applied sciences

MDPI

Article

LPCP: An efficient Privacy-Preserving Protocol for Polynomial Calculation Based on CRT

Jiajian Tang, Zhenfu Cao, Jiachen Shen * and Xiaolei Dong

Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai 200062, China; 51194501014@stu.ecnu.edu.cn (J.T.); zfcao@sei.ecnu.edu.cn (Z.C.); dongxiaolei@sei.ecnu.edu.cn (X.D.)
* Correspondence: jcshen@sei.ecnu.edu.cn

Abstract: With the development of privacy-preserving techniques, the increasing demand for secure multiparty computation (MPC) of mobile devices has become a significant challenge. Unfortunately, it is inapplicable for mobile devices to implement the existing secure multiparty computation schemes that rely on costly computation and communication overhead. To solve this problem, we propose an efficient two-party computation protocol secure against semi-honest adversary based on the Chinese remainder theorem (CRT). Our protocol utilizes CRT-based encryption and re-encryption techniques to realize additional and multiplicative homomorphic encryption, which can be transformed into a two-party secure computation scheme. Then, we extend our two-party LPCP protocol into a multiparty LPCP protocol, which is much faster and more space saving than the previous works. For practical purpose, we describe a distance measurement application for mobile devices based on LPCP. In the end, we implement LPCP codes and compare the experimental results to the state-of-the-art two-party and multiparty computation protocols. The experimental result shows that the high computation and communication efficiency of LPCP makes it possible for low computing-power mobile devices to implement multiparty secure computation protocols in reality.

Keywords: secure multiparty computation; semi-honest adversary; Chinese remainder theorem

Citation: Tang, J.; Cao, Z.; Shen, J.; Dong, X. LPCP: An efficient Privacy-Preserving Protocol for Polynomial Calculation Based on CRT. *Appl. Sci.* **2022**, *12*, 3117. https://doi.org/10.3390/app12063117

Academic Editors: Konstantinos Demertzis, Konstantinos Rantos and George Drosatos

Received: 25 January 2022
Accepted: 16 March 2022
Published: 18 March 2022

1. Introduction

Multiparty computation (MPC) [1] allows a group of parties to compute a pre-defined function jointly without revealing their input information. MPC was initially introduced by Yao [2] in the 1980s. After decades of development, MPC has overcome many theoretical bottlenecks [3] and come into use in many areas, such as privacy-preserving machine learning [4,5] and cloud data storage [6]. In particular, two-party computation (2PC) plays a fundamental but indispensable role in the development of MPC because many 2PC schemes can be easily transformed into MPC schemes [7]. So we mainly focus on 2PC in this paper. The traditional 2PC schemes strongly rely on advanced computing power and large storage space machines. In the latest 2PC work by Lindell [8], their 2PC experiments are executed on two Amazon AWS instances: one is c4.8xlarge with 36 virtual 2.9 GHz CPUs and 64 GB RAM, and the other is c4.2xlarge with 8 virtual 2.9 GHz CPUs and 15 GB RAM, which is inapplicable for mobile devices. Although many optimization methods, such as amortization, pre-processing and round optimization, have been employed to decrease the computational and communication complexity in recent works [9–11], these improved MPC schemes are still far from daily application on mobile devices. Therefore, it is necessary to introduce a computing-efficient and communication-efficient MPC scheme for mobile devices with low computing ability.

In this paper, we propose an efficient MPC scheme called the lightning polynomial computation protocol (LPCP), which solves the problems mentioned above. Our starting point is a fully homomorphic encryption (FHE) [12], as two-party secure computation schemes can be constructed from a FHE scheme. However, the existing FHE-based 2PC

schemes, such as threshold FHE [13,14] and multi-key FHE [15,16], do not meet our computational and communication efficiency demand. Inspired by Zhou's [17] outsourced secure computation scheme LPPA, we find a delicate way to boost the execution process of MPC. In the LPPA paper, Zhou employed the Chinese remainder theorem (CRT) to construct an online computation scheme that allows additive homomorphic and multiplicative homomorphic operations with the help of a trusted third party and a double server. The advantage of LPPA is that the low computational and space requirement can be easily met by mobile devices, while the disadvantage is that all the computation tasks are assigned to a trusted third party and a double server, which are not allowed to exist in MPC cases. To construct a secure MPC scheme, we aim to remove the trusted third party and the double server. Therefore, we reassign the tasks of the trusted third party and double server directly to the two participant parties. The tasks of the double server are separated apart so that the original re-encryption task is assigned to the first party, and the original evaluation task is assigned to the second party. In the meanwhile, our modifications do not increase the computational and communication complexity. As a result, a 2PC protocol is accomplished without the double server and a trusted third party. Furthermore, we can transform our 2PC protocol into a MPC protocol in a recursive way.

1.1. Our Contributions

In this paper, we propose an efficient CRT-based MPC scheme for polynomial computations, which is secure against a semi-honest adversary. As far as we know, our LPCP scheme is the first protocol that introduces the Chinese remainder theorem encryption [18] and the re-encryption techniques into an MPC scheme. The computational complexity of LPCP is reduced to $\mathcal{O}(1) \cdot T_{trapdoor} + \mathcal{O}(max(log_2^{p+q}, term_F)) \cdot T_{multiplication} + \mathcal{O}(1) \cdot T_{hash}$, where (p, q) is the big prime parameters, $term_F$ is the polynomial term number, $T_{trapdoor}$ is the time of executing a trapdoor permutation encryption, $T_{multiplication}$ is the time of executing multiplication arithmetic operations, and T_{hash} is the time of executing a hash function. The communication complexity of LPCP is reduced to $\mathcal{O}(1) \cdot L_{trapdoor} + \mathcal{O}(1) \cdot L_{ciphertext} + \mathcal{O}(1) \cdot L_{hash}$, where $L_{trapdoor}$is the length of the trapdoor permutation ciphertext, $L_{ciphertext}$ is the length of the ciphertext, and L_{hash} is the length of the hash value. For two-party secure computation, compared to the latest two-party secure computation protocol Emp-sh2pc [19], our experimental result is more than 100 times faster than Emp-sh2pc in 128-bit and 256-bit additive and multiplicative tests, and more than 100,000 times faster in the 1024-bit modular exponential tests. For multiparty secure computation, our experimental result is much more scalable than those semi-honest protocols from MP-SPDZ [20]. Both the computational and communication overhead of LPCP is affordable for mobile devices with low computation power. As a result, it is feasible for privacy-preserving mobile devices to implement two-party secure computation with our protocol LPCP.

1.2. Related Works

In 1978, Rivest, Adleman and Dertouzos [21] first introduced privacy homomorphism that allows computation on encrypted data without decryption, and they give a CRT-based example. In 2009, Gentry [22] proposed the first concrete fully homomorphic encryption scheme based on ideal lattices. Based on the learning-with-error (LWE) assumption, Cheon [23] proposed a widely used FHE scheme called CKKS, which has been realized in many cryptography libraries. As the fully homomorphic encryption scheme can directly perform the computations on the ciphertexts without decryption, it brings many MPC ideas into being. López-Alt [14] proposed a new MPC scheme based on FHE. In the CRT fields, Kim [18] introduced a CRT-based fully homomorphic encryption scheme over the integers, which is secure against the chosen plaintext attacks under the decisional approximate greatest common divisor (AGCD) assumption and the sparse subset sum assumption. Zhou [17] proposed a CRT-based secure outsourced computation scheme that realizes fully the homomorphic property in a re-encryption way.

1.3. Orgnization

The paper is arranged as follows. Firstly, we introduce the preliminaries in Section 2. Then, the framework for two-party LPCP is described in Section 3. In Section 4, we give the flow chart of LPCP and present the concrete construction of LPCP for a two-party computation situation. In Section 5, we prove the correctness of LPCP and the privacy security of LPCP in the hybrid simulation. In Section 6, the original two-party construction of LPCP is extended into three or more party secure computation schemes. In Section 7, we describe a mobile device application scene based on our LPCP scheme. In Section 8, we evaluate the performance of our protocol in the aspects of running time and communication overhead. Then, we compare the result of our experiment to the latest works. Finally, we make a conclusion of the advantages and disadvantages of LPCP in Section 9.

2. Preliminaries

2.1. One-Way Trapdoor Permutation

One-way trapdoor permutation is a triple of function $\{f, f^{-1}, t\}$, where f is a one-way function and t is secret information generated by security parameter λ. Furthermore, function $f : f(x) \to x$ is also a permutation, where $x \in D(1^\lambda)$. There exists a polynomial algorithm that correctly computes function f^{-1} with the trapdoor t. For all probabilistic polynomial algorithms without the trapdoor t, there exists

$$Pr[\mathcal{A}(1^\lambda, f(x)) = f^{-1}(f(x))] \leq \epsilon(\lambda) \tag{1}$$

where $x \in D(1^\lambda)$ and $\epsilon(\lambda)$ are negligible in λ.

2.2. Euler'S Theorem

Let n and a be coprime positive integers, that is, $gcd(n, a) = 1$, then

$$a^{\varphi(n)} \equiv 1 \bmod n \tag{2}$$

where φ is Euler's totient function.

2.3. Chinese Remainder Theorem (CRT)

The Chinese remainder theorem asserts that if positive integers $m_1, m_2, \ldots, m_k$ are pairwise coprime, $gcd(m_i, m_j) = 1$ for $i \neq j$ and every integer $a_1, a_2, \ldots, a_n$ satisfies $0 \leq a_i < m_i$, there exists one and only one solution x such that $0 \leq x < m$ to the following congruence.

$$\begin{aligned} x &\equiv a_1 \bmod m_1, \\ x &\equiv a_2 \bmod m_2, \\ &\vdots \\ x &\equiv a_k \bmod m_k \end{aligned} \tag{3}$$

with $m = \prod_{i=1}^{k} m_i$, $m = m_i M_i$ and $M_i M_i' \equiv 1 \bmod m_i$, the solution $x = M_1' M_1 b_1 + M_2' M_2 b_2 + \cdots + M_k' M_k b_k \bmod m$.

2.4. Hash Function

Hash function $\mathcal{H}$ is a function which maps data of arbitrary size to fixed-size values $\{0,1\}^\lambda \to \{0,1\}^\lambda$. A good hash function should be collision resistant such that for all probabilistic polynomial-time adversaries $\mathcal{A}$, $\mathcal{A}$ cannot find a collision $x \neq x'$ such that $\mathcal{H}(x) = \mathcal{H}(x')$ with non-negligible probability.

3. Framework for Two-Party LPCP

In this section, we introduce the framework for our semi-honest two-party secure computation protocol LPCP.

First of all, we define the security model of our two-party LPCP scheme. Considering the following situation, two semi-honest parties P_1 and P_2 intend to calculate a polynomial function $F(m_1, m_2)$ with inputs m_1 and m_2, respectively. For convenience, we use $\mathcal{S}$ to denote sender P_1 and $\mathcal{R}$ to denote receiver P_2 in this section. At the beginning of the protocol execution, one party is statically corrupted by the adversary. The corrupted party is allowed to learn as much information as possible, but cannot violate the protocol rules, which meets the definition of a semi-honest adversary.

To reduce the time and communication complexity, we abandon those widely used techniques, such as garbled circuits, oblivious transfer and secret sharing. Instead, LPCP implements the Chinese remainder theorem and re-encryption technique to realize additive and multiplicative homomorphism. The main idea of LPCP is that two parties encrypt their plaintexts with different secret keys, initially. Then, one party re-encrypts the other party's ciphertext, transforming both parties' ciphertext under the same secret key, where the computation process can be executed quickly. Finally, both parties decrypt the computed ciphertext to obtain the resulting answer. For concrete proof of the homomorphism, see Proof 4. LPCP consists of five ideal functionalities—the setup functionality $\mathcal{F}_{Setup}$, the key generation functionality$\mathcal{F}_{KeyGen}$, the encryption functionality $\mathcal{F}_{Enc}$, the re-encryption functionality $\mathcal{F}_{Re-Enc}$, the evaluation functionality $\mathcal{F}_{Eval}$, and the decryption functionality $\mathcal{F}_{Dec}$.

In $\mathcal{F}_{Setup}$, two parties jointly decide the security parameter λ and the polynomial function $F(m_1, m_2)$. The polynomial function $F(m_1, m_2) = \sum_i a_i m_1^{b_i} m_2^{c_i}$ takes in m_1 and m_2, and is expected to output the correct result. All the coefficients a_i and degrees b_i, c_i of polynomial terms are decided by $\mathcal{S}$ and $\mathcal{R}$ in advance.

Ideal Functionality $\mathcal{F}_{Setup}$

1. $\mathcal{S}$ and $\mathcal{R}$ run $para \leftarrow Setup(1^\lambda)$.
2. $\mathcal{S}$ and $\mathcal{R}$ decide the polynomial function $F = \sum_{i=1}^{n} a_i m_1^{b_i} m_2^{c_i}$.

Once the initial parameter *para* and the polynomial function are decided, both $\mathcal{S}$ and $\mathcal{R}$ begin to generate their private keys according to the parameter *para* given in the functionality F_{KeyGen}. It is important to remember that our protocol implements symmetric encryption instead of asymmetric encryption. Therefore, the private key generated should be kept secret by each party respectively. Additionally, the randomness parameter should not be leaked as well.

Ideal Functionality $\mathcal{F}_{KeyGen}$

1. $\mathcal{S}$ and $\mathcal{R}$ call the parameter *para* to generate their secret keys *sk* and randomness parameters *r* by running $\{sk, r\} \leftarrow KeyGen(para)$.

In the encryption phase, $\mathcal{S}$ encrypts its message m_1 with its private key sk_1 and a random element *r* generated in $\mathcal{F}_{KeyGen}$. Receiver $\mathcal{R}$ encrypts its message in the same way as sender $\mathcal{S}$, except that $\mathcal{R}$ does not multiply a random element with its ciphertext.

Ideal Functionality $\mathcal{F}_{Enc}$

1. $\mathcal{S}$ and $\mathcal{R}$ encrypt their messages respectively by running $C \leftarrow Enc_{sk}(m, r)$.
2. $\mathcal{S}$ sends its ciphertext to $\mathcal{R}$.

As mentioned above, sender $\mathcal{S}$ and receiver $\mathcal{R}$ encrypt their messages with different private keys and different randomness tuples, which results in their ciphertexts being in two distinct ciphertext spaces. Direct addition and multiplication on such two ciphertexts result in a wrong answer. That is why we introduce the re-encryption functionality $\mathcal{F}_{Re-Encryption}$. The re-encryption functionality aims to make sure that *mathcalS*'s ciphertext is correctly calculated with receiver $\mathcal{R}$'s ciphertext without leaking secret information.

Ideal Functionality $\mathcal{F}_{Re-Enc}$

1. Receiver $\mathcal{R}$ receives the original ciphertext *C* from sender $\mathcal{S}$.
2. Receiver $\mathcal{R}$ partially decrypts the ciphertext *C* in order to obtain the intermediate ciphertext $C_{intermediate}$.

3. Receiver $\mathcal{R}$ re-encrypts the intermediate ciphertext $C_{intermediate}$ to obtain the sender $\mathcal{S}$'s ciphertext under the receiver $\mathcal{R}$'s secret key by running $C' \leftarrow Re - Enc_{sk}(C_{intermediate})$.
4. Receiver $\mathcal{R}$ sends back the re-encryption result C' to sender $\mathcal{S}$.

The evaluation functionality is the key step where sender $\mathcal{S}$ finishes the calculation. The ciphertexts are transformed into the same ciphertext space; however, with different polynomial degrees. It is necessary for sender $\mathcal{S}$ to unify the degree of every term in the polynomial before the evaluation. After that, sender $\mathcal{S}$ calculates the sum of every polynomial term, finishing the evaluation step.

Ideal Functionality $\mathcal{F}_{Eval}$

1. Sender $\mathcal{S}$ receives $\mathcal{R}$'s ciphertext C together with the re-encryption ciphertext C' and reduces the initial randomness r from its re-encryption ciphertext.
2. Sender $\mathcal{S}$ carries out the computation function $C_F \leftarrow Eval(C, C')$.
3. Sender $\mathcal{S}$ transfers C_F to receiver $\mathcal{R}$.

In the end, we reach the final step—the decryption functionality. Receiver $\mathcal{R}$, who owns the private key sk_2, implements the decryption and sends the output back to $\mathcal{S}$.

Ideal Functionality $\mathcal{F}_{Dec}$

1. Receiver $\mathcal{R}$ runs the decryption functionality to obtain $Output \leftarrow Dec_{sk,r}(C_F)$.
2. Receiver $\mathcal{R}$ shares the $Output$ to $\mathcal{S}$.

In a nutshell, sender $\mathcal{S}$ appends a random integer r to its message m_1 and encrypts them with $\mathcal{S}$'s private key. Receiver $\mathcal{R}$ encrypts its message m_2 with $\mathcal{R}$'s private key. Up to now, the two ciphertexts from $\mathcal{S}$ and $\mathcal{R}$ lie in the different ciphertext space. Then, $\mathcal{S}$ sends its ciphertext to $\mathcal{R}$. $\mathcal{R}$ re-encrypts $\mathcal{S}$'s ciphertext with $\mathcal{R}$'s private key so that both $\mathcal{S}$'s ciphertext and $\mathcal{R}$'s ciphertext lie in the same ciphertext space, where the addition and multiplication operations can be performed directly. After that, $\mathcal{R}$ sends the re-encrypted ciphertexts to $\mathcal{S}$. It is $\mathcal{S}$'s turn to eliminate the random integer r introduced by itself and carry out the polynomial calculation. At last, $\mathcal{S}$ sends the calculation result back to $\mathcal{R}$, which can only be decrypted by $\mathcal{R}$'s private key. Once $\mathcal{R}$ finishes the decryption, it shares the output with $\mathcal{S}$. Throughout the process, the privacy of the inputs and intermediate results are kept secret, which fulfills the goal of 2PC.

4. Concrete Construction for Two-Party LPCP

In this section, we give a detailed description of our two-party secure computation protocol LPCP. For three-party and more parties MPC cases, see Section 6.

In the setting of two-party LPCP, there are two semi-honest parties P_1 and P_2, with inputs m_1 and m_2, respectively. Both parties intend to compute a polynomial function $F(m_1, m_2) = \sum_{i=1}^{n} a_i m_1^{b_i} m_2^{c_i}$ without leaking any information of their input. Here, we assume the communication channel and both parties' identifications are authenticated, so they do not need additional negotiations. For convenience, we use $\mathcal{S}$ to denote sender P_1 and $\mathcal{R}$ to denote receiver P_2. As Section 3 shows, our protocol consists of six phases—the Setup phase, KeyGen phase, Enc phase, Re-Enc phase, Eval phase, and Dec phase in four communication rounds. Figure 1 shows the sketch map of our protocol.

Setup: At the beginning of the protocol, P_1 and P_2 decide the security parameter 1^λ. If party P_i has not generated the one-way trapdoor permutation function, P_i runs the one-way trapdoor permutation generation algorithm to obtain a one-way trapdoor permutation function pair (f, f^{-1}) on $\{0,1\}^{2\lambda}$. We denote the one-way trapdoor permutation function key pair of sender $\mathcal{S}$ as (sk_{Sen}, pk_{Sen}) and the key pair of receiver $\mathcal{R}$ as (sk_{Rec}, pk_{Rec}). If both parties have generated their one-way trapdoor permutation function, they can skip the above generation process.

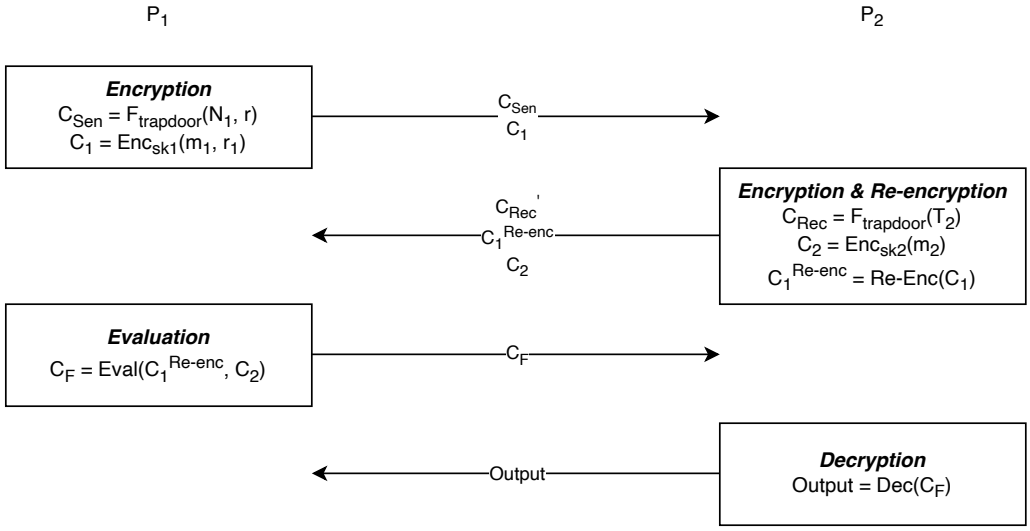

Figure 1. The interaction flow chart of LPCP's framework. The process of P_1 is on the left side and the process of P_2 is on the right side.

Once the security parameter length λ is decided, all the parameters can be generated without this communication. For this reason, we can pre-process the setup phase as follows. Both parties pre-process the generation of parameters while they are free. In this way, we can reduce one communication round to optimize the setup phase. As a result, when the request arrives, both parties choose the corresponding security parameters from the parameter library and run the protocol as soon as possible.

KeyGen: In the key generation phase, both parties initialize an integer $N_0 \in \{0,1\}^{2\lambda}$. The message space $\mathcal{M}$ is set to be $\mathbb{Z}^*_{N'_0}$ where $N'_0 = \{0,1\}^{\lambda-1}$.

For each party P_i where $i = 0$ or $i = 1$, three primes p_i, q_i, s_i are randomly and independently chosen such that $|p_i| = |q_i| = |s_i| = \lambda$. P_i computes $N_i = p_i q_i$ satisfying $N_i \geq N_0$ and $T_i = p_i q_i s_i$. p_i^{-1} is the multiplicative inverse element of p_i modulo q_i such that $p_i^{-1} p_i \equiv 1 \bmod q_i$ and q_i^{-1} is the multiplicative inverse element of q_i modulo p_i such that $q_i^{-1} q_i \equiv 1 \bmod p_i$. Each party holds $(p_i, q_i, s_i, N_i, T_i)$, where p_i, q_i, N_i, s_i remain secret, while T_i is public. As LPCP only involves symmetric encryption, there is no public key. This generation of big primes p_i, q_i, s_i and the multiplicative inverse elements p_i^{-1}, q_i^{-1} can be pre-processed before the protocol when the parties are free. When the secure computation request comes, the parties can quickly choose a tuple of pre-generated parameters and consume it in the following execution. It must be ensured that every secure computation consumes a differently new tuple of (p_i, q_i, s_i).

Enc: In the encryption phase, sender $\mathcal{S}$ randomly chooses $r_1 \in \{0,1\}^{\lambda}$ and $r'_1, r \in \{0,1\}^{2\lambda}$, then computes

$$\begin{aligned} C_{Sen} &= f_{pk_{Rec}}(N_1 || T_1 || r) \\ m_{1,p_1} &= m_1 \bmod p_1 \\ m_{1,q_1} &= m_1 \bmod q_1 \\ C_{1,CRT} &= p_1^{-1} p_1 m_{1,q_1}^{q_1} + q_1^{-1} q_1 m_{1,p_1}^{p_1} \\ C_1 &= < r_1, r'_1 > \cdot < C_{1,CRT}, N_1 > \bmod T_1 \\ &= (r_1 (p_1^{-1} p_1 m_{1,q_1}^{q_1} + q_1^{-1} q_1 m_{1,p_1}^{p_1}) + r'_1 N_1) \bmod T_1 \\ hash_1 &= \mathcal{H}(C_{Sen} || C_1) \end{aligned} \tag{4}$$

$\mathcal{S}$ encrypts his secret modulus N_1 and a random element r with the public key of $\mathcal{R}$'s trapdoor permutation. Then $\mathcal{S}$ implements the Chinese remainder theorem and the Euler's theorem to construct $C_{1,CRT}$. Here, the purpose of introducing Euler's theorem is to mess

up the plaintext information. The ciphertext C_1 equals the inner product of randomness tuple $< r_1, r_1' >$ and secret tuple $< C_{1,CRT}, N_1 >$ modulo T_1. In the first communication round, $\mathcal{S}$ sends $C = (C_1, C_{Sen})$ together with the hash value $hash_1$ to $\mathcal{R}$.

On the other hand, receiver $\mathcal{R}$ randomly chooses $r_2 \in \{0,1\}^\lambda$ and computes

$$\begin{aligned} C_{Rec} &= f_{pk_{Sen}}(T_2) \\ m_{2,p_2} &= m_2 \bmod p_2 \\ m_{2,q_2} &= m_2 \bmod q_2 \\ C_{2,CRT} &= p_2^{-1} p_2 m_{2,q_2}^{q_2} + q_2^{-1} q_2 m_{2,p_2}^{p_2} \\ C_2 &= < 1, r_2 > \cdot < C_{2,CRT}, N_2 > \bmod T_2 \\ &= (p_2^{-1} p_2 m_{2,q_2}^{q_2} + q_2^{-1} q_2 m_{2,p_2}^{p_2} + r_2 N_2) \bmod T_2 \end{aligned} \tag{5}$$

The encryption process of receiver $\mathcal{R}$ slightly differs from $\mathcal{S}$, as the first component of ciphertext $C_{2,CRT}$ is multiplied by the constant 1 instead of a random element. For $\mathcal{S}$, the existence of the random element r_1 is to mask the message for the subsequent process and expand the ciphertext space into $\mathbb{Z}_{T_2}$. As there is no need for C_2 to be masked and re-encrypted, $\mathcal{R}$ multiplies $C_{2,CRT}$ by the constant 1. To reduce the communication round, receiver $\mathcal{R}$ does not send out the ciphertext immediately. Instead, receiver $\mathcal{R}$ delays sending the ciphertext to the re-encryption phase.

Re-Enc: After receiving the ciphertext, $\mathcal{R}$ checks whether $\mathcal{H}(C_1||C_{Sen})$ equals the hash value $hash_1$. If verified, $\mathcal{R}$ continues the re-encryption process. Otherwise, $\mathcal{R}$ outputs $\perp$ to abort the execution. Then $\mathcal{R}$ decrypts C_{Sen} to obtain the modulus N_1 and randomly chooses $r_2' \in_R \{0,1\}^{2\lambda}$. $\mathcal{R}$ re-encrypts $\mathcal{S}$'s ciphertext C_1 as follows.

$$\begin{aligned} N_1||r &= f_{sk_{Rec}}(C_{Sen}) \\ C_1' &= C_1 \bmod N_1 \\ C_{1,q_2} &= C_1' \bmod q_2 \\ C_{1,p_2} &= C_1' \bmod p_2 \\ C_1^{re-enc} &= (p_2^{-1} p_2 C_{1,q_2}^{q_2} + q_2^{-1} q_2 C_{1,p_2}^{p_2} + r_2' N_2) \bmod T_2 \\ hash_2 &= \mathcal{H}(C_{Rec}||C_2||C_1^{re-enc}) \end{aligned} \tag{6}$$

As a result, the ciphertext C_1' is equivalent to $r_1 m_1 \bmod N_1$ according to the Chinese remainder theorem and Euler's theorem. The re-encryption operation transforms $\mathcal{S}$'s ciphertext space into $\mathcal{R}$'s ciphertext space T_2, which lays a foundation for the next evaluation phase. In the second communication round, receiver $\mathcal{R}$ sends ciphertext (C_1^{re-enc}, C_2) with the hash signature $hash_2$ back to $\mathcal{S}$. Here, the security problem comes if $\mathcal{R}$ knows the ciphertext C_1 and the modulus N_1, does it reveal any information about the input of sender $\mathcal{S}$.

Theorem 1. *Assuming the secret parameter r' is randomly chosen from $\{0,1\}^{2\lambda}$ and the modulus T are known to the adversary. The encryption scheme of receiver $\mathcal{R}$*

$$C = ((p^{-1} p m_q^q + q^{-1} q m_p^p) + r' N) \bmod T$$

is semantically secure in the presence of an eavesdropper.

Proof. First, our protocol is under the private key encryption scheme. We define the experiment $PrivK_{\mathcal{A},\Pi}^{eva}$ as follows.

1. The adversary $\mathcal{A}$ outputs a pair of messages $|m_0| = |m_1|$ where $m_0, m_1 \in \{0,1\}^\lambda$.
2. Two big primes p and q are generated using *KeyGen*, and a uniform bit $b \in \{0,1\}$ is chosen. Ciphertext $C \leftarrow Enc_{sk(p,q)}(m_b)$ is computed and given to $\mathcal{A}$.
3. $\mathcal{A}$ outputs a bit b'.

4. The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise. We write $PrivK^{eva}_{\mathcal{A},\Pi} = 1$ if the output of the experiment is 1 and in this case we say that $\mathcal{A}$ succeeds.

We construct an adversary $\mathcal{A}'$ so that it emulates the eavesdropping experiment for $\mathcal{A}$. We define the experiment $Factor_{\mathcal{A}'}(T)$ as follows.

1. Take as input the product of three primes $T \in \{0,1\}^{3\lambda}$.
2. Generate the ciphertext C as in Enc.
3. Give C to $\mathcal{A}$ and obtain output N. Output 1 if $T = Ns$, and output 0 otherwise.

If $\mathcal{A}$ can break the ciphertext C to obtain the plaintext m_b, then $\mathcal{A}$ can calculate $C - m_b \bmod T$. As $(C - m_b) \equiv 0 \bmod N$, $\mathcal{A}$ can obtain the modulus N by calculating the greatest common divisor $gcd(C - m_b, T)$. $\mathcal{A}$ returns the modulus N to $\mathcal{A}'$. $\mathcal{A}'$ can factor the product of big primes $T = Ns$, which breaks the factor assumption. That is, if $\mathcal{A}$ can break the ciphertext with a non-negligible advantage, $\mathcal{A}'$ can break the factor assumption. Therefore, the encryption scheme is semantically secure in the presence of an eavesdropper. □

Theorem 2. *Assume that the secret parameter r is randomly chosen from $\{0,1\}^{\lambda}$ and the moduli N and T are known to the adversary $\mathcal{A}$. The encryption scheme of the sender*

$$C = (r(p^{-1}pm_q^q + q^{-1}qm_p^p) + r'N) \bmod T$$

is semantically secure in the presence of an eavesdropper.

Proof. First, our protocol is under the private key encryption scheme. We define the experiment $PrivK^{eva}_{\mathcal{A},\Pi}$ as follows.

1. The adversary $\mathcal{A}$ outputs a pair of messages $|m_0| = |m_1|$ where $m_0, m_1 \in \{0,1\}^{\lambda}$.
2. Two big primes p and q are generated using $KeyGen$, and a uniform bit $b \in \{0,1\}$ is chosen. Ciphertext $C \leftarrow Enc_{sk(p,q)}(m_b)$ is computed and given to $\mathcal{A}$.
3. $\mathcal{A}$ outputs a bit b'.
4. The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise. We write $PrivK^{eva}_{\mathcal{A},\Pi} = 1$ if the output of the experiment is 1 and in this case, we say that $\mathcal{A}$ succeeds.

We construct an adversary $\mathcal{A}'$ so that it emulates the eavesdropping experiment for $\mathcal{A}$. We define the experiment $Factor_{\mathcal{A}'}(N)$ as follows.

1. Take as input the product of two primes $N \in \{0,1\}^{2\lambda}$.
2. Generate the ciphertext C as in Enc.
3. Give C to $\mathcal{A}$ and obtain output p, q. Output 1 if $N = pq$, and output 0 otherwise.

If $\mathcal{A}$ can break the ciphertext C to obtain the plaintext m_b, then $\mathcal{A}$ can obtain the randomness r by multiplying m^{-1}, the inverse element of m, with the result of C mod N. According to the Euler theorem, $m_q^{q-1} \equiv 1 \bmod q$ and $m_p^{p-1} \equiv 1 \bmod p$. We can obtain $m_q^q \equiv m_q \bmod q$ and $m_p^p \equiv m_p \bmod p$. According to the Chinese theorem, the adversary $\mathcal{A}$ can obtain m_p and m_q. Thus, $\mathcal{A}$ obtains the prime p by calculating the greatest common divisor $gcd(N, m - m_p)$ and obtains the prime q by calculating the greatest common divisor $gcd(N, m - m_q)$.

Now the adversary $\mathcal{A}'$ takes N as input. $\mathcal{A}'$ constructs the ciphertext C and gives (C, N) to $\mathcal{A}$ as input. As $\mathcal{A}$ can break the ciphertext and return the big primes p, q to $\mathcal{A}'$, $\mathcal{A}'$ can return $N = pq$ and achieve the prime decomposition, which breaks the factor assumption. $\mathcal{A}'$ can factor the product of big primes $N = pq$, which breaks the factor assumption. That is, if $\mathcal{A}$ can break the ciphertext with a non-negligible advantage, $\mathcal{A}'$ can break the factor assumption. Therefore, the encryption scheme is semantically secure in the presence of an eavesdropper. □

Now the focus of our work is the presentation of the LPCP protocol and a more formal security analysis is required in future work.

Eval: Before introducing the evaluation phase, we prove the additive and multiplicative homomorphism of our CRT-based encryption scheme. The purpose of the re-encryption phase is to unify ciphertexts of P_1 and P_2 into the same ring space Z_{T_2}. After the re-encryption, ciphertexts C_1^{re-enc} and C_2 are encrypted by the same private key of P_2 in the same ring space Z_{T_2}, where the additional and multiplicative operations can be performed directly on the ciphertexts. Below, we prove the homomorphism property that the decryption result of $C_1^{re-enc} + C_2 \equiv r_1 m_1 + m_2 \bmod N_2$ and the decryption result of $C_1^{re-enc} * C_2 \equiv r_1 m_1 * m_2 \bmod N_2$.

Proof.

$$\begin{aligned}
&Enc(r_1 m_1) + Enc(m_2) = C_1^{re-enc} + C_2 \\
&\equiv p_2^{-1} p_2 (C_{1,q_2}^{q_2} + C_{2,q_2}^{q_2}) + q_2^{-1} q_2 (C_{1,p_2}^{p_2} + C_{2,p_2}^{p_2}) + RN_2 \\
&\bmod T_2 \\
&Enc(r_1 m_1) * Enc(m_2) = C_1^{re-enc} * C_2 \\
&\equiv (p_2^{-1} p_2)^2 (C_{1,q_2} C_{2,q_2})^{q_2} + (q_2^{-1} q_2)^2 (C_{1,p_2} C_{2,p_2})^{p_2} + R' N_2 \\
&\bmod T_2
\end{aligned}$$

$$\begin{aligned}
&Dec(C_1^{re-enc} + C_2) \equiv (m_1 + m_2) \bmod N_2 \\
&Dec(C_1^{re-enc} * C_2) \equiv (m_1 * m_2) \bmod N_2
\end{aligned}$$

□

After receiving the message $(C_1^{re-enc}, C_2, C_{Rec}, hash_2)$, sender $\mathcal{S}$ checks whether $\mathcal{H}(C_1^{re-enc} || C_2 || C_{Rec})$ equals the hash value $hash_2$. If verified, $\mathcal{S}$ continues the evaluation process. Otherwise, $\mathcal{S}$ outputs $\perp$ to abort the execution. Then $\mathcal{S}$ decrypts C_{Rec} to obtain T_2 and partially decrypts C_1^{re-enc} while keeping C_2 unchanged. The multiplicative inverse of r_1 satisfies $r_1 r_1^{-1} = 1 \bmod T_2$. As parameter N_2 is known to $\mathcal{R}$, it is easy for $\mathcal{R}$ to obtain the inverse element r_1^{-1} in the decryption phase. Degree deg_F is the maximum degree of polynomial $F(x_1, x_2)$, and deg_i is the degree of the i^{th} term of the polynomial. There are two reasons why the random element r is introduced into the ciphertext. One reason is to prevent $\mathcal{R}$ from deducing the plaintext m_1, reversely referring to C_F. The other reason is to unify the degree of the polynomial terms to perform the additive operations. In the third communication round, $\mathcal{S}$ sends the final ciphertext C_F and hash signature $hash_3$ to $\mathcal{R}$.

$$\begin{aligned}
T_2 &= f_{sk_{Sen}}(C_{Rec}), \\
C_{1,eval} &= r r_1^{-1} C_1^{re-enc} \bmod T_2 \\
C_{2,eval} &= r C_2 \bmod T_2 \\
C_{term_i} &= a_i r^{(deg_F - deg_i)} C_{1,eval}^{b_i} C_{2,eval}^{c_i} \bmod T_2 \\
deg_F &= \max_i (deg_i) \\
deg_i &= b_i + c_i \\
C_F &= \sum_{i=1}^{n} C_{term_i} \\
hash_3 &= \mathcal{H}(C_F)
\end{aligned} \tag{7}$$

Dec: In the decryption phase, after receiving the message $(C_F, hash_3)$, receiver $\mathcal{R}$ checks whether $\mathcal{H}(C_F)$ equals the hash value $hash_3$. If verified, $\mathcal{R}$ continues the decryption

process. Otherwise, $\mathcal{R}$ outputs $\perp$ to abort the execution. $\mathcal{R}$ obtains the random element r from the trapdoor permutation and computes the multiplicative inverse of r that satisfies $rr^{-1} = 1 \bmod N_2$. In the last communication round, $\mathcal{R}$ shares the result $Output$ with $\mathcal{S}$.

$$\begin{aligned} r &= f_{sk_{Rec}}(C_{Sen}) \\ Output &= r^{-deg_F} C_F \bmod N_2 \end{aligned} \tag{8}$$

At the end, $\mathcal{S}$ and $\mathcal{R}$ obtain the result of polynomial function $F(m_1, m_2)$ without revealing their inputs m_1 and m_2, which achieves the goal of secure two-party computation.

5. Security Proof

In this section, we give the security proof of LPCP.

5.1. Correctness

Before the security proof, we present the correctness of LPCP.

$$\begin{aligned} & r^{-deg_F} C_F \bmod N_2 \\ &= r^{-deg_F} \sum_{i=1}^{n} k_i r^{(deg_F - deg_i)} C_{1,eval}^{deg_i^1} C_{2,eval}^{deg_i^2} \bmod N_2 \\ &= \sum_{i=1}^{n} k_i (r_1^{-1} C_1'')^{deg_i^1} C_2{}^{deg_i^2} \bmod N_2 \\ &= \sum_{i=1}^{n} k_i (r_1^{-1} C_1'' \bmod N)^{deg_i^1} (C_2 \bmod N)^{deg_i^2} \bmod N_2 \\ &= \sum_{i=1}^{n} k_i (r_1^{-1} r_1 m_1)^{deg_i^1} (m_2)^{deg_i^2} \bmod N_2 \\ &= \sum_{i=1}^{n} k_i m_1^{deg_i^1} m_2^{deg_i^2} \bmod N_2 \end{aligned} \tag{9}$$

5.2. Privacy

We give the privacy proof of the 2PC LPCP protocol by simulation.

First, we consider the case of sender $\mathcal{S}$ being corrupted by the semi-honest adversary $\mathcal{A}_1$. The adversary $\mathcal{A}_1$ obtains access to all the input and output of $\mathcal{S}$, including $\mathcal{S}$'s security parameter 1^λ, the input message m_1, the prime tuple (p_1, q_1, s_1), the randomness tuple, the polynomial function $F(m_1, m_2)$ and the ciphertexts received. Receiver $\mathcal{R}$ sends the ciphertexts to the adversary $\mathcal{A}_1$ in the second communication round and fourth communication round. The input ciphertexts are simulated as follows:

1. In the second round simulation, $\mathcal{S}$ receives the ciphertext (C_1^{re-enc}, C_2).
2. In the fourth round simulation, $\mathcal{S}$ receives the plaintext $Output$.

Below, we prove that the honest sender $\mathcal{S}$ cannot distinguish its simulated view with its real execution view.

1. $\mathcal{H}'_{real}$: This hybrid is the same as the real-world execution.
2. $\mathcal{H}'_1$: This hybrid is the same as $\mathcal{H}'_{real}$, except we change the output from $Output$ to $Output^{Sim}$. $\mathcal{R}$ randomly chooses $Output^{Sim} \in \{0,1\}^{2\lambda}$. According to Theorem 2 and the reason that $\mathcal{S}$ does not know the modulus N_2 for decryption, $\mathcal{S}$ cannot distinguish the real output $Output$ from the simulated output $Output^{Sim}$.
3. $\mathcal{H}'_2$: This hybrid is the same as $\mathcal{H}'_1$, except we change the ciphertext from (C_1^{re-enc}, C_2) to $(C_1^{re-enc^{Sim}}, C_2^{Sim})$. $\mathcal{R}$ randomly chooses the ciphertext $C_1^{re-enc^{Sim}} \leq T$ and $C_2^{Sim} \leq T$. As $\mathcal{S}$ has no information about $\mathcal{R}$'s value m_2, N_2, r_2, $\mathcal{S}$ cannot distinguish the real ciphertext C_2 from the simulated ciphertext C_2^{Sim}. According to Theorem 1, $\mathcal{S}$ cannot distinguish the real re-encryption ciphertext C_1^{re-enc} from the simulated re-encryption ciphertext $C_1^{re-enc^{Sim}}$.

The hybrid model $\mathcal{H}'_{real}$ is statistically indistinguishable from the hybrid model $\mathcal{H}'_2$, where $\mathcal{H}'_{real}$ is the real-world view of $\mathcal{S}$, and $\mathcal{H}'_2$ is the ideal world view of $\mathcal{S}$. Therefore, the 2PC LPCP protocol is secure when $\mathcal{S}$ is corrupted.

Secondly, we consider the case of $\mathcal{R}$ being corrupted by the semi-honest adversary $\mathcal{A}_2$. The adversary $\mathcal{A}_2$ gets access to all the input and output of receiver $\mathcal{R}$, including $\mathcal{R}$'s security parameter 1^λ, the input message m_2, the prime tuple (p_2, q_2, s_2), the randomness tuple, the polynomial function $F(m_1, m_2)$ and the ciphertexts received. The adversary $\mathcal{A}_2$ receives ciphertexts in the first and third communication round.

1. In the first round, $\mathcal{R}$ receives the ciphertext C_1.
2. In the third round, $\mathcal{R}$ receives the ciphertext C_F.

Below, we prove that the semi-honest sender $\mathcal{R}$ can not distinguish the simulated view with the real execution view.

1. $\mathcal{H}_{Real}$: This hybrid model is the same as the real-world execution.
2. $\mathcal{H}_1$: This hybrid model is the same as $\mathcal{H}_{real}$, except we change the ciphertext (C_1^{re-enc}, C_2) to the randomly simulated ciphertexts $(C_1^{re-enc,Sim}, C_2^{Sim})$. When $\mathcal{S}$ simulates the ciphertext (C_1^{re-enc}, C_2), $\mathcal{S}$ chooses a random element $C_F^{Sim} \in \{0,1\}^{3\lambda}$ and sends C_F^{Sim} to $\mathcal{R}$. As $\mathcal{R}$ does not know the randomness r_1, the ciphertext C_1^{re-enc} is multiplied with r_1^{-1}, the inverse element of r_1. Therefore, $\mathcal{R}$ cannot distinguish the randomly forged ciphertext C_F^{Sim} from the real ciphertext C_F.

The hybrid model $\mathcal{H}_{real}$ is statistically indistinguishable from the hybrid model $\mathcal{H}_1$, where $\mathcal{H}_{real}$ is the real-world view of $\mathcal{R}$ and $\mathcal{H}_1$ is the ideal-world view of $\mathcal{R}$. Therefore, the 2PC LPCP protocol is secure when $\mathcal{R}$ is corrupted.

In summary, our 2PC LPCP is secure against the semi-honest adversary and guarantees the privacy of both parties' input information. Now the focus of our work is the presentation of the LPCP protocol. A more formal security analysis is required in future work.

6. Extension to Three and More Parties LPCP

In Section 4, we present a two-party secure computation construction. Furthermore, LPCP can be extended into an MPC scheme with three and more parties utilizing the two-party LPCP scheme. In the following part, we present a three-party MPC construction. There are three parties P_1, P_2 and P_3 with inputs m_1, m_2 and m_3, respectively, who aim to compute a polynomial function $F(m_1, m_2, m_3) = \sum_i k_i m_1^{a_i} m_2^{b_i} m_3^{c_i}$.

Our main idea is that P_1 and P_2 perform the LPCP protocol and then perform the other LPCP protocol with P_3. Concretely speaking, we execute the two-party LPCP scheme between P_1 and P_2. Then we see P_1 and P_2 as a whole part and execute the other two-party LPCP scheme between this whole part and P_3. To ensure the privacy security, the original two-party LPCP scheme is modified slightly. First, P_1 executes the operations as the sender, and P_2 executes the same operations as the receiver in 4, except that P_1 does not send the last ciphertext C_F to P_2 in the re-encryption phase. Instead, P_1 encrypts the last ciphertext with its secret keys p_2, q_2 and begins a new round of the two-party LPCP scheme with P_3. The true decryption phase is delayed to the end of the whole protocol, where P_2 and P_1 decrypt the final ciphertext to obtain the answer.

Setup: The setup phase is the same as the two-party LPCP scheme.

KeyGeneration: For $i = 0, 1, 2$, each party P_i randomly and independently chooses three length-equal primes p_i, q_i, s_i which satisfy $|p_i| = |q_i| = |s_i| = \lambda$. Each party P_i generates a trapdoor permutation key pair sk_{P_i} and pk_{P_i}. Then, P_i computes $N_i = p_i q_i$ and $T_i = p_i q_i s_i$.

Encryption: P_1 randomly chooses $r_1 \in \{0,1\}^\lambda$ and $r'_1, r \in \{0,1\}^{3\lambda}$. Then P_1 computes as follows and sends C_{P_1} and C_1 to P_2.

$$\begin{aligned} C_{P_1} &= f_{pk_{P_2}}(N_1) \\ m_{1,p_1} &= m_1 \bmod p_1 \\ m_{1,q_1} &= m_1 \bmod q_1 \\ C_{1,CRT} &= p_1^{-1} p_1 m_{1,q_1}^{q_1} + q_1^{-1} q_1 m_{1,p_1}^{p_1} \\ C_1 &= < r_1, r_1' > \cdot < C_{1,CRT}, N_1 > \bmod T_1 \\ &= (r_1(p_1^{-1} p_1 m_{1,q_1}^{q_1} + q_1^{-1} q_1 m_{1,p_1}^{p_1}) + r_1' N_1) \bmod T_1 \end{aligned} \tag{10}$$

P_2 randomly chooses $r_2 \in \{0,1\}^{3\lambda}$ and computes

$$\begin{aligned} C_{P_2} &= f_{pk_{P_1}}(T_2) \\ m_{2,p_2} &= m_2 \bmod p_2 \\ m_{2,q_2} &= m_2 \bmod q_2 \\ C_{2,CRT} &= p_2^{-1} p_2 m_{2,q_2}^{q_2} + q_2^{-1} q_2 m_{2,p_2}^{p_2} \\ C_2 &= < 1, r_2 > \cdot < C_{2,CRT}, N_2 > \bmod T_2 \\ &= (p_2^{-1} p_2 m_{2,q_2}^{q_2} + q_2^{-1} q_2 m_{2,p_2}^{p_2} + r_2 N_2) \bmod T_2 \end{aligned} \tag{11}$$

P_3 randomly chooses $r_3 \in \{0,1\}^{\lambda}$ and $r_3' \in \{0,1\}^{3\lambda}$. Then, P_3 computes as follows and sends C_{P_3} and C_3 to P_2.

$$\begin{aligned} C_{P_3} &= f_{pk_{P_2}}(N_3) \\ m_{3,p_3} &= m_3 \bmod p_3 \\ m_{3,q_3} &= m_3 \bmod q_3 \\ C_{3,CRT} &= p_3^{-1} p_3 m_{3,q_3}^{q_3} + q_3^{-1} q_3 m_{3,p_3}^{p_3} \\ C_3 &= < r_3, r_3' > \cdot < C_{3,CRT}, N_3 > \bmod T_3 \\ &= (r_3(p_3^{-1} p_3 m_{3,q_3}^{q_3} + q_3^{-1} q_3 m_{3,p_3}^{p_3}) + r_3' N_3) \bmod T_3 \end{aligned} \tag{12}$$

Re-Encryption: In the re-encryption phase, P_2 receives the ciphertexts C_{P_1} and C_1 from P_1 and C_{P_3} and C_3 from P_3. P_2 opens the trapdoor permutation with its secret key sk_{P_2}, randomly chooses r_2' and $r_2'' \in \{0,1\}^{3\lambda}$, then re-encrypts the ciphertexts C_1 and C_3 as follows.

$$\begin{aligned} N_1 &= f_{sk_{P_2}}(C_{P_1}) \\ N_3 &= f_{sk_{P_2}}(C_{P_3}) \\ C_1' &= C_1 \bmod N_1 \\ C_{1,p_2} &= C_1' \bmod p_2 \\ C_{1,q_2} &= C_1' \bmod q_2 \\ C_1^{re-enc} &= (p_2^{-1} p_2 C_{1,q_2}^{q_2} + q_2^{-1} q_2 C_{1,p_2}^{p_2} + r_2' N_2) \bmod T_2 \\ C_3' &= C_3 \bmod N_3 \\ C_{3,p_2} &= C_3' \bmod p_2 \\ C_{3,q_2} &= C_3' \bmod q_2 \\ C_3^{re-enc} &= (p_2^{-1} p_2 C_{3,q_2}^{q_2} + q_2^{-1} q_2 C_{3,p_2}^{p_2} + r_2'' N_2) \bmod T_2 \end{aligned} \tag{13}$$

After the re-encryption, P_2 sends the re-encrypted ciphertexts C_1^{re-enc} and C_3^{re-enc} back to P_1. P_1 randomly chooses $r_4 \in \{0,1\}^{3\lambda}$ and re-encrypts C_3^{re-enc} as follows.

$$\begin{aligned} C_{3,p_2}^{re-enc} &= C_3^{re-enc} \bmod p_1 \\ C_{3,q_2}^{re-enc} &= C_3^{re-enc} \bmod q_1 \\ C_3^{re-enc'} &= (p_1^{-1} p_1 C_{3,q_2}^{re-enc q_1} + q_1^{-1} q_1 C_{3,p_2}^{re-enc p_2} + r_4 N_1) \bmod T_1 \end{aligned} \tag{14}$$

Evaluation Round1: In the first round of the evaluation phase, P_1 encrypts the resulting ciphertext C_{round1} instead of sending it back to P_2. P_1 randomly chooses $r_5 \in \{0,1\}^{\lambda}$ and $r_5' \in \{0,1\}^{3\lambda}$.

$$\begin{aligned} T_2 &= f_{sk_{P_1}}(C_{P_2}), \\ C_{1,eval} &= r_1^{-1} C_1^{re-enc} \bmod T_2 \\ C_{2,eval} &= C_2 \bmod T_2 \\ C_{term_i} &= k_i C_{1,eval}^{a_i} C_{2,eval}^{b_i} \bmod T_2 \\ C_{round1} &= \sum_{i=1}^{n} C_{term_i} \end{aligned} \tag{15}$$

Then P_2 computes the ciphertext C'_{round1} and sends it to the third participant party P_3.

$$\begin{aligned} C_{round1,p_1} &= C_{round1} \bmod p_1 \\ C_{round1,q_1} &= C_{round1} \bmod q_1 \\ C_{round1,CRT} &= p_1^{-1} p_1 C_{round1,q_1}^{q_1} + q_1^{-1} q_1 C_{round1,p_1}^{p_1} \\ C'_{round1} &= < r_5, r_5' > \cdot < C_{round1,CRT}, N_1 > \bmod T_1 \\ &= (r_5 (p_1^{-1} p_1 C_{round1,q_1}^{q_1} + q_1^{-1} q_1 C_{round1,p_1}^{p_1}) + r_5' N_1) \bmod T_1 \end{aligned} \tag{16}$$

Evaluation Round2 P_3 receives the ciphertext C'_{round1} and $C_3^{re-enc'}$, then computes as follows.

$$\begin{aligned} C'_{round1,eval} &= r_3^{-1} C'_{round1} \bmod T_1 \\ C_{3,eval}^{re-enc'} &= C_3^{re-enc'} \bmod T_1 \\ C'_{term_i} &= C'_{round1,eval} C_{3,eval}^{re-enc' c_i} \bmod T_1 \\ C_{round2} &= \sum_{i=1}^{n} C'_{term_i} \end{aligned} \tag{17}$$

After the evaluation, the ciphertext C_{round2} is sent to P_1 for partial decryption.

Decryption In the decryption phase, P_1 performs the first decryption, then P_2 performs the second decryption.

$$\begin{aligned} C_F &= C_{round2} \bmod N_1 \\ Output &= C_F \bmod N_2 \\ Output &= F(m_1, m_2, m_3) \end{aligned} \tag{18}$$

The existence of two rounds of evaluation is to make sure the ciphertexts are encrypted by the same private space and lie in the same ring space. When there are more than three parties, we can make use of the paradigm above to modify our LPCP scheme. However, the more parties there are, the more communication rounds there will be. When the party number grows large, the communication overhead increases rapidly, which leads to high latency. We will research this problem in our later work.

7. Mobile Device Distance Measurement Applications

In this section, we present a mobile device distance measurement application based on LPCP.

Nowadays, many mobile phone applications (APP) services are built on the location based service (LBS) [24,25]. For example, when users want to search the nearest take-out restaurant for dinner or they want to make friends with the stranger within a

three-kilometer range on dating software, they need to upload their location information to APP servers. However, this uploading location information may reveal their location or even reveal their home address. Therefore, it is necessary to introduce location privacy techniques into those LBS-based APPs. Multiparty secure computation techniques can solve the location privacy problems above. However, it takes too much time, from seconds to minutes, for traditional MPC protocols to be executed on mobile devices with low computing power. Our secure computation protocol LPCP becomes of use in such circumstances. LPCP's advantages of fast execution and low communication overhead make it possible for mobile devices to execute MPC protocols.

Supposing there are two users $User_1, User_2$ with mobile devices who wish to measure the distance between each other in an APP locally. The main idea is as follows. As the local APP service only makes sense in a limited and effective range, we divide the geographical positions into many $n \times m$ two-dimensional planes, where $n, m \leq \frac{2^\lambda}{\delta}$. Here, the notation λ is the security parameter, and δ is the amplification factor. Then, the location of each user is mapped into the coordinate system (x_i, y_i) in a co-located rectangle area. Concretely, the location of $User_1$ is scaled up by δ and mapped into the integer coordinate system (x_1, y_1). The location of $User_2$ is also scaled up by δ and mapped into the integer coordinate system (x_2, y_2) so that our LPCP can perform arithmetic operations. The problem of the privacy-preserving distance measurement is now transformed into calculating the polynomial $(x_1 - x_2)^2 + (y_1 - y_2)^2$, which can be efficiently solved by the LPCP protocol.

In the setup phase, both mobile devices obtain their geographical information from the location service and map the geographical information into the integer coordinate system (x_1, y_1) and (x_2, y_2). The security parameter 1^λ of LPCP is assumed to be fixed to a static value. To avoid the known-plaintext attack, each mobile device adds a relatively small noise (e_i, e_i'), where $e_i, e_i' \leq 2^{\lambda/\delta}$ to its location (x_i, y_i) to conceal its real position. After the pre-processing, the coordinate of $User_1$ becomes $(x_1 + e_1, y_1 + e_1')$, and the coordinate of $User_2$ becomes $(x_2 + e_2, y_2 + e_2')$ where the introduced small noise e_1, e_1', e_2, e_2' does not cause a distance result error.

In the key generation phase, both mobile devices implement the KeyGen algorithm to generate the prime tuple and the randomness as Section 4.

In the encryption phase, $User_1$ plays the role as sender and $User_2$ plays the role as receiver. $User_1$ executes the sender's encryption scheme to encrypt messages $x_1 + e_1$ and $y_1 + e_1'$, respectively. $User_2$ executes the receiver's encryption scheme to encrypt messages $x_2 + e_2$ and $y_2 + e_2'$, respectively. $User_1$ and $User_2$ deal with the x-coordinate and y-coordinate calculations separately.

The remaining re-encryption phase, evaluation phase, and decryption phase are the same as Section 4. In the re-encryption phase, $User_2$ executes the Re-Enc algorithm. In the evaluation phase, $User_1$ executes the Eval algorithm. In the decryption phase, $User_2$ executes the Dec algorithm.

In the end, both mobile devices obtain the approximate distance between each other while keeping their exact position unexposed.

8. Performance and Evaluation

In this section, we evaluate the theoretical computation and communication complexity of LPCP and evaluate the performance of LPCP.

8.1. Theoretical Analysis

For the computation aspects, the computation overhead comprises three parts—the one-way trapdoor permutation, arithmetic ring operations and hash operations. Both the one-way trapdoor permutation encryption and the one-way trapdoor permutation decryption are implemented twice. The main overhead of the arithmetic ring operations lies in the re-encryption phase. The hash operations is implemented six times. Therefore, the computational complexity of LPCP is $\mathcal{O}(1) \cdot T_{trapdoor} + \mathcal{O}(max(log_2^{p+q}, term_F)) \cdot T_{multiplication} + \mathcal{O}(1) \cdot T_{hash}$, where (p, q) is the big prime parameters, $term_F$ is the polynomial term number,

$T_{trapdoor}$ is the time of executing a trapdoor permutation encryption, $T_{multiplication}$ is the time of executing the multiplication arithmetic operations, and T_{hash} is the time of executing a hash function.

For the communication aspects, the communication packages contain two trapdoor permutation ciphertexts, five 3λ-long ciphertexts and three hash signature messages. To sum up, the communication complexity of LPCP is reduced to $\mathcal{O}(1) \cdot L_{trapdoor} + \mathcal{O}(1) \cdot L_{ciphertext} + \mathcal{O}(1) \cdot L_{hash}$, where $L_{trapdoor}$is the length of the trapdoor permutation ciphertext, $L_{ciphertext}$ is the length of the ciphertext and L_{hash} is the length of the hash signature.

8.2. Practical Performance

Firstly, we run the LPCP code on a Raspberry Pi to show the compatibility of our scheme with the ARM architecture. The concrete configuration of our ARM environment is shown in Table 1.

Table 1. Configurations of experiment on ARM architecture.

CPU	Quad core Cortex-A72 (ARM v8) 64-bit @ 1.5 GHz
RAM	1GB LPDDR4
OS	Ubuntu 20.04
Programming language	c++11
Compiler	gcc version 9.3.0
Libraries	openssl-3.0.1, gmp-6.2.1

Secondly, we run the LPCP code on the Intel X86 architecture computer in order to compare the experimental results with other secure computation protocol implementations which are only supported in the X86 instruction set. The concrete configuration of our X86 environment is shown in Table 2.

Table 2. Configurations of experiment on X86 architecture.

CPU	i5-8259U
RAM	8GB DDR4
OS	Ubuntu 20.04.3 LTS
Programming language	c++11
Compiler	gcc version 9.3.0
Libraries	openssl-3.0.1, gmp-6.2.1

All the experiments are run on the two virtual machines on the same localhost, where the bandwidth bound is 100 Mbps and the latency can be ignored. Given the input m_i, the addition operation expression is $\sum_{i=1}^{n} m_i$ and the multiplication operation expression is $\prod_{i=1}^{n} m_i$.

Figure 2 shows the LPCP running time comparison of different arithmetic operation types both on ARM architecture and X86 architecture. When the security parameter length is 32 bits, 64 bits and 128 bits, the running time of LPCP on the ARM architecture is approximately 10 times slower than the running time of LPCP on the X86 architecture. When the security parameter length grows to 256 bits and 1024 bits, the running time of LPCP on the ARM architecture is approximately four times slower than the running time of LPCP on the X86 architecture. It turns out that LPCP protocol can be effectively implemented on the mobile devices based on ARM architecture.

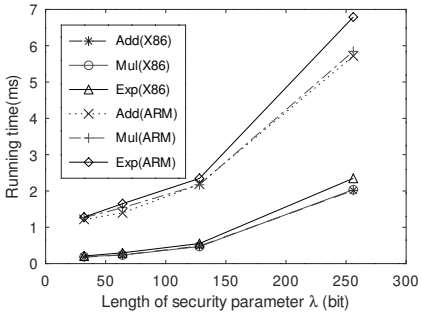

Figure 2. The running time comparison of additive, multiplicative and exponential arithmetic operations of LPCP on ARM and X86 architecture.

Table 3 shows the practical performance of the two-party LPCP protocol on additive and multiplicative arithmetic operations in both the ARM architecture and the X86 architecture environment. In Table 3, the first column is the arithmetic operation type; the second column is the running time of LPCP execution in ARM architecture environment; the third column is the running time of LPCP execution in X86 architecture environment; and the last column is the communication overhead of ciphertexts transferred between two parties.

For comparison with other works, we choose one fully homomorphic encryption open-source library and two secure computation open-source libraries. The homomorphic encryption library is HElib, which realizes a leveled fully homomorphic encryption BGV [26] and CKKS [23]. As for the MPC libraries, one is the EMP toolkit [19], which realizes a 2PC protocol Emp-sh2pc secure against a semi-honest adversary. The other is the MP-SPDZ library [20], which benchmarks various secure multi-party computation (MPC) protocols, such as SPDZ [27], SPDZ2k, MASCOT [28], Overdrive [29], BMR garbled circuits [30], Yao's garbled circuits, and MPC based on Shamir's secret sharing.

For the FHE comparison, we choose the BGV implementation of arithmetic operations in the HElib library and compare the experimental results to our LPCP data. The experiment divides the running time into encryption time, arithmetic operation time and decryption time. Table 4 respectively shows the encryption time, arithmetic operation time, decryption time and total running time of 64-bit and 1024-bit arithmetic operations. In the additive arithmetic operations, the running time of 64-bit and 1024-bit HElib addition is much faster than LPCP. However, the encryption and decryption times of HElib lag behind, which results in the total running time of HElib being slower than LPCP. In the multiplicative arithmetic operations, the running time of both 64-bit and 1024-bit Helib multiplication is slower than LPCP.

Next, we analyze the two-party secure computation experimental results comparison between LPCP and Emp-sh2pc. Figure 3 shows the running time comparison of different arithmetic operations, and Figure 4 shows the communication overhead comparison. We can see that LPCP comes in the lead in both the running time and communication overhead. The running time of our protocol is more than 100 times faster than Emp-sh2pc in 128-bit and 256-bit additive and multiplicative tests. In particular, in the 1024-bit modular exponential operation test, the running time of LPCP is more than 100,000 times faster than Emp-sh2pc. Although the running time of LPCP's additive computation becomes almost as costly as Emp-sh2pc when the bit length of the security parameter comes to 1024 bits. According to the analytic computational and space complexity, the running time of our protocol grows linearly with the largest exponent of the polynomial, which means the multiplicative computation of LPCP takes almost the same amount of time as the additive computation. For this reason, the running time of LPCP's multiplicative computation is exponentially faster than Emp-sh2pc. As the communication latency increases, the additional one communication round of LPCP results in more communication time than the other 2PC protocols. In the communication space comparison, the total communication space is more than 10,000 times smaller than the result of Emp-sh2pc in 128-bit and 256-bit additive and multiplicative tests.

Table 3. Performance of LPCP on additive, multiplicative and exponential operations.

Arithmetic Operation Type	Running Time of ARM Architecture	Running Time of X86 Architecture	Total Communication
32-bit add	1.219 ms	0.1779 ms	89B
32-bit mul	1.268 ms	0.1931 ms	89B
32-bit exp	1.279 ms	0.2098 ms	89B
64-bit add	1.395 ms	0.2387 ms	111B
64-bit mul	1.54 ms	0.2366 ms	111B
64-bit exp	1.651 ms	0.2969 ms	111B
128-bit add	2.189 ms	0.484 ms	155B
128-bit mul	2.171 ms	0.4693 ms	155B
128-bit exp	2.346 ms	0.5529 ms	155B
256-bit add	5.716 ms	2.0154 ms	243B
256-bit mul	5.845 ms	2.0358 ms	243B
256-bit exp	6.79 ms	2.3488 ms	243B
1024-bit add	213.43 ms	85.376 ms	771B
1024-bit mul	221.993 ms	86.446 ms	771B
1024-bit exp	238.629 ms	86.21 ms	771B

Table 4. Performance of BGV implemented in HElib library on additive and multiplicative operations.

Arithmetic Operation Type	Encryption Time	Arithmetic Operation Time	Decryption Time	Total Running Time
64-bit add	0.643 ms	0.000000011 ms	2.010 ms	2.653 ms
64-bit mul	0.643 ms	6.972 ms	2.010 ms	9.625 ms
1024-bit add	935.44 ms	0.000126427 ms	589.312 ms	1524.752 ms
1024-bit mul	935.44 ms	3843.84 ms	589.312 ms	5368.592 ms

When it comes to three-party and four-party secure computation, we compare our LPCP experimental result with MP-SPDZ. To ensure fairness, we choose two kinds of MPC protocols, Semi and Hemi, which are both secure against the semi-honest adversary from the MP-SPDZ library. The difference of the chosen MPC protocol lies in that Semi is built based on oblivious transfer (OT) [31] and Hemi is built based on semi-homomorphic encryption [27]. Figure 5 shows the running time of Semi, Hemi and LPCP in the two-party, three-party and four-party experiments. Figure 6 shows the total communication overhead of Semi, Hemi and LPCP in the two-party, three-party and four-party experiments. The advantage of the fast execution and low communication cost of LPCP becomes much more clear as the number of parties grows because the time complexity and communication complexity grow approximately linearly to the party numbers. From the figures above, we can see that LPCP has better scalability than other traditional MPC protocols in the same condition. It is especially suitable for those mobile devices with low computing power and small storage space to execute LPCP for data privacy.

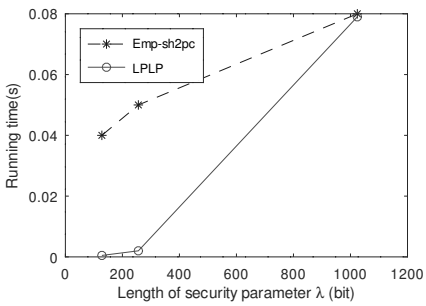

Figure 3. The running time comparison of additive operation between LPCP and Emp-sh2pc.

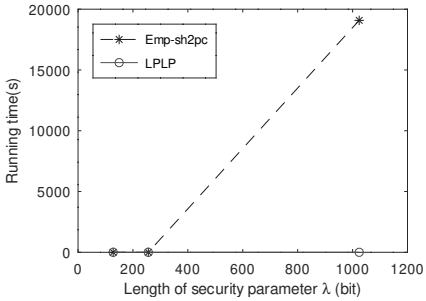

Figure 4. The running time comparison of multiplicative operation between LPCP and Emp-sh2pc.

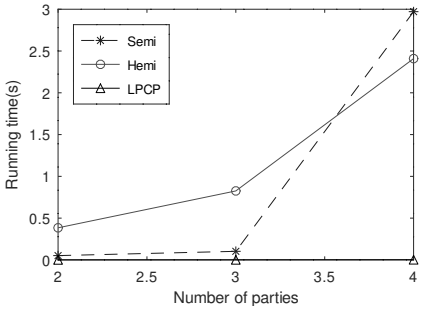

Figure 5. Running time of Semi, Hemi and LPCP in different party numbers conditions.

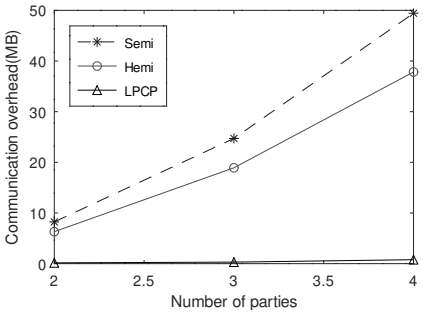

Figure 6. Total communication overhead of Semi, Hemi and LPCP in different party numbers conditions.

9. Conclusions

In this paper, an efficient CRT-based secure computation protocol LPCP for polynomial calculation is proposed. Based on LPCP, we propose a privacy-preserving distance measurement computation protocol that is affordable for mobile devices.

In the end, we implement LPCP in the ARM architecture environment to show its compatibility with mobile devices. We also implement LPCP in the X86 architecture environment and compare the experimental results with fully homomorphic encryption schemes and different kinds of MPC schemes. In both 2PC and MPC comparisons, our LPCP scheme has a great advantage of both running time and communication overhead, which makes it possible to implement secure computation on mobile devices. For future works, we aim to perform a formal security analysis of LPCP protocol and strengthen the security model against the malicious adversary.

Author Contributions: Conceptualization and methodology, J.T., Z.C. and J.S.; writing—original draft, formal analysis, software and visualization, J.T.; writing—review and editing, Z.C., J.S., X.D.; project administration, funding acquisition, Z.C., J.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Key Research and Development Program of China (Grant No. 2020YFA0712300), in part by the National Natural Science Foundation of China (Grant No. 62132005, 61632012, 62172162).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

2PC	Two-party computation
MPC	Multiparty computation
CRT	Chinese remainder theorem
LPCP	Lightning polynomial computation protocol
FHE	Fully homomorphic encryption
KeyGen	Key generation
Enc	Encryption
Re-Enc	Re-encryption
Dec	Decryption
$\mathcal{S}$	Sender
$\mathcal{R}$	Receiver
λ	Security parameter

References

1. Lindell, Y. Secure Multiparty Computation (MPC). *IACR Cryptol. ePrint Arch.* **2020**, 2020, 300.
2. Yao, A.C. Protocols for Secure Computations (Extended Abstract). In Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, Chicago, IL, USA, 3–5 November 1982; IEEE Computer Society: Washington, DC, USA, 1982; pp. 160–164.
3. Mouchet, C.; Troncoso-Pastoriza, J.R.; Hubaux, J. Multiparty Homomorphic Encryption: From Theory to Practice. *IACR Cryptol. ePrint Arch.* **2020**, 2020, 304.
4. Patra, A.; Suresh, A. BLAZE: Blazing Fast Privacy-Preserving Machine Learning. In Proceedings of the 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, CA, USA, 23–26 February 2020; The Internet Society: Reston, VA, USA, 2020.
5. Byali, M.; Chaudhari, H.; Patra, A.; Suresh, A. FLASH: Fast and Robust Framework for Privacy-preserving Machine Learning. *Proc. Priv. Enhancing Technol.* **2020**, *2020*, 459–480. [CrossRef]
6. Rezaeipour, D. Secure Computation for Cloud data Storage. *IACR Cryptol. ePrint Arch.* **2019**, 2019, 709.
7. Demmler, D.; Schneider, T.; Zohner, M. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In Proceedings of the 22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, CA, USA, 8–11 February 2015; The Internet Society: Reston, VA, USA, 2015.
8. Lindell, Y.; Riva, B. Blazing Fast 2PC in the Offline/Online Setting with Security for Malicious Adversaries. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, 12–16 October 2015; Ray, I., Li, N., Kruegel, C., Eds.; ACM: New York, NY, USA, 2015; pp. 579–590.
9. Rindal, P.; Rosulek, M. Faster Malicious 2-Party Secure Computation with Online/Offline Dual Execution. In Proceedings of the 25th USENIX Security Symposium, USENIX Security 16, 10–12 August 2016; Holz, T., Savage, S., Eds.; USENIX Association: Austin, TX, USA, 2016; pp. 297–314.
10. Smart, N.P.; Tanguy, T. TaaS: Commodity MPC via Triples-as-a-Service. In Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, CCSW@CCS 2019, London, UK, 11 November 2019; Sion, R., Papamanthou, C., Eds.; ACM: New York, NY, USA, 2019; pp. 105–116.
11. Ciampi, M.; Ostrovsky, R.; Waldner, H.; Zikas, V. Round-Optimal and Communication-Efficient Multiparty Computation. *IACR Cryptol. ePrint Arch.* **2020**, 2020, 1437.
12. Brakerski, Z.; Vaikuntanathan, V. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In Proceedings of the Advances in Cryptology-CRYPTO 2011—31st Annual Cryptology Conference, Santa Barbara, CA, USA, 14–18 August 2011; Proceedings; Rogaway, P., Ed.; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6841, pp. 505–524.
13. 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 Proceedings of the Advances in Cryptology—EUROCRYPT 2012—31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, 15–19 April 2012; Proceedings; Pointcheval, D., Johansson, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7237, pp. 483–501.
14. López-Alt, A.; Tromer, E.; Vaikuntanathan, V. On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption. *IACR Cryptol. ePrint Arch.* **2013**, *2013*, 94.

15. Mukherjee, P.; Wichs, D. Two Round Multiparty Computation via Multi-key FHE. In Proceedings of the Advances in Cryptology—EUROCRYPT 2016—35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, 8–12 May 2016; Part II; Fischlin, M., Coron, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; Volume 9666, pp. 735–763.
16. Li, Z.; Ma, C.; Zhou, H. Multi-key FHE for multi-bit messages. *Sci. China Inf. Sci.* **2018**, *61*, 029101:1–029101:3. [CrossRef]
17. Zhou, J.; Cao, Z.; Qin, Z.; Dong, X.; Ren, K. LPPA: Lightweight Privacy-Preserving Authentication From Efficient Multi-Key Secure Outsourced Computation for Location-Based Services in VANETs. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 420–434. [CrossRef]
18. Kim, J.; Lee, M.S.; Yun, A.; Cheon, J.H. CRT-based Fully Homomorphic Encryption over the Integers. *IACR Cryptol. ePrint Arch.* **2013**, *2013*, 57.
19. Wang, X.; Malozemoff, A.J.; Katz, J. Faster Secure Two-Party Computation in the Single-Execution Setting. In Proceedings of the Advances in Cryptology—EUROCRYPT 2017—36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, 30 April–4 May 2017; Part III; Coron, J., Nielsen, J.B., Eds.; Volume 10212, pp. 399–424.
20. Keller, M. MP-SPDZ: A Versatile Framework for Multi-Party Computation. *IACR Cryptol. ePrint Arch.* **2020**, *2020*, 521.
21. Rivest, R.L.; Adleman, L.; Dertouzos, M.L. On data banks and privacy homomorphisms. *Found. Secur. Comput.* **1978**, *4*, 169–180.
22. Gentry, C. Fully homomorphic encryption using ideal lattices. In Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, 31 May—2 June 2009; Mitzenmacher, M., Ed.; ACM: New York, NY, USA, 2009; pp. 169–178.
23. Cheon, J.H.; Kim, A.; Kim, M.; Song, Y.S. Homomorphic Encryption for Arithmetic of Approximate Numbers. In Proceedings of the Advances in Cryptology—ASIACRYPT 2017—23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, 3–7 December 2017; Proceedings, Part I; Takagi, T., Peyrin, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2017; Volume 10624, pp. 409–437.
24. Zhou, J.; Dong, X.; Cao, Z.; Vasilakos, A.V. Secure and privacy preserving protocol for cloud-based vehicular DTNs. *IEEE Trans. Inf. Forensics Secur.* **2015**, *10*, 1299–1314. [CrossRef]
25. Ren, W.; Tong, X.; Du, J.; Wang, N.; Li, S.; Min, G.; Zhao, Z.; Bashir, A.K. Privacy-preserving using homomorphic encryption in Mobile IoT systems. *Comput. Commun.* **2021**, *165*, 105–111. [CrossRef]
26. Brakerski, Z.; Gentry, C.; Vaikuntanathan, V. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory (TOCT)* **2014**, *6*, 1–36. [CrossRef]
27. Damgård, I.; Pastro, V.; Smart, N.P.; Zakarias, S. Multiparty Computation from Somewhat Homomorphic Encryption. In Proceedings of the Advances in Cryptology—CRYPTO 2012—32nd Annual Cryptology Conference, Santa Barbara, CA, USA, 19–23 August 2012; Safavi-Naini, R., Canetti, R., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7417, pp. 643–662.
28. Keller, M.; Orsini, E.; Scholl, P. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S., Eds.; ACM: New York, NY, USA, 2016; pp. 830–842.
29. Keller, M.; Pastro, V.; Rotaru, D. Overdrive: Making SPDZ Great Again. In Proceedings of the Advances in Cryptology—EUROCRYPT 2018—37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, 29 April–3 May 2018; Part III; Nielsen, J.B., Rijmen, V., Eds.; Springer: Berlin/Heidelberg, Germany, 2018; Volume 10822, pp. 158–189.
30. Beaver, D.; Micali, S.; Rogaway, P. The Round Complexity of Secure Protocols (Extended Abstract). In Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, 13–17 May 1990; Ortiz, H., Ed.; ACM: New York, NY, USA, 1990; pp. 503–513.
31. Tzeng, W. Efficient oblivious transfer schemes. *IACR Cryptol. ePrint Arch.* **2001**, 2001, 73.