PREFHE, PREFHE-AES and PREFHE-SGX: Secure Multiparty Computation Protocols from Fully Homomorphic Encryption and Proxy ReEncryption with AES and Intel SGX

Cavidan Yakupoglu(✉) and Kurt Rohloff

New Jersey Institute of Technology, Newark, NJ 07102, USA
{cy267,rohloff}@njit.edu

Abstract. We build our secure multiparty computation (MPC) protocols on top of the fully homomorphic encryption (FHE) scheme, BFVrns, and augment it with Proxy Re-Encryption (PRE). We offer three distinct secure MPC protocols that make use of the Advanced Encryption Standard (AES) and Intel Software Guardian Extension (SGX). The PREFHE protocol is based on FHE and PRE that offers a reasonable computational time of milliseconds or seconds, depending on the function computed jointly on the parties' encrypted data. It offers 4 rounds and a communication cost that only depends on the parties' ciphertext size. PREFHE-AES employs AES-128 encryption, which reduces the cost of communication to bits rather than kilobytes or megabytes while maintaining the same number of rounds as PREFHE. PREFHE-SGX is another novel approach that reduces the number of rounds from 4 to 3 by utilizing only one untrusted server. Additionally, it delivers a reasonable level of performance that is applicable to real-world use cases. We pioneer the use of SGX and FHE in secure MPC protocols, resulting in reduced number of rounds. In the protocols, after parties send their encrypted data to the server, they are not required to be online that improves practicality in the protocols. Additionally, the parties are not required to collaborate on any computations during the encryption and decryption phases that makes our protocols more efficient than other proposed protocols.

Keywords: Multiparty computation · Homomorphic encryption · RLWE · Proxy reencryption · Intel SGX

1 Introduction

With the improvement of technology, new approaches have been proposed such as cloud storage/computing or distributed computing. While these services offer ease and practicality in real life, it comes at a price of privacy. When we store

F. Li et al. (Eds.): SecureComm 2022, LNICST 462, pp. 819–837, 2023.
https://doi.org/10.1007/978-3-031-25538-0_43

our data in the cloud, the cloud or other cloud users may snoop our confidential data. To protect this sensitive information, storing the data in encrypted way is the first step to solve this problem. This idea leads to another problem of making computations on this data. Homomorphic encryption (HE) enables computation of some functions on the encrypted data while FHE allows any kind of computation. Secure MPC takes this approach to another level with computing on the encrypted data collaboratively. It enables n parties computing a function F on n encrypted inputs, such that each party learns the result of the function but not any others' input.

Secure MPC was first introduced by Yao in 1986 [48] for two-party case and multiparty case by Goldreich, Micali and Wigderson [27]. For constructing secure MPC two different approaches have been proposed in general. The first method is based on secure secret sharing where there is an honest majority among the parties [4,10]. The second approach uses binary circuit representation of the function [27,48]. Secure MPC has progressed in terms of efficiency with the recent studies. Based on HE, many MPC studies were proposed such as [5,14,17,18,38]. After the introduction of FHE, many MPC studies were proposed based on FHE such as [33] and FHE variants such as threshold FHE [3] and multikey FHE [34].

FHE refers to a family of encryption methods that allows evaluation of arbitrary computation on encrypted data. FHE was first proposed by Rivest et al. [41] in 1978. It was an open problem until Gentry proved that FHE is possible by using lattices in his work [25] in 2009. After Gentry's breakthrough, many variants of FHE were proposed in the literature [19,43]. FHE offers privacy for data and efficiency on distributed computing on encrypted data and enables powerful privacy-preserving applications. One of these important applications is the family of secure MPC protocols. FHE allows constructing efficient secure MPC schemes. Lattice-based cryptography is known for its quantum resistance, so our secure MPC protocols are quantum-resistant as well. We propose three secure MPC protocols with BFVrns [7,22,29] that is a ring-learning with error (RLWE) based FHE scheme. We offer efficient secure MPC implementations based on PALISADE lattice cryptography library[1] [39].

Efficiency of secure MPC is measured based on three metrics such as rounds, computation cost and communication cost. Rounds refers to total number of communication between parties and servers and among servers. Communication cost is used for the amount of data that is transferred during the execution of the protocol. Computation cost is the amount of computation conducted in each party and server to process the protocol. Some studies suggest some secure MPC protocols that computational and communication complexities of all the parties who participate in the protocol depend on the complexity of the function [4,10,27,30]. This type of protocols may not be efficient in real-life application due to the high complexity of the function. Some recent protocols require communication at every gate and all parties to be online [5,16,18] which is not very possible all the time. In some protocols, decryption is jointly computed [34] which is not very practical in most of use cases. The protocol of [3] requires the

[1] https://gitlab.com/palisade/palisade-release.

parties to generate a joint public key online which is not always achievable in real life. The study in [28] suggests a secure MPC protocol for a small class of functions that is not enough for most of the applications. To overcome these challenges, we introduce our FHE and PRE-based secure MPC construction and its two variants with AES and SGX.

We introduce a novel concept of secure MPC as a hybrid of FHE and PRE. Our secure MPC constructions mainly depend on FHE approach. We make use of its proxy-reencryption (PRE) capability for constructing the secure MPC protocols. PRE allows us to conduct computation on encrypted data which are encrypted under many different keys while FHE enables secure computation for two parties. The recent Threshold FHE schemes have some pitfalls that a third party should know all secret keys of the parties to create a common multiparty operation key. We overcome this problem by using PRE. PRE allows reencryption of the ciphertext under a common key to enable evaluation of the function on the given ciphertexts by different parties. It also enables reencrytion of the result of the function evaluation under individual keys of each party so that each party can decrypt the result without exchanging any keys or partial decrypted results.

Our protocols depend on honest-but-curious (semi-malicious) security model that is a prevalent security model used in practical implementations of secure MPC due to performance reasons [6,9,20,21,32]. We assume that two untrusted servers in the protocols do not collude. The assumption of existence of two non-colluding servers is widely used in many studies [9,11,15,21]. Furthermore, we propose further improvements to enhance the security model to malicious model with non-interactive zero knowledge (NIZK) proofs proving plaintext knowledge as done in the work [34]. Our protocols can be easily used by the privacy-preserving applications such as auctions, electronic voting or secure machine algorithms can be good use cases for our secure MPC protocols.

In our main protocol PREFHE, we assume that we have two non-colluding but untrusted servers. One of the server is responsible for creating FHE parameters and generating common key pair and reencryption keys for each part. Another one conducts evaluation of the function on the ciphertexts and reencryption of the result. In PREFHE, the parties are involved in the protocol while uploading their encrypted data and decrypting the final result. In the rest of the protocol, any party does not need any further communication with the servers. The communication between the clients or servers does not depend on the complexity of the function to be evaluated. Also, the computation of the parties/clients does not depend on the complexity of the function F. All parties are not expected to be online after sending their encrypted data to the server. In the encryption or decryption phase, we do not require any joint computation by the parties. The parties only send their encrypted data to the server and they get the final result in encrypted at the end of the protocol.

In the AES variation of PREFHE, PREFHE-AES, the parties send their data encrypted under AES-128 that enables less communication cost between the parties and the server (S_1). We adapt the idea of the work in [36] to our

secure MPC protocol. This adaption creates an extra computational cost on the server (S_1), but this can be handled with improvements such as parallelization or including special hardware for AES encryption/decryption. We propose SGX variation of PREFHE as PREFHE-SGX that combines PALISADE library (for implementation of the FHE scheme), Intel SGX technology and secure MPC. We use Gramine as a bridge between PALISADE and Intel SGX to avoid adjusting the PALISADE code to SGX [31].

1.1 Our Contributions

PREFHE: PREFHE proposes the first combination of FHE and PRE approach for secure MPC protocols. It enables practical secure MPC implementation for real-life use cases. It requires 4 rounds including key exchange phase which is not included as a round in many previous works. The parties in the PREFHE protocol are not required to be online after they send their encrypted data to the server. This makes the protocol a good fit for the applications that do not allow online access all the time. The parties send their data encrypted under the BFVrns scheme that may result in kilobytes sometimes megabytes. This may cause some problems in low-bandwidth network or applications that have small memory or network channel. To handle this problem, we propose AES version of PREFHE.

PREFHE-AES: To reduce the communication cost of PREFHE, the parties send their data encrypted in AES-128 which has smaller data size to be sent to the server. We reduce communication cost, but it comes at a price of computation cost on the server side. Computing decryption of AES-128 homomorphically is a costly operation. For applications that require less communication cost, PREFHE-AES is a good fit with the same number of rounds as PREFHE at a reasonable computational time.

PREFHE-SGX: For the applications that allow only one server or have constraints of local computation, PREFHE-SGX is a convenient approach with 3 rounds that has less than most of the state-of-the-art protocols. It also introduces reasonable computational cost which only depends on the function to be evaluated. We lead adapting SGX and FHE together in secure MPC protocols that leads practical use of secure MPC protocols in real-life applications.

Performance In Practice: Our three protocols provide reasonable amount of running time on the client and server side with milliseconds or seconds that mainly depends on the function. Even if circuit depth is high, we use an efficient FHE scheme, BFVrns, that takes advantages of Residue Number System (RNS) and packing of plaintext in SIMD manner [24,42].

1.2 Related Works

The idea of using threshold homomorphic encryption in MPC protocols was first presented by Cramer, Damgard, and Nielsen [14]. Somewhat homomorphic encryption was used to boost implementation of MPC protocols in some studies

such as [5,18]. In their protocols, all parties compute proportional to the complexity of the function to be computed and interact at every gate. Choudhury et al. proposed better communication at a computation cost [12]. Their work suggested a kind of interactive bootstrapping protocol to refresh ciphertexts. Cloud server idea came with the work by Kamara et al. [30]. They proposed server-aided MPC idea by assigning large amount of works from the computation to some parties. Halevi et al. suggested the idea of secure computation on the web to minimize communication between parties in the computation [28]. After FHE is proposed by Gentry [25], new approaches based on FHE were suggested by Lopez et al. [33,34] using multikey FHE approach. Asharov et al. presented an efficient threshold FHE based secure MPC scheme in terms of round, communication and computation costs [3]. TFHE schemes allow to jointly generate a common FHE public key along with a secret key that is shared by them later. For decryption, they conduct a collobarative decryption process on ciphertexts to get the final plaintext without learning others' inputs. Garg et al. achieved 2-round MPC from indistinguishability obfuscation [23]. As an optimization they suggested another 2-round MPC protocol from multikey FHE that is independent of the circuit to be computed. [37] proposes a Intel SGX as TEE and FHE-based multiparty computation that makes use of a certification authority (CA) for aunthentication of the parties. [46] presents a multiparty construction that uses partial HE and Intel SGX as TEE. The latest two constructions do not use Gramine.

2 Background

2.1 Fully Homomorphic Encryption Scheme: BFVrns

We give a brief explanation of the BFVrns scheme referenced from [7,22,29] but mainly from [29]. Fan and Vercauteren [22] present the RLWE version of work proposed by Brakerski in [7]. Residue Number System (RNS) variant of the BFV scheme is presented by Halevi et al. [29] for more efficient procedures for BFV and it is implemented in PALISADE library. BFVrns provides improvements on decryption and homomorphic multiplication by using Chinese Remainder Theorem (CRT) representation. BFVrns utilizes some parameters such as m, t, $q \in \mathbb{Z}$. t stands for plaintext modulus, N stands for $\phi(m) = N$, ciphertext modulus is $q = \prod_{i=1}^{k}, q_i$ for the same size q_i, σ is the standard deviation of error distribution χ. rw stands for the size of the relinearization window. Let rings be $\mathcal{R} = \mathbb{Z}[x]/\Phi_m(x)$, $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$, $\mathcal{R}_t = \mathcal{R}/t\mathcal{R}$. We choose a uniform $\alpha_i \in \mathcal{R}_q$ and $e_i \leftarrow \chi$, ${q_i}^* = q/q_i$, $q_i' = [{q_i}^{*-1}]_{q_i}$, $\beta_i = [q_i' {q_i}^* s^2 - \alpha_i s + e_i]_q$ for $i = 0, 1, \ldots, k$. The public key consists of pk and $W_i := (\beta_i, \alpha_i)$.

- **KeyGeneration:** Secret Key: Sample $s \leftarrow \chi$, set $sk = (1, s) \in \mathcal{R}^2$.
  Public Key: Sample $a \leftarrow \mathcal{R}_q$, $e \leftarrow \chi$, set $pk = ([-(a \cdot s + e)]_q, a) \in \mathcal{R}_q^2$.
- **Encryption**(m, pk)**:** $m \in \mathcal{R}_t$, $u \leftarrow \chi$, $e_0, e_1 \leftarrow \chi$, $ct = [u \cdot pk + (e_0, e_1) + (\Delta m, 0)]$ where $\Delta = q/t$. Output ct.

- **Decryption**(ct, sk)**:** $ct = (ct[0], ct[1])$, $x = [\langle sk, ct\rangle]_q = [c[0] + c[1]s]_q$ and output $m := [\lceil x \cdot t/q \rfloor]_t$.
- **Add**(ct_0, ct_1)**:** Output $[ct_0 + ct_1]_q$.
- **Mult**(ct_0, ct_1)**:** For ct_0, ct_1, tensoring and relinearization are computed as follows:
  - **Tensoring:** $c[0] = ct_0[0]ct_1[0]$, $c[1] = ([ct_0[0]ct_1[1] + [ct_0[1]ct_1[0]]_q)$, $c[2] = ct_0[1]ct_1[1]$ and $c = (c[0], c[1], c[2])$. Output $c'[i] = [\lceil t/q \cdot c[i] \rfloor]_q$ for $i = 0, 1, 2$.
  - **Relinearization:** Decompose $c'[2]$ into its CRT components $c'[2][i] = [c'[2]]_{q_i}$, set $c''[0] = [\sum_{i=1}^{k} \beta_i c'[2][i]]_q$, $c''[1] = [\sum_{i=1}^{k} \alpha_i c'[2][i]]_q$, output $ct_{mult} = [(c'[0] + c''[0], c'[1] + c''[1])]_q$.
- **MultiPartyKeyGen**(pk)**:** $pk = (p_0, p_1)$, $a \leftarrow p_1$, $s \leftarrow \chi$, $b = -(e + (a \cdot s)) + p_0$, set new key $pk' = (b, a)$ and $sk = s$.
- **ReKeyGen**$(newpk, oldsk)$**:** For $rw = 0$, $newpk = (p_0, p_1)$, $e1_i, e2_i \leftarrow \chi$, $u_i \leftarrow \chi$. For each element in $oldsk$; $c0_i = p_0 \cdot u_i + e1_i + f_i$ where f_i are elements at index i in $oldsk$. $c1_i = p_1 \cdot u_i + e2_i$. Set $evalKey = (\{c0_k\}, \{c1_k\})$ for $0 \leq k < v$ where v is the number of elements in $oldsk$. Output $evalKey$.
- **ReEncrypt**$(evalKey, ciphertext)$**:** Apply KeySwitch on $ciphertext$ with $evalKey$.
- **KeySwitch**$(evalKey, ciphertext)$**:** $evalKey = (b, a)$. Set $digitsC1$ as CRT-Decompose of $ciphertext[1]$ on base rw, $c_1 = digitsC1[0] \cdot a[0]$, $c_0 = c_0 + digitsC1[0] \cdot b[0]$. Set $c_0 = c_0 + \sum_{i=1}^{k-1}(digitsC1[i] \cdot b[i])$, $c_1 = \sum_{i=1}^{k-1}(digitsC1[i] \cdot a[i])$ where k is size of $digitsC1$. Output $newCiphertext = (c_0, c_1)$.

2.2 Intel SGX

The Trusted Execution Environment (TEE) is an approach for secure computation that enables the processing of sensitive data within the main processor's secure area (enclave). TEE delivers memory in secure enclaves for isolated computation in the presence of a malicious host. Other processes, such as user or kernel level operations, cannot modify the code contained within the enclave. SGX is a hardware-assisted version of TEE that is available on various Intel processors. SGX enables code to execute in a protected enclave that can communicate with other applications through a dedicated channel, but other applications cannot access the enclave itself [13]. Enclave execution takes place in the protected mode (at ring 3) and follows the address translation done by the operating system kernel [13]. SGX has remote attestation to prove the integrity of the code running in the enclave. When a malicious party attacks a system, it cannot access the code running/stored in the enclave. This reduces the attack surface of the system. SGX has also some disadvantages as follow.

- Paging cost: Data is encrypted and decrypted during exchanging the data between the enclave and outer program. This step creates latency during the paging process.
- Memory limit: SGX has physical memory limit of 128 MB while the practical limit is 90 MB [13,26].

– Lack of library support: Some C++ operations or libraries cannot be used in SGX such as vector from standard library and system calls.

In our study, SGX helps us to execute sensitive data in an untrusted server. In PREFHE, we have to use two servers to prevent disclosing of parties' data. In the PREFHE-SGX version, one server handles the tasks of two servers. When the server in PREFHE-SGX runs the sensitive data that helps decrypting the clients' encrypted data. In the new version, creating the common key pair and reencryption key that use the secret key of common key pair take place in the SGX enclave which cannot be tampered by the host server or any other malicious third parties.

Gramine: Gramine is an open-source, lightweight Lib OS(Library Operating System) project that supports Intel SGX and allows users to run their existing applications on Intel SGX [31]. Intel Labs initiated Gramine (previously called Graphene) to provide an open-source compatibility layer for Intel SGX. Gramine bridges various kind of applications and Intel SGX without modifying the application code. Gramine supports native Linux binaries on all platforms. We use Gramine to integrate PALISADE homomorphic encryption library and Intel SGX. Since Intel SGX does not allow usage of some libraries, Gramine handles this problem for developers. PALISADE, Gramine and Intel SGX are first used by Takeshita et al. [45] while we pioneer using this system for building secure MPC protocols.

3 Secure MPC Protocols

In this section, we introduce our three secure MPC protocols in detail.

3.1 PREFHE: Secure MPC from Multikey FHE and PRE

We construct a secure MPC protocol based on the BFVrns scheme which is a prominent lattice-based FHE scheme. Our protocol is constructed on two untrusted servers and n clients who want to run secure MPC on their secret data collabaratively. We utilize PRE approach to enable privacy of the inputs of the clients and the result of secure multiparty computation. PRE allows reencrypting different ciphertext under the same secret key. Indirectly, it also allows decrypting the same input under different secret keys. In the protocol, we utilize the PRE method proposed in [40] as a building block to manage two untrusted server setting. We assume that these two servers do not collude and the clients and S_1 use a secure channel to prevent S_2 to access any ciphertext which is encrypted under reencryption keys generated with the common key pair. We summarize how this protocol works step by step. Also, we provide the security analysis for semi-malicious model (a.k.a. honest-but-curious model).

1. The untrusted server S_2 decides on the FHE parameter set which is generated by the work [47] and verified by LWE Estimator [2] that is used in Homomorphic Encryption Standard [1]. S_2 decides on the client index j who initiates

Key Generation and S_2 creates the crytocontext cc and a common key pair $CKPair$ from cc to reencrypt the ciphertext under the same key pairs, publishes FHE parameters, $CKPair.pk$ and j to the clients as in Fig. 1.
2. The clients can check the security of the parameter set with LWE Estimator and if it does not provide necessary security limit (i.e. at least 128-bit security level), they can ask for S_2 to generate a new parameter set and publish it to all parties.
3. The client j initiates the *Key Generation* process and broadcasts his public key as pk_j.
4. Other clients operate *Multiparty Key Generation* using pk_j. Each party encrypts their data under their individual public key.
5. All clients operate *ReKey Generation* process to generate new encryption keys to enable S_1 to eval the function on the data.
6. The clients reencrypt their ciphertext under their new individual reencryption keys. They send these new ciphertexts $cptxtC_i$ to S_1. S_1 evaluates the function to be computed on these ciphertexts as in Fig. 2. S_1 does not know the secret key of the common key pair $CKPair$, thus S_1 cannot decrypt the ciphertexts of the clients.
7. As seen in Fig. 3, S_2 creates reencryption keys to allow the clients to decrypt the result under their individual secret keys. S_2 sends these keys $REKey_i$ to S_1 allowing that noone can see the result in plaintext including S_1 and S_2.
8. S_1 reencrypts the result under corresponding new reencryption keys of the clients and sends all r_i to corresponding public key holder clients where $1 \leq i \leq n$.
9. Each client decrypts their reencrypted ciphertext r_i under their individual secret keys and learns the result of the multiparty computation.

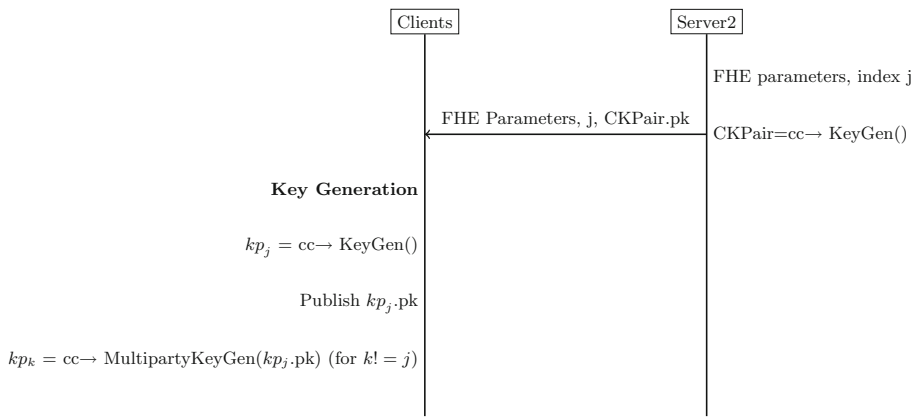

Fig. 1. Key generation and exchange phases of the PREFHE protocol.

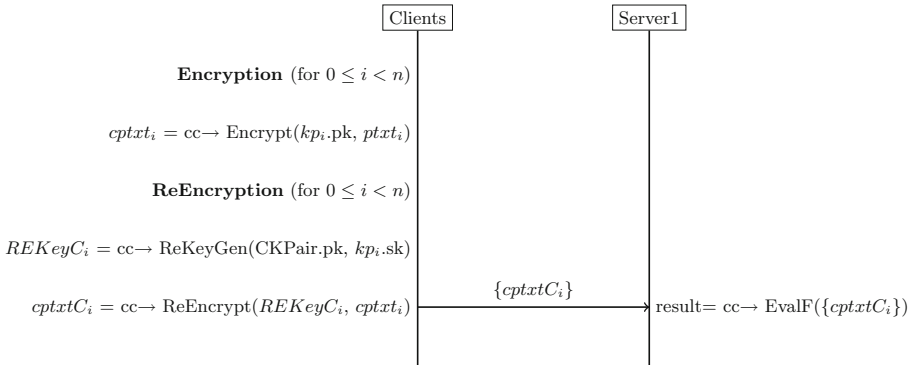

Fig. 2. Reencryption phase of PREFHE and PREFHE-SGX protocol.

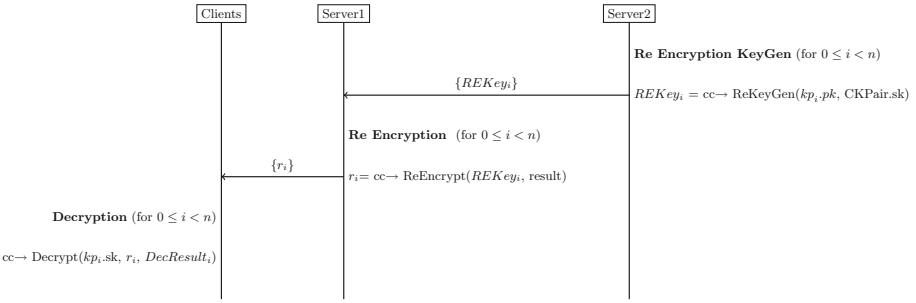

Fig. 3. Second reencryption and decryption of the PREFHE protocol.

3.2 PREFHE-AES: Secure MPC from Multikey FHE and PRE with AES

We combine the first version of our secure MPC protocol with AES-128 encryption method to reduce communication cost between the servers and clients. In the previous scheme, the clients need to send their ciphertexts that are encrypted under the BFVrns scheme. The ciphertext size in the FHE schemes is large such as $2N \log q$ bits where N is the ring dimension, $\log q$ is ciphertext modulus. To overcome this problem, the clients send their data encrypted under AES-128 encryption that results in a far smaller ciphertext sizes than the BFVrns encrypted one. The ciphertext size in AES-128 depends on the plaintext size, mode of operation and padding. The steps of the protocol are explained as follows:

1. The protocol starts with the key generation and key distribution as in Fig. 4. The FHE parameters are decided by S_2 as in PREFHE and distributed to the clients. At the same time, S_2 creates a cryptocontext cc and generates a common key pair $CKPair$ from cc to let the clients create their reencryption keys and reencrypt their ciphertexts.

2. The clients generate their public, private key pairs and encrypt their data under their AES-128 key as c_i as in Fig. 4.
3. The clients encrypt their AES key with their FHE public key as ck_i.
4. They create reencryption key from the common key pair as $REKP_i$ and send $REKP_i$, c_i and ck_i to S_1.
5. S_1 encrypts the ciphertexts with their FHE public keys and unravels the ciphertexts with homomorphic AES-128 decryption (AES^{-1}) as in Fig. 6.
6. S_1 reencrypts these ciphertexts again under their corresponding reencryption keys and evaluates the necessary function on the FHE ciphertexts.
7. S_2 generates the second reencryption key to enable the ciphertexts to be decrypted under individual private keys of the clients and sends $REKeyRC_i$ to S_1 as in Fig. 5.
8. S_1 reencrypts the result with their respective reencryption keys and conducts dimension reduction as in [8] to reduce the size of the ciphertexts, then S_1 sends these ciphertexts to the corresponding clients.
9. The clients decrypt their ciphertext with their private keys and get the result in plaintext.

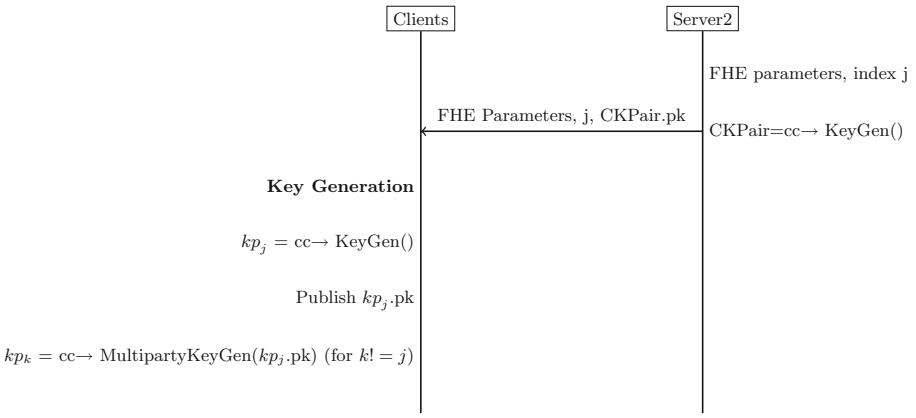

Fig. 4. Key generation and distribution of the PREFHE-AES protocol.

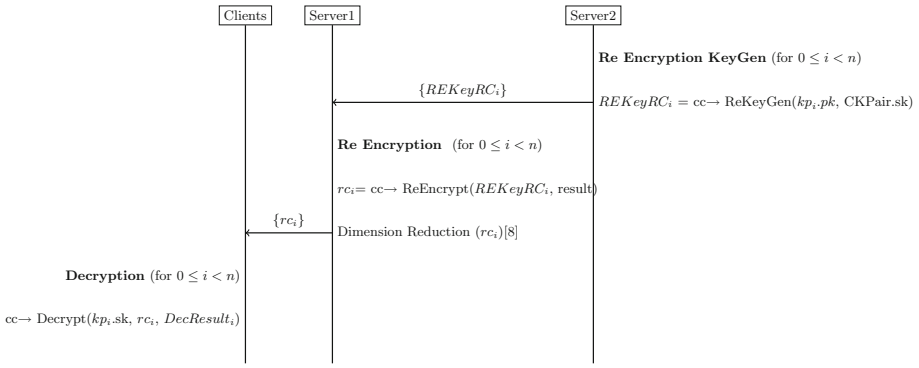

Fig. 5. The second reencryption and decryption phase of the PREFHE-AES protocol.

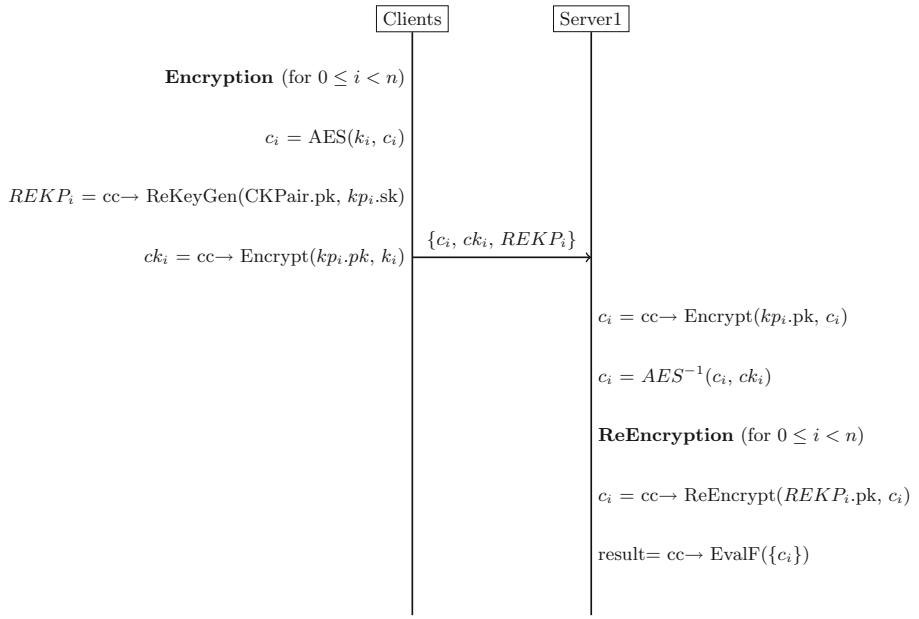

Fig. 6. The first reencryption and evaluation of function F phase of the PREFHE-AES protocol.

3.3 PREFHE-SGX: Secure MPC from Multikey FHE and PRE with SGX

In this section, we present PREFHE-SGX that enables constructing the PREFHE protocol with one-server setting. Intel SGX enables calculation of sensitive data in the SGX enclave by protecting the data from the outer applications or adversaries. The difference between this protocol and PREFHE is calculating

common key pair and reencryption keys in the SGX enclave. In PREFHE, S_2 conducts these operations separately to prevent the clients' data to be seen in plaintext by S_1.

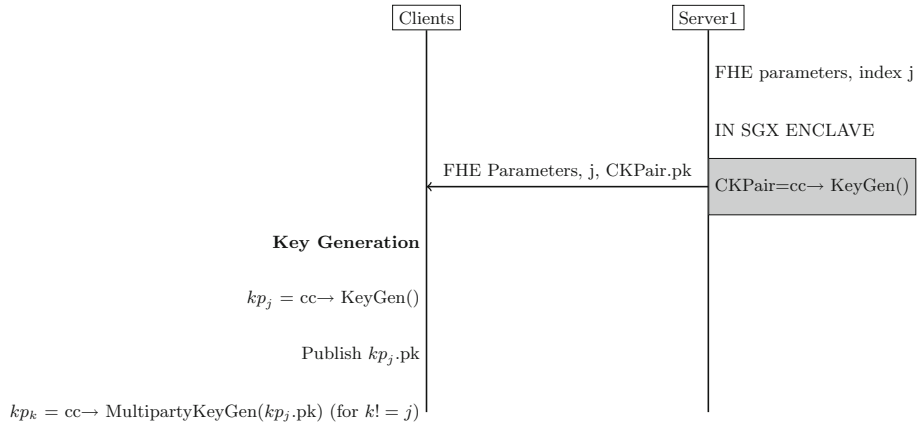

Fig. 7. Key exchange phase of the PREFHE-SGX protocol. (Grey box represents the SGX enclave.)

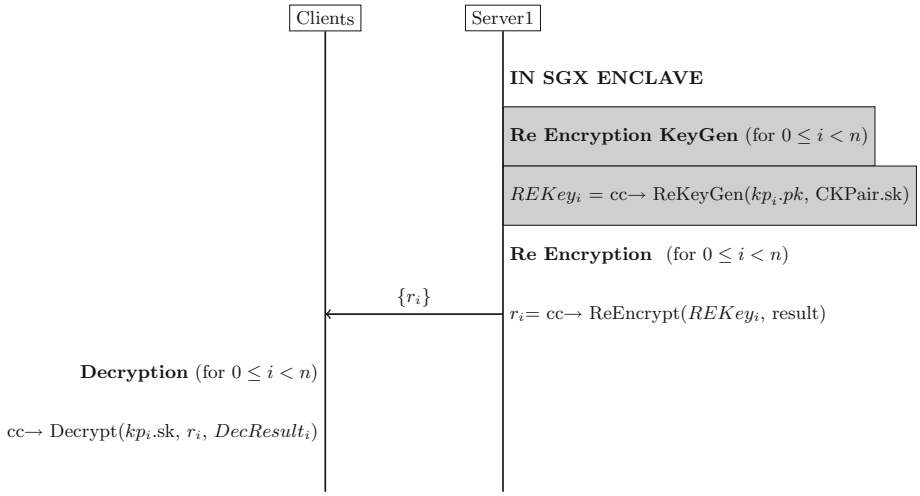

Fig. 8. Second reencryption and decryption of the PREFHE-SGX protocol. (Grey box represents the SGX enclave.)

With SGX, the secret key of the common key pair is stored in the SGX enclave. The operations which use this secret key also locate in the SGX enclave so that the server cannot decrypt the clients' encrypted data. We refer the reader to the explanation of PREFHE for further details due to space limit. In this protocol, Gramine allows us to run PALISADE code on the SGX enclave without any adjustments on the code. Figure 7, 2 and 8 depict the steps of the protocol in order.

3.4 Correctness

Correctness of the protocols mainly depends on the BFVrns scheme given in Sect. 2.1. Halevi et al. provides correctness proof of the scheme in [29]. We present the correctness of the PREFHE protocol in this section. Other two protocols follow the same correctness proof.

At the beginning of the protocol, *KeyGeneration* step creates the public, private key pair for the encryption step. After this step, we make use of PRE scheme that proposed in [40]. In this step, PRE allows encrypting of all ciphertexts under the same key pair to evaluate a function on them. PRE has *ReKeyGen* routine that creates new *evalKey* for transforming the ciphertext to another one. The second step of PRE is *ReEncryption* step which calls *KeySwitch* routine due to being a PRE operation. *KeySwitch* changes the given input ciphertext to another ciphertext that is encrypted under the new key given in *ReKeyGen* step. *KeySwitch* method uses the algorithm given in [8] as digit decomposition method. Correctness of PRE scheme is proved in [40]. In this step all ciphertexts are transformed to be encrypted under the same public key. The corresponding secret key of this public key is only known by S_2. All ciphertext are sent to S_1 to evaluate the function which clients want to calculate. S_1 evaluates the function on the ciphertexts. S_2 runs *ReKeyGen* routine to create new *evalKey* set for reencrypting the result to be decrypted under each clients' secret key. S_1 runs *ReEncryption* and sends the results to corresponding clients. Correctness of this step also depends on the PRE scheme. Clients decrypt these new individual ciphertexts with their secret key and open the result in plaintext. Correctness of this protocol mainly depends on PRE which is used two times as a subroutine.

The final decryption of ciphertext after reencryption can be represented as follows where $pk = (p0, p1)$ and $b = p0 \cdot u[i] + e1[i] + s[i]$ from ReKeyGen procedure:

$$\begin{aligned} c_0 - s \cdot c1 &= c0 + \sum_{i=1}^{k-1}(digitsC1[i] \cdot b[i]) - s \cdot \sum_{i=1}^{k-1}(digitsC1[i] \cdot a[i]) \\ &= c0 + \sum_{i=1}^{k-1}(digitsC1[i] \cdot (p0 \cdot u[i] + e1[i] + s[i])) - s \cdot \sum_{i=1}^{k-1}(digitsC1[i] \cdot a[i]). \end{aligned} \tag{1}$$

After scaling down by t/q;

$$\lceil (c_0 - s * c_1)t/q \rfloor = \lceil t/q(c_0 - s \cdot \sum_{i=1}^{k-1}(digitsC1[i] \cdot a[i]) + \sum_{i=1}^{k-1}(digitsC1[i] \cdot (p0 \cdot u[i] + e1[i] + s[i]))) \rfloor. \tag{2}$$

$\lceil \sum_{i=1}^{k-1}(digitsC1[i] \cdot ((-e - as) \cdot u[i] + e1[i] + s[i]))t/q \rfloor$ is small enough due to $q \ll t$. It can be seen that

$$\lceil (c_0 - s \cdot c_1)t/q \rfloor = m \pmod{t}. \tag{3}$$

3.5 Security Analysis

Security for the Semi-honest Model: The semi-honest model implies that all parties follow the protocol description, but they still try to gather information about other parties' inputs, intermediate results or overall outputs just by looking at the protocol's transcripts.

Security of the PREFHE protocol mainly depends on underlying FHE scheme of the protocol and BFVrns depends on RLWE assumption as follows.

Definition 1. ***(RLWE** [44]): For security parameter λ, $f(x) = x^n + 1$ where n is power of 2. q is $q \geq 2$. Let the ring $\mathcal{R} = \mathbb{Z}[X]/f(x)$ and $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$. χ is a distribution over $\mathcal{R}_q$. $RLWE_{Q,\chi,n}$ problem concerns about distinguishing following two distributions. The first distribution is uniformly generated samples $(a_i, b_i) \in \mathcal{R}_q^2$. In the second distribution, samples s from $\mathcal{R}_q$ uniformly, $(a_i, b_i) \in \mathcal{R}_q^2$ where $a_i \leftarrow \mathcal{R}_q$ uniformly and $e_i \leftarrow \chi$, $b_i = a_i \cdot s + e_i$. Since SVP problem can be reduced to RLWE [35], RLWE is considered a hard problem.*

KeyGen/MultipartyKeyGen: This step creates random sk and pk pairs for the clients and its security depends on RLWE assumption.

Encryption: The clients encrypt their inputs under their public key so noone else can decrypt and see their input.

ReKeyGen + ReEncrypt: This step depends on PRE scheme proposed in [40] which is proved as IND-CPA secure in [40]. The clients create new $evalKey$ to reencrypt their encrypted inputs to allow function evaluations on all of the clients' data. This step requires individual sk so that noone else can create other $evalkey$ from their sk. These reencrypted ciphertexts are sent to only S_1 through a secure channel to prevent any decryption by S_2. According to our assumption on two non-colluding servers, S_1 cannot decrypt ciphertexts sent by the clients and result of function evaluation on ciphertexts because it has no access to $CKPair.sk$. Also, S_2 cannot decrypt or manipulate ciphertexts because it cannot see clients' ciphertexts or the final result. S_1 cannot get any additional information from intermediate outputs. *ReKeyGen + ReEncrypt* is used in 2., 3. and 4. rounds to prevent leaking any additional information about the ciphertexts and result.

When all rounds come together, the PREFHE protocol ensures security in the semi-honest model. We can improve the security model to malicious model with NIZK protocols to prove plaintext knowledge as in the work proposed in [34].

4 Software Implementation

4.1 Implementation on PALISADE

Experimental Setup. We run the experiments on the Microsoft Azure Standard DC2s v2 virtual machine that has 2 cores and 8 GB memory running Ubuntu 20.04. We use PALISADE lattice cryptography library version 1.10.6 [39] and Gramine version 1.1[2].

4.2 Perfomance

In this section, we analyze our secure MPC protocols in terms of three metrics such as rounds, communication and computation complexity and compare with the state-of-the-art protocols. Due to the unavailability of their implementations, it is not possible to compare running times with other suggested protocols. We compare the protocols such as [3,33,34,38] which provide semi-malicious security model version to provide a fair comparison.

Rounds: PREFHE and PREFHE-AES propose 4 rounds while PREFHE-SGX has 3 rounds in total. Each round in our protocols requires less computation than the rounds in [33,34]. Our protocols do not require the parties to be online after they send their encrypted data to the server. On the other hand, some cutting-edge protocols require communication at every gate and the presence of all parties online [5,16,18] which is not feasible in real-life applications. In some protocols, decryption is jointly computed [34] that increases the number of rounds. In the work presented in [33], encryption key is jointly computed while in our protocols, this is not required which means more practicality. In our protocols, Table 1 suggests that PREFHE-SGX has the less number of rounds. In PREFHE-SGX, one server handles two servers' jobs, so it reduces the number of total rounds.

Communication Complexity: The communication cost between clients and the server is independent of the function to be computed. In PREFHE and PREFHE-AES, the communication cost between two servers is also independent of the complexity of the function. The works in the [3,33], communication cost depends on the length of the input and outputs. They generally contain a set of indices, ciphertexts and eval keys which is larger than a ciphertext size.

Computation Complexity: In our protocols, the computation complexity on the server S_1 is linear in the size of the circuit computing F. Since multiplication of two ciphertexts is considered the most costly operation, the multiplicative depth of the function mainly determines the computation cost.

Performance Results: Table 2 suggests that PREFHE has the fastest client and servers time with 2 untrusted server setting. Since homomorphic decryption of AES-128 takes a long time, Server 1 computation time of PREFHE-AES is around 45 s which is still practical for real life. For functions having larger number of multiplications, computation cost can be improved with better hardware

[2] https://github.com/gramineproject/gramine.

Table 1. Performance comparison of main and our protocols. (Communication complexity refers to the communication cost between a client/party and server that evaluates the function F. $\|c\|$ represents size of a ciphertext and $\|c_{AES}\|$ stands for AES-128 encrypted ciphertext size.)

	[33]	[38]	[3]	PREFHE	PREFHE-AES	PREFHE-SGX
Rounds	4	4	5	4	4	3
Communication Comp.	$\|I/O\|$	$\|c\|$	$\|I/O\|$	$\|c\|$	$\|c_{AES}\|$	$\|c\|$
Computation Comp. (Server)	$\|F\|$	$\|F\|$	$\|F\|$	$\|F\|$	$\|F\|$	$\|F\|$

Table 2. Performance results of the protocols for $t = 32769$, $m = 16384$, $\log q_i = 55$, $\sigma = 3$, $rw = 0$, $\lambda = 128$. F has one multiplication as an example in this experiment. Time unit is ms. Client time represents the total runtime of one client.

	PREFHE	PREFHE-AES	PREFHE-SGX
Client time	24.931	12.276	26.354
Server 1 time	19.183	45142.833	25836.862
Server 2 time	27.701	29.975	NA

and parallelization techniques. The client side of PREFHE-AES has better performance over other two protocols due to outsourcing the encryption of the data to Server 1. PREFHE-SGX performs better than PREFHE-AES for Server 1 but worse than PREFHE. The reason is that Server 1 in PREFHE-SGX handles two servers' tasks and it utilizes SGX that has paging latency.

Trade-Off Between Protocols: According to the needs of the application, the user may consider trade-off between client and server side computations or rounds or communication cost. For applications that have network bandwidth constraints, the user may prefer PREFHE-AES over others. For computational constraints or time sensitivity, PREFHE is the best fit with the short latency. For the systems that allow one server in the secure MPC setting, PREFHE-SGX signifies the best round-efficient one in all secure MPC protocols.

5 Conclusion

We propose three distinct secure MPC protocols constructed from FHE, AES-128, and Intel SGX. PREFHE is highly efficient in real-world applications, whereas PREFHE-AES introduces a communication-efficient protocol that is highly efficient in low-bandwidth networks. PREFHE-SGX proposes a single-server setting with 3 rounds and is a pioneer in the use of FHE and SGX in secure MPC protocols. Our protocols' communication costs are function-independent. Additionally, our protocols do not require parties to be online following the transmission of encrypted data to Server 1. The decryption phase does not require any collaboration on the part of the parties, which increases the protocols' practicality. We present efficient and secure MPC protocols that are applicable to a variety of use cases in real life.

References

1. Albrecht, M., et al.: Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018
2. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Math. Cryptol. **9**(3), 169–203 (2015)
3. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 483–501. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_29
4. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali, pp. 351–371 (2019)
5. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 169–188. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_11
6. Bogetoft, P., et al.: Secure multiparty computation goes live. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 325–343. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03549-4_20
7. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 868–886. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_50
8. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. Cryptology ePrint Archive, Report 2011/344 (2011). https://eprint.iacr.org/2011/344
9. Catrina, O., Kerschbaum, F.: Fostering the uptake of secure multiparty computation in e-commerce. In: 2008 Third International Conference on Availability, Reliability and Security, pp. 693–700. IEEE (2008)
10. Chaum, D., Crépeau, C., Damgard, I.: Multiparty unconditionally secure protocols. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, pp. 11–19 (1988)
11. Choi, S.G., Elbaz, A., Juels, A., Malkin, T., Yung, M.: Two-party computing with encrypted data. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 298–314. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76900-2_18
12. Choudhury, A., Loftus, J., Orsini, E., Patra, A., Smart, N.P.: Between a rock and a hard place: interpolating between MPC and FHE. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013. LNCS, vol. 8270, pp. 221–240. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-42045-0_12
13. Costan, V., Devadas, S.: Intel SGX explained. Cryptology ePrint Archive (2016)
14. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 280–300. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44987-6_18
15. Damgård, I., Ishai, Y.: Constant-round multiparty computation using a black-box pseudorandom generator. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 378–394. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_23

16. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority – or: breaking the SPDZ limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 1–18. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40203-6_1
17. Damgård, I., Orlandi, C.: Multiparty computation for dishonest majority: from passive to active security at low cost. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 558–576. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_30
18. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_38
19. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_2
20. Erkin, Z., Franz, M., Guajardo, J., Katzenbeisser, S., Lagendijk, I., Toft, T.: Privacy-preserving face recognition. In: Goldberg, I., Atallah, M.J. (eds.) PETS 2009. LNCS, vol. 5672, pp. 235–253. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03168-7_14
21. Erkin, Z., Veugen, T., Toft, T., Lagendijk, R.L.: Generating private recommendations efficiently using homomorphic encryption and data packing. IEEE Trans. Inf. Forensics Secur. **7**(3), 1053–1066 (2012)
22. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012). https://eprint.iacr.org/2012/144
23. Garg, S., Gentry, C., Halevi, S., Raykova, M.: Two-round secure MPC from indistinguishability obfuscation (2014)
24. Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 465–482. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_28
25. Gentry, C., et al.: Fully homomorphic encryption using ideal lattices. In: Stoc, pp. 169–178 (2009)
26. Gjerdrum, A.T., Pettersen, R., Johansen, H.D., Johansen, D.: Performance of trusted computing in cloud infrastructures with intel SGX. In: CLOSER, pp. 668–675 (2017)
27. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game, or a completeness theorem for protocols with honest majority. In: Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali, pp. 307–328 (2019)
28. Halevi, S., Lindell, Y., Pinkas, B.: Secure computation on the web: computing without simultaneous interaction. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 132–150. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_8
29. Halevi, S., Polyakov, Y., Shoup, V.: An improved RNS variant of the BFV homomorphic encryption scheme. In: Matsui, M. (ed.) CT-RSA 2019. LNCS, vol. 11405, pp. 83–105. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-12612-4_5
30. Kamara, S., Mohassel, P., Raykova, M.: Outsourcing multi-party computation. Cryptology ePrint Archive (2011)
31. Kuvaiskii, D., Kumar, G., Vij, M.: Computation offloading to hardware accelerators in intel SGX and Gramine library OS. arXiv preprint arXiv:2203.01813 (2022)

32. Lindell, Y., Pinkas, B.: A proof of security of Yao's protocol for two-party computation. J. Cryptol. **22**(2), 161–188 (2009)
33. López-Alt, A., Tromer, E., Vaikuntanathan, V.: Cloud-assisted multiparty computation from fully homomorphic encryption. Cryptology ePrint Archive (2011)
34. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing, pp. 1219–1234 (2012)
35. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_1
36. 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)
37. Natarajan, D., Dai, W., Dreslinski, R.: CHEX-MIX: combining homomorphic encryption with trusted execution environments for two-party oblivious inference in the cloud. Cryptology ePrint Archive, Paper 2021/1603 (2021). https://eprint.iacr.org/2021/1603
38. Peter, A., Tews, E., Katzenbeisser, S.: Efficiently outsourcing multiparty computation under multiple keys. IEEE Trans. Inf. Forensics Secur. **8**(12), 2046–2058 (2013)
39. Polyakov, Y., Rohloff, K., Ryan, G.W.: Palisade lattice cryptography library (2018). https://palisade-crypto.org/
40. Polyakov, Y., Rohloff, K., Sahu, G., Vaikuntanathan, V.: Fast proxy re-encryption for publish/subscribe systems. ACM Trans. Privacy Secur. (TOPS) **20**(4), 1–31 (2017)
41. Rivest, R.L., Adleman, L., Dertouzos, M.L., et al.: On data banks and privacy homomorphisms. Found. Secure Comput. **4**(11), 169–180 (1978)
42. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. Des. Codes Crypt. **71**(1), 57–81 (2014)
43. Stehlé, D., Steinfeld, R.: Faster fully homomorphic encryption. In: Proceedings of ASIACRYPT 2010, pp. 377–394 (2010)
44. Stehlé, D., Steinfeld, R., Tanaka, K., Xagawa, K.: Efficient public key encryption based on ideal lattices. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 617–635. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_36
45. Takeshita, J., McKechney, C., Pajak, J., Papadimitriou, A., Karl, R., Jung, T.: GPS: integration of graphene, palisade, and SGX for large-scale aggregations of distributed data. Cryptology ePrint Archive (2021)
46. Wu, P., Ning, J., Shen, J., Wang, H., Chang, E.C.: Hybrid trust multi-party computation with trusted execution environment. In: The Network and Distributed System Security (NDSS) Symposium 2022 (2022)
47. Yakupoglu, C., Kurt, R.: Parameter selection for computationally efficient use of the BFVRNS FHE scheme. Under review (2022)
48. Yao, A.C.C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (SFCS 1986), pp. 162–167. IEEE (1986)