertext of the form

$$((1/m_b)^{1/x_0}T, g^v) = (\tilde{m}_b T, g^v)$$

such that T is either $g^{vw} = g^{sx_1}$ or a random element of $\mathbb{Z}_N^*$. Following the same reasoning, we can show that $\mathcal{B}$ has advantage at least $\epsilon_{\mathcal{A}_2}$ in solving the DDH problem. □

Case 2 (Honest Proxy): If $\mathcal{A}$ is the server, it is given the public key $pk^\times$ and the key share $k_1^\times$ for the MUL scheme. Similarly to Case 1, if $\mathcal{A}$ wishes to attack the AddToMul algorithm, it learns no information about m_b from $\langle \mathcal{E}^\times(m_b), \mathcal{E}^+(m_b), \mathcal{E}^+(\mathcal{E}^\times(m_b))\rangle$ by the security definition of the ADD scheme and Lemma 1.

However, if $\mathcal{A}$ attacks the MulToAdd algorithm, it is given access to $\langle \mathcal{E}^\times(m_b), \mathcal{E}^+(\mathcal{E}^\times(m_b)), c'', R'\rangle$. We then focus on what can be inferred from non-ADD ciphertext components

$$\langle \mathcal{E}^\times(m_b) = (m_b h^r, g^r),\ R' = g^{sx_0},\ k_1^\times = x_1\rangle.$$

Lemma 3. *Given $\langle g, h, x_1, s, g^r, g^{sx_0}, m_b h^r\rangle$, any probabilistic polynomial-time adversary $\mathcal{A}$ has negligible advantage $\epsilon_{\mathcal{A}_3}$ in distinguishing $\mathcal{E}^\times(m_0)$ from $\mathcal{E}^\times(m_1)$, assuming that the DDH problem is hard.*

Proof of Lemma 3. Given knowledge of s and x_1, clearly $\mathcal{A}$ can compute g^{x_0} and $(m_b h^r)^{1/x_1} = m_b^{1/x_1} g^{rx_0}$, where x_0 is unknown to $\mathcal{A}$. Using the same proof technique for Lemma 1, we can show that given (g, g^r, g^{x_0}), it is infeasible to distinguish g^{rx_0} from a random group element under the DDH assumption. If there exists an adversary $\mathcal{A}$ with advantage $\epsilon_{\mathcal{A}_3}$ in breaking the security of the MUL scheme, then we can build an algorithm $\mathcal{B}$ to solve the DDH problem with advantage $\epsilon_{\mathcal{A}_3}$. □

What remains is to analyze if $\mathcal{A}$ can deduce any information about the MUL private key just from one of the given key shares.

Lemma 4. *Given only one of the secret key shares, either $k_0^\times$ or $k_1^\times$, any probabilistic polynomial-time adversary $\mathcal{A}$ has only negligible probability $\epsilon_{\mathcal{A}_4}$ in learning the corresponding MUL private key $sk^\times$.*

Proof of Lemma 4. Since $k_0^\times = x_0$ and $k_1^\times = x_1$ are randomly and independently chosen, clearly the distribution for each key share is indistinguishable from $\mathcal{A}$'s viewpoint. Even though $\mathcal{A}$ with knowledge of x_0 (resp. x_1) can compute g^{x_1} (resp. g^{x_1}), recovering the other unknown key share x_1 (resp. x_0) is equivalent to solving the discrete logarithm problem. Thus, given only one of the two key shares, $\mathcal{A}$ can only perform exhaustive search on the other key share, and thus, has only negligible probability $\epsilon_{\mathcal{A}_4}$ in recovering the correct original decryption key $x = x_0x_1$. □

Combining Lemmas 1–4, we have $\epsilon_{she} \geq \epsilon_{\mathcal{A}_1} + \epsilon_{\mathcal{A}_2} + \epsilon_{\mathcal{A}_3} + \epsilon_{\mathcal{A}_4}$. This completes the proof of Theorem 1. □

5 Evaluation

5.1 Implementation

SHE. We implement our SHE scheme using the GNU MP library version 5.1.3 in 1540 lines of `C` code. We choose a modulus N of size 1024 bits for implementing the Paillier AHE and the ElGamal MHE schemes. Our current implementation is unoptimized and does not use any parallelization techniques.

GC. We use FastGC [37], which is implemented in Java, and is one of the fastest available implementations of the MPC framework that is based on GC. For benchmarking purposes, we construct circuits for 32-bit integer multiplication, matrix multiplication, and polynomial evaluation. We assume that the client sends all the inputs along with the circuit to the server, which in turn, evaluates the circuit and returns the results to the client. For every new input which has to be evaluated, a new garbled circuit has to be generated with new input labels, as the garbled circuits in FastGC can be used only once for secure computation. Hence our experiments take into account the time required for the circuit construction, garbling or any circuit upload while measuring the execution time using FastGC.

SWH. Our implementation of SWH is based on a recently released homomorphic encryption library called HElib [45], which is written in `C++`. It implements the BGV homomorphic encryption scheme [6] and makes use of Smart and Vercauteren's packing techniques [60] allowing SIMD-like computation. The optimization provides support for parallel computation on `NSLOTS` with a single SIMD-like instruction. The HElib library supports both bit and integer operations on encrypted data. Moreover, we choose the security parameter to be 80 bits, roughly equivalent to the 1024-bit security level achieved by our SHE scheme. The library, however, does not provide the bootstrapping function required for FHE, and thus imposing a limit on the depth of the evaluated circuit. The number of consecutive multiplication operations required in an application decides the depth of a circuit. The deeper the circuit, the slower is the performance of each multiplication operation. The maximal depth of the circuit that we could support is 26 multiplications. Table 4 shows different choices of levels, number of slots and their corresponding times required for multiplication and the maximal supported number of multiplications.

FHE. We also implement FHE based on the Scarab Library [46], which in turn, is based on the work of Gentry [30], and Smart and Vercauteren [60]. The library implements the "re-encrypt" operation required for bootstrapping and refreshing a ciphertext, and it supports bitwise operations over encrypted bit inputs. In order to be used for our benchmarking, we modify the library such that it supports integer addition and multiplication. There are other faster variants of FHE scheme, such as López-Alt *et al.*'s scheme that is based on NTRU encryption [47]. However, no open source implementation of these schemes is available yet. Nevertheless, Doroz *et al.* [19] report that the NTRU-based scheme can be faster than using HElib by a factor of 6. As we show next, our SHE scheme can be up to roughly $2,500$ times faster (3 orders of magnitude) for cases which yield circuits with greater depths.

Application	**Amazon-Rackspace (251 Mbits/sec)**					**On LAN (937 Mbits/sec)**					**On Same Machine (64.8 Gbits/sec)**				
	SHE	**FHE**	**GC**	**SWH**	**Parallel SWH**	**SHE**	**FHE**	**GC**	**SWH**	**Parallel SWH**	**SHE**	**FHE**	**GC**	**SWH**	**Parallel SWHE**
Matrix Multiplication	Baseline (ms)	Slow down (X times)				Baseline (ms)	Slow down (X times)				Baseline (ms)	Slow down (X times)			
Size 2	340	940	30	−1.44	−121	116	1, 906	56	1.25	−38	41	5, 405	127	4	−14
Size 5	4, 563	1, 094	5	−1.24	−104	1, 825	1, 884	7	1.23	−39	622	5, 523	12	4	−13
Size 10	38, 324	N/S	2	−1.31	−110	14, 959	N/S	4	1.19	−40	4, 973	N/S	9	4	−13
Naive Poly-Eval. Method															
Degree 6	303	790	35	24	−13	49.94	3, 305	164	128	−3	23	7, 074	103	274	−1.20
Degree 13	339	1, 533	35	265	−1.39	141.45	2, 523	66	520	−1.13	50	7, 122	92	1, 468	2.50
Degree 26	424	2, 452	36	2, 699	2.64	284.53	2, 516	39	3, 029	2.40	101	7, 323	90	8, 539	6.76
Horner's Method															
Degree 6	1, 021	234	10	8	−42	66.77	2, 473	119	112	−2.93	26	6, 426	89	292	−1.13
Degree 13	2, 244	232	5	43	−9	244.96	1, 459	36	415	−1.42	57	6, 257	106	1, 779	3.03
Degree 26	4, 483	232	3	279	−4	321.84	2, 224	30	2, 978	2.36	115	6, 209	84	8, 317	6.58

Table 3: Experimental results of SHE compared to SWH, FHE and GC run on 3 different setups. The SHE column reports the time in miliseconds to execute the applications and the other columns report the comparison with SHE as the baseline. A positive value means that SHE performs that many times faster as compared to running the same application with a particular scheme. N/S denotes "not supported".

Level	**Number of slots**	**Time for single multiplication (ms)**	**Max. Mul. Supported**
16	330	669	6
32	588	2471	13
64	1264	9510	26

Table 4: Parameter choices for SWH.

5.2 Selection of Benchmarks

SHE is based on partially homomorphic encryption schemes that support limited but fast arithmetic operations. Hence, we focus on benchmarks that operate on integer values as inputs and do not consider applications using bitwise operations like AES. To evaluate the performance of SHE, we select polynomial evaluation and matrix multiplication that are commonly used in privacy-preserving computation applications as our benchmarks. We evaluate these benchmarks for various input sizes using SHE and other primitives described above. We limit the input sizes to a small number *i.e.*, 26 degree for polynomial and 10 for matrix as the tools we compare to did not scale for larger values.

Polynomial Evaluation. Encrypted polynomial evaluation is a fundamental building block of many interesting privacy-preserving applications, for example, privacy-preserving set intersection [24], private information retrieval [4], and regular expression matching [55]. In our experiments, we measure the computational cost for polynomial evaluation using both the naive method and Horner's rule. The naive way of evaluating a polynomial involves: (i) computing all the terms in the polynomial; and (ii) adding up all the terms. The input to this application is encrypted polynomial coefficients and a point at which the polynomial is to be evaluated. However, the degree of the polynomial is unencrypted. All the inputs are in the integer domain.

Matrix Multiplication. Our second benchmark is based on matrix multiplication, which is a core operation used in various applications such as image transformation, page rank algorithms, and machine learning algorithms []. We evaluate multiplication of two square matrices of different dimensions where the inputs are encrypted. Unlike polynomial evaluation which may have long chains of multiplications, matrix multiplication requires only a small depth circuit. Thus our selected benchmarks allow us to evaluate the efficiency of SHE on both small and larger depth circuits.

5.3 Evaluation

We perform our evaluation with the following goals:

- To examine the impact of the network latency between the two nodes, including on real clouds.
- To evaluate the efficiency of SHE for different depth of circuits which is indicated by the number of continuous chain of multiplications.
- To compare the performance of SHE with publicly available implementations for secure arithmetic computation on encrypted data.

Methodology. We measure the performance of SHE for three different setups: (i) between two clouds, (ii) within a LAN, and (iii) on the same machine.

For evaluating our scheme in the two-cloud model (*i.e.*, between proxy and server), we use Amazon EC2 and Rackspace servers connected over a bandwidth of 251 Mbits/sec (measured using `iperf` [63]). We set up a Ubuntu-trusty-14.04-amd64-server instance on Amazon EC2 having 8 vCPU's, 28 compute units (ECU) and 15 GB memory. We use this instance as our server. We then host the proxy as the same Ubuntu version on Rackspace with 4 vCPU's and 15 GB memory. We host the proxy and the server in geographically different regions to evaluate the communication overhead between these two clouds. Our proxy is hosted in US Northern Virginia, while the server is hosted in US West (Oregon).

For evaluation on a local area network (LAN), we use two machines connected over a bandwidth of 937 Mbits/sec. We use a Dell Latitude E6430s machine with 8 GB RAM and Intel(R) Core(TM) i7-3520M CPU at 2.90 GHz 4-core processor having cache size of 4096 KB as the proxy. On the other hand, we use a Dell Optiplex D990MT Desktop machine with 8 GB RAM and Intel(R) Core(TM) i7-2600M CPU at 3.40 GHz 8-core processor having cache size of 8192 KB as the server.

In the third setting, we run both the proxy and the server on the same machine. One can imagine this to be useful in a scenario where a machine contains two CPUs manufactured by different vendors and a possible threat arises from a hardware backdoor present in either of the CPUs [65]. The two CPUs can act as the two nodes in our SHE design and execute the proxy and server logic respectively. We use the Dell Optiplex D990MT Desktop machine to measure the performance when proxy and server are hosted on the same machine.

Summary of Evaluation. The results of our evaluation can be summarized as follows:

- SHE is a new design point in the space of secure arithmetic computation in the two cloud model and is expressive enough for applications of different circuit depths like matrix multiplication (depth = 1) and polynomial evaluation (depth = 26)[8].
- In applications with no data parallelism, our scheme demonstrates a stark improvement of 2500 time speed up over FHE and SWH, and 35 times over GC for naive evaluation of a polynomial (deep circuit) and 1000 times speed up over FHE, 2 times over GC and almost comparable performance with SWH for matrix multiplication (low depth circuit).
- The dominant cost in the performance is due to the communication overhead between two clouds which varies our speed up from 2500 to 7000 over FHE when moved from cloud setting to same machine setting that has zero network latency.

5.4 Results

Table 3 shows the results of our evaluation in the three experimental setups. It gives the time required, in milliseconds, to execute our benchmark applications for various sizes of input. It also shows the speed up of our scheme as compared to FHE, GC , SWH and SWH when the parallel functionality is used. We report on matrices of size 2, 5 and 10 since the FHE library did not scale for matrix size of 10 in our experiments. We evaluate the polynomial for a maximum of 26 degree as the SWH library could not scale for more than 26 degree because of its limited expressiveness.

Performance of SHE. Our scheme takes around 400 ms to evaluate a polynomial of degree 26 using the naive method and 4.5 seconds using Horner's method in the cloud setting. Both methods require conversions from ElGamal ciphertexts to Paillier ciphertexts (MulToAdd) proportional to the degree of the polynomial. However in the naive way of evaluating a polynomial, all the multiplications are done together and sent to the proxy for conversion in a single request; whereas Horner's method requires to make communication calls to the proxy that is equal to the degree of the polynomial. Thus, the communication overhead increases in the latter case. The time for evaluating a polynomial using both the naive method and Horner's method are almost the same (101 ms and 115 ms resp.) when executed on the same machine (zero network latency). This confirms our observation that the dominant cost comes from the communication overhead between the two clouds. Matrix multiplication makes N^3 conversion calls for multiplying two matrices of size $N \times N$. On the same machine setting, SHE takes 5 s for multiplying matrices of size 10. This shows that the second dominant factor after network latency that effects the performance is the number of conversions from ElGamal to Paillier ciphertexts. Thus, SHE is comparatively much more efficient for applications which have longer chains of multiplications and fewer conversion calls.

[8]depth is defined by the number of continuous multiplication operations in the application

KeyGen	MulToAdd	**Pai. Mul**	AddToMul	**Pai. Add**	**ElG. Mul**
66 ms	11 ms	2 ms	5 ms	0.003 ms	0.002 ms

Table 5: Time required for individual operations on SHE (averaged over 10 runs).

Comparison with other schemes. The performance of SHE is roughly 2500 and 7000 times better than FHE when evaluated on two-cloud servers and on the same machine, respectively, for applications with a high-depth circuit like polynomial evaluation. FHE involves the computationally expensive "re-encrypt" function after every multiplication operation in order to remove the noise from the ciphertext. This operation increases the execution time for longer chain of multiplications. SHE scheme does not suffer from any such costly computations, and thus, shows consistent speed up in the performance even for execution of deep circuits. However, the network latency between the proxy and the server effects the speed up in the performance of SHE when it is run in the cloud setting.

For non-parallel applications, our scheme is around 2500 times faster than SWH in the two cloud setting and 8500 times faster when executed on the same machine setting for high-depth polynomial evaluation. Our scheme shows comparable performance against SWH for matrix multiplication which has depth size of 1 (as the depth of circuit is only one, we set the level parameter as 2 in the SWH scheme for evaluating this particular application). This is due to the fact that SWH is faster for small chain of multiplications and slows down with increase in the circuit depth. After a certain number of multiplications, the ciphertext becomes noisy and cannot be used further. This limits the maximum multiplications that can be supported by SWH. Table 4 shows the maximal number of supported multiplications in the HElib library. For applications that can support highly parallel tasks, we show the comparison to our scheme by dividing the execution time of SWH with the `NSLOTS` (the maximum parallelism possible) in Table 4. We observe that SHE is considerably slow for matrix multiplication and Horner's method, and is comparable to SWH for naive method of polynomial evaluation using batch processing offered by SWH. However, for dynamic queries, *e.g.*, private information retrieval, where client does not have enough input data available to leverage the benefits of such batch processing in SWH, SHE is faster. Even with parallel processing, SHE performs 2.64 to 6.76 times better than SWH for 26 degree polynomial. The benefits of SHE in terms of efficiency and expressiveness are visible for deep chains of multiplication which SWH cannot support due to its limitations.

Our benchmarks execute around 35 times faster using SHE as compared to running with GC in the cloud setting. The main performance overhead in GC comes from the circuit generation and circuit garbling operations that are proportional to the size of the input. Unlike SHE which can reuse the same application to evaluate on different set of encrypted inputs, GC is limited to a one-time use of the circuit for a particular set of inputs.

Individual Operation costs. Table 5 shows the computation times required for various operations in the SHE scheme. The key generation step is performed by the client and takes around 66 ms. Paillier homomorphic addition and ElGamal homomorphic multiplication require only 0.003 ms and 0.002 ms, respectively. Multiplying a Paillier ciphertext with a constant value and AddToMul both require on an average of less than 5 ms. The MulToAdd call call to the proxy takes around 10 ms (which includes the network latency). All the timing measurements are averaged over 10 runs of each operation.

6 Related Work

Secure Arithmetic Operations. In the secure two-party or multi-party computation (2PC/MPC) setting, there has been an extensive amount of work in secure arithmetic operations, such as addition, multiplication, division, exponentiation and modulo reduction over integers, see for example [26, 39, 17, 33, 35, 62, 67]. Typically, 2PC is performed over two input values a and b from each party such that both parties learn only their respective input and the output $c = a \circ b$ for some operator $\circ$. The communication and computational complexity of an MPC protocol is generally high. For example, Rabin's n-party *multiplication* protocol via *secret sharing* takes a single round of communication and involves the exchange of $O(n^2)$ elements from a relevant finite field [26]. This is because each of the n players must perform Shamir secret sharing as part of the protocol. In the garbled circuit setting, on the other hand, multiplication of two l-bit unsigned integers requires a circuit of size approximately $O(l^{1.6})$ with Karatsuba's algorithm. The complexity can increase drastically even for a slightly more advanced operation. For example, Yu *et al.*'s two-party *exponentiation* protocol takes 24 rounds and requires roughly $279l$ secure multiplications [67] for l-bit shares.

Outsourced MPC. Kamara *et al.* [40, 59] studied MPC in a server-aided setting, where the involved clients/parties, particularly those with limited computing power, are able to outsource expensive computation to an untrusted server with vast amount of computational resources. The server neither has any input to the computation nor receives any output from the computation. However, any secure computation would require interaction between the involved parties and the server. On the other hand, Halevi *et al.* [34] proposed a non-interactive MPC solution that allows outsourcing of computation to a web server. However, the server is allowed to learn the output of the computation.

Peter *et al.* [56] recently extended the above previous works to the multi-key setting, proposing an MPC construction that allows input to be encrypted with multiple keys corresponding to different parties/clients. Their construction, as with our work, relies on two non-colluding servers and no interaction is required between the clients and the servers during secure computation. However, it makes use of traditional MPC protocols as building blocks, and thus, the resulting construction has relatively high computational overhead. For example, one multiplication operation alone takes approximately 200 ms using the Bresson *et al.* encryption scheme [7] with a 1536-bit modulus. Other work on secure polynomial evaluation in the multi-key setting can be found in [66].

Threshold FHE & Multi-key FHE. The notion of FHE typically assumes a semi-honest adversary model. As observed by Gentry [30], however, we can extend this to the fully malicious model using threshold cryptography. Asharov *et al.* [1] proposed a threshold FHE (TFHE) scheme in which all clients share a common public key and decryption requires a collaborative effort from all the clients. Using their TFHE scheme, one can derive simple MPC protocols secure against fully malicious attackers, while preserving relatively low communication complexity.

López-Alt *et al.* [47] proposed the concept of multi-key FHE (MFHE) that allows computation on data encrypted under multiple unrelated public keys, analogous to multi-key MPC [56]. One major advantage of such approach is that dynamically chosen computation can be performed on data belonging to arbitrary sets of users on-the-fly and in a non-interactive manner. Nevertheless, it still relies on an interactive protocol during decryption of the computed results. The performance of the schemes is faster than FHE but still unsuitable for practical use.

Hybrid Approach. There also exist works on combining homomorphic encryption with 2PC/MPC protocols to support a wider range of operations, including both arithmetic and Boolean circuits, while achieving more optimal performance compared to using each primitive/protocol individually [35, 13, 68]. Our SHE scheme is orthogonal to the hybrid framework and can be used as a faster alternative to homomorphic encryption for supporting arithmetic operations. Algorithms for automated protocol selection among different types of secure computation protocols for optimal performance can be found in [43].

Others. Naehrig *et al.* [53] showed that SWH is sufficient for a number of real-world applications, particularly in the medical, financial, and the advertising domains. Reasonable efficiency can be achieved by carefully performing application-specific optimization to an SWH scheme. The efficiency of our SHE scheme is comparable to that of an SWH scheme for a small number of homomorphic additions and multiplications. For a larger number of similar operations, our scheme outperforms the latter. We provide a more detailed performance comparison against SWH in Section 5. In a recent independent work, Boneh *et al.* [4] showed how private database conjunction queries can be performed using techniques from SWH. For a database with $2,000$ records, the required query time is roughly 10 min.

Our solution is also related to Tople *et al.*'s proposal of AutoCrypt [64], which supports both homomorphic addition and multiplication on outsourced encrypted web contents using the Paillier ADD and the ElGamal MUL schemes. However, one major drawback of AutoCrypt is that the client has to reveal its decryption keys to the hosting server, such that the server can decrypt and re-encrypt ciphertexts on the client's behave. (Hence, the server has full access to the plaintexts and is trusted to not expose them to any outsider.)

7 Open Problems

Many interesting open problems arise from this work. First, it would be interesting to investigate if our SHE scheme can be extended to support parallel homomorphic computation. Second, it is also interesting to explore if our ideas and techniques can be adapted or generalized to improve existing secure computation techniques for more complex arithmetic operations, such as division, exponentiation, and modulo reduction over integers. Finally, if there exists a provably secure MUL scheme which does not inherit the zero-plaintext problem and that is compatible with any existing ADD scheme, then more efficient and expressive SHE may be realizable.

Acknowledgement

We thank Florian Kerschbaum for useful feedback on a preliminary version of this paper.

References

[1] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques – Advances in Cryptology (EUROCRYPT)*, pages 483–501, 2012.

[2] D. Asonov and J. Freytag. Almost optimal private information retrieval. In *Proceedings of the International Conference on Privacy Enhancing Technologies (PET)*, pages 209–223, 2003.

[3] P. Bogetoft, D.L. Christensen, I. Damgård, M. Geisler, T.P. Jakobsen, M. Krøigaard, J.D. Nielsen, J.B. Nielsen, K. Nielsen, J. Pagter, M.I. Schwartzbach, and T. Toft. Secure multiparty computation goes live. In *Proceedings of the International Conference on Financial Cryptography and Data Security (FC)*, pages 325–343, 2009.

[4] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D.J. Wu. Private database queries using somewhat homomorphic encryption. In *Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS)*, pages 102–118, 2013.

[5] BoxCryptor. `https://www.boxcryptor.com/`.

[6] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:111, 2011.

[7] E. Bresson, D. Catalano, and D. Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security – Advances in Cryptology (ASIACRYPT)*, pages 37–54, 2003.

[8] S. Bugiel, S. Nürnberger, A. Sadeghi, and T. Schneider. Twin clouds: Secure cloud computing with low latency. In *Proceedings of the IFIP International Conference on Communications and Multimedia Security (CMS)*, pages 32–44, 2011.

[9] D. Catalano and R. Gennaro. New efficient and secure protocols for verifiable signature sharing and other applications. In *Proceedings of the Annual Cryptology Conference – Advances in Cryptology (CRYPTO)*, pages 105–120, 1998.

[10] D. Champagne and R.B. Lee. Scalable architectural support for trusted software. In *Proceedings of the IEEE International Conference on High-Performance Computer Architecture (HPCA)*, pages 1–12, 2010.

[11] S. Chhabra, B. Rogers, Y. Solihin, and M. Prvulovic. SecureME: a hardware-software approach to full system security. In *Proceedings of the ACM International Conference on Supercomputing (ICS)*, pages 108–119, 2011.

[12] S.G. Choi, A. Elbaz, A. Juels, T. Malkin, and M. Yung. Two-party computing with encrypted data. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security – Advances in Cryptology (ASIACRYPT)*, pages 298–314, 2007.

[13] A. Choudhury, J. Loftus, E. Orsini, A. Patra, and N.P. Smart. Between a rock and a hard place: Interpolating between MPC and FHE. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security – Advances in Cryptology (ASIACRYPT)*, pages 221–240, 2013.

[14] S.S.M. Chow, J. Lee, and L. Subramanian. Two-party computation model for privacy-preserving queries over distributed databases. In *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*, 2009.

[15] CloudFogger. `http://www.cloudfogger.com/en/`.

[16] E. De Cristofaro, Y. Lu, and G. Tsudik. Efficient techniques for privacy-preserving sharing of sensitive information. In *Proceedings of the International Conference on Trust and Trustworthy Computing (TRUST)*, pages 239–253, 2011.

[17] I. Damgård, M. Fitzi, E. Kiltz, J.B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Proceedings of the Theory of Cryptography Conference (TCC)*, pages 285–304, 2006.

[18] I. Damgård and Y. Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *Proceedings of the Annual Cryptology Conference – Advances in Cryptology (CRYPTO)*, pages 378–394, 2005.

[19] Y. Doroz, Y. Hu, and B. Sunar. Homomorphic AES evaluation using NTRU. *IACR Cryptology ePrint Archive*. Report 2014/039.

[20] W. Du and Z. Zhan. A practical approach to solve secure multi-party computation problems. In *Proceedings of the Workshop on New Security Paradigms (NSPW)*, pages 127–135, 2002.

[21] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of the Annual Cryptology Conference – Advances in Cryptology (CRYPTO)*, pages 10–18, 1984.

[22] U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation (extended abstract). In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*, pages 554–563, 1994.

[23] M. Frankling and S. Haber. Joint encryption and message-efficient secure computation. *Journal of Cryptology*, 9(4):217–232, 1996.

[24] M.J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques – Advances in Cryptology (EUROCRYPT)*, pages 1–19, 2004.

[25] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Proceedings of the Annual Cryptology Conference – Advances in Cryptology (CRYPTO)*, pages 465–482, 2010.

[26] R. Gennaro, M.O. Rabin, and T. Rabin. Simplified VSS and fact-track multiparty computations with applications to threshold cryptography. In *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 101–111, 1998.

[27] C. Gentry, S. Halevi, and N.P. Smart. Fully homomorphic encryption with polylog overhead. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques – Advances in Cryptology (EUROCRYPT)*, pages 465–482, 2012.

[28] C. Gentry, S. Halevi, and N.P. Smart. Homomorphic evaluation of the AES circuit. In *Proceedings of the Annual Cryptology Conference – Advances in Cryptology (CRYPTO)*, pages 850–867, 2012.

[29] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Proceedings of the Annual Cryptology Conference – Advances in Cryptology (CRYPTO)*, pages 75–92, 2013.

[30] G. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*, pages 169–178, 2009.

[31] O. Goldreich. *The Foundations of Cryptography – Volume 2, Basic Applications.* Cambridge University Press, 2004.

[32] S. Goldwasser, Y.T. Kalai, R.A. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Proceedings of the ACM Symposium on Theory of Computing Conference (STOC)*, pages 555–564, 2013.

[33] J. Guajardo, B. Mennink, and B. Schoenmakers. Modulo reduction for paillier encryptions and application to secure statistical analysis. In *Proceedings of the International Conference on Financial Cryptography and Data Security (FC)*, pages 375–382, 2010.

[34] S. Halevi, Y. Lindell, and B. Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *Proceedings of the Annual Cryptology Conference – Advances in Cryptology (CRYPTO)*, pages 132–150, 2011.

[35] W. Henecka, S. Kögl, A. Sadeghi, T. Schneider, and I. Wehrenberg. Tasty: Tool for automating secure two-party computations. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 451–462, 2010.

[36] S. Hohenberger and A. Lysyanskaya. How to securely outsource cryptographic computations. In *Proceedings of the Theory of Cryptography Conference (TCC)*, pages 264–282, 2005.

[37] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of the USENIX Security Symposium*, page 35, 2011.

[38] Y. Huang, J. Katz, and D. Evans. Efficient secure two-party computation using symmetric cut-and-choose. In *Proceedings of the Annual Cryptology Conference – Advances in Cryptology (CRYPTO)*, pages 18–35, 2013.

[39] M. Jakobsson and A. Juels. Addition of ElGamal plaintexts. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security – Advances in Cryptology (ASIACRYPT)*, pages 346–358, 2000.

[40] S. Kamara, P. Mohassel, and M. Raykova. Outsourcing multi-party computation. *IACR Cryptology ePrint Archive.* Report 2011/272.

[41] J. Katz and Y. Lindell. *Introduction to Modern Cryptography.* CRC Press, 2007.

[42] J. Katz and L. Malka. Constant-round private function evaluation with linear complexity. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security – Advances in Cryptology (ASIACRYPT)*, pages 556–571, 2011.

[43] F. Kerschbaum, T. Schneider, and A. Schröpfer. Automatic protocol selection in secure two-party computations. In *Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS)*, pages 566–584, 2014.

[44] V. Kolesnikov and T. Schneider. A practical universal circuit construction and secure evaluation of private functions. In *Proceedings of the International Conference on Financial Cryptography and Data Security (FC)*, pages 83–97, 2008.

[45] HELib: Homomorphic Encryption Library. `https://github.com/shaih/HElib`.

[46] Scarab Library. `https://hcrypt.com/scarab-library/`.

[47] A. López-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the Symposium on Theory of Computing Conference (STOC)*, pages 1219–1234, 2012.

[48] S. Lu and R. Ostrovsky. Distributed oblivious RAM for secure two-party computation. In *Proceedings of the Theory of Cryptography Conference (TCC)*, pages 377–396, 2013.

[49] J.M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V.D. Gligor, and A. Perrig. TrustVisor: Efficient TCB reduction and attestation. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 143–158, 2010.

[50] K.S. McCurley. A key distribution system equivalent to factoring. *Journal of Cryptology*, 1(2):95–105, 1988.

[51] P. Mohassel and B. Riva. Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In *Proceedings of the Annual Cryptology Conference – Advances in Cryptology (CRYPTO)*, pages 36–53, 2013.

[52] R.A. Mollin. *RSA and Public-Key Cryptography.* CRC Press, 2003.

[53] M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the ACM Cloud Computing Security Workshop (CCSW)*, pages 113–124, 2011.

[54] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques – Advances in Cryptology (EUROCRYPT)*, pages 223–238, 1999.

[55] J. Patel, A.X. Liu, and E. Torng. Bypassing space explosion in regular expression matching for network intrusion detection and prevention systems. In *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*, 2012.

[56] A. Peter, E. Tews, and S. Katzenbeisser. Efficiently outsourcing multiparty computation under multiple keys. *IEEE Transactions on Information Forensics and Security*, 8(12):2046–2058, 2013.

[57] R.A. Popa, C.M.S. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 85–100, 2011.

[58] A. Prakash and A. Aziz. Binary decision graph. In *Encyclopedia of Algorithms*. Springer, 2008.

[59] M. Raykova. *Secure Computation in Heterogeneous Environments: How to Bring Multiparty Computation Closer to Practice?* PhD thesis, Columbia University, 2012.

[60] N. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Proceedings of the International Conference on Practice and Theory in Public Key Cryptography (PKC)*, pages 420–443, 2010.

[61] E. Stefanov and E. Shi. Multi-cloud oblivious storage. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 247–258, 2013.

[62] T. Toft. Sub-linear, secure comparison with two non-colluding parties. In *Proceedings of the International Conference on Practice and Theory in Public Key Cryptography (PKC)*, pages 174–191, 2011.

[63] Iperf Tool. `http://iperf.fr/`.

[64] S. Tople, S. Shinde, Z. Chen, and P. Saxena. Autocrypt: Enabling homomorphic computation on servers to protect sensitive web content. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 1297–1310, 2013.

[65] A. Waksman and S. Sethumadhavan. Silencing hardware backdoors. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 49–63, 2011.

[66] B. Wang, M. Li, S.S.M. Chow, and H. Li. Computing encrypted cloud data efficiently under multiple keys. In *Proceedings of the IEEE Conference on Communications and Network Security (CNS)*, pages 504–513, 2014.

[67] C. Yu, S.S.M. Chow, K. Chung, and F. Liu. Efficient secure two-party exponentiation. In *Proceedings of the Cryptographers' Track at the RSA Conference – Topics in Cryptology (CT-RSA)*, pages 17–32, 2011.

[68] Y. Zhang, A. Steele, and M. Blanton. PICCO: a general-purpose compiler for private distributed computation. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 813–826, 2013.

[69] Z. Zhou, J. Han, Y. Lin, A. Perrig, and V.D. Gligor. KISS: "key it simple and secure" corporate key management. In *Proceedings of the International Conference on Trust and Trustworthy Computing (TRUST)*, pages 1–18, 2013.