Towards Practical Topology-Hiding Computation

Shuaishuai Li[1,2]*

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

lishuaishuai@iie.ac.cn

Abstract. Topology-hiding computation (THC) enables n parties to perform a secure multi-party computation (MPC) protocol in an incomplete communication graph while keeping the communication graph hidden. The work of Akavia et al. (CRYPTO 2017 and JoC 2020) shown that THC is feasible for any graph. In this work, we focus on the efficiency of THC and give improvements for various tasks including broadcast, sum and general computation. We mainly consider THC on undirected cycles, but we also give two results for THC on general graphs. All of our results are derived in the presence of a passive adversary statically corrupting any number of parties.

In the undirected cycles, the state-of-the-art topology-hiding broadcast (THB) protocol is the Akavia-Moran (AM) protocol of Akavia et al. (EUROCRYPT 2017). We give an optimization for the AM protocol such that the communication cost of broadcasting $O(\kappa)$ bits is reduced from $O(n^2\kappa^2)$ bits to $O(n^2\kappa)$ bits. We also consider the sum and general computation functionalities. Previous to our work, the only THC protocols realizing the sum and general computation functionalities are constructed by using THB to simulate point-to-point channels in an MPC protocol realizing the sum and general computation functionalities, respectively. By allowing the parties to know the exact value of the number of the parties (the AM protocol and our optimization only assume the parties know an upper bound of the number of the parties), we can derive more efficient THC protocols realizing these two functionalities. As a result, comparing with previous works, we reduce the communication cost by a factor of $O(n\kappa)$ for both the sum and general computation functionalities.

As we have mentioned, we also get two results for THC on general graphs. The state-of-the-art THB protocol for general graphs is the Akavia-LaVigne-Moran (ALM) protocol of Akavia et al. (CRYPTO 2017 and JoC 2020). Our result is that our optimization for the AM protocol also applies to the ALM protocol and can reduce its communication cost by a factor of $O(\kappa)$. Moreover, we optimize the fully-homomorphic encryption (FHE) based GTHC protocol of LaVigne et al. (TCC 2018) and reduce its communication cost from $O(n^8\kappa^2)$ FHE ciphertexts and $O(n^5\kappa)$ FHE public keys to $O(n^6\kappa)$ FHE ciphertexts and $O(n^5\kappa)$ FHE public keys.

1 Introduction

The theory of secure multiparty computation (MPC) has drawn a great deal of attention since introduced by Yao [28] in 1982. In MPC, n parties $P_1, \ldots, P_n$ seek to compute some public function on their private inputs while keeping their inputs secret. There have been a great body of works to make MPC more and more general and efficient. However, most of these works assume that the communication graph is complete, meaning that every two parties can communicate directly, which is not always the case in real-world situations. For example, two parties may can not directly communicate with each other due to their long physical distance or other confidentiality reasons. For this reason, a line of works [20,8,18,19,9] considered designing MPC protocols over incomplete communication graph.

Moran et al. [25] considered a more complicated situation, where the communication graph is not only incomplete but also *sensitive*. They formalized the concept of topology-hiding computation (THC), which aims to design MPC protocols while keeping the graph topology hidden. There are many scenes, such as social networks, ISP networks, vehicle-to-vehicle communications, and other Internet of Things networks, where keeping the graph topology hidden is of great importance.

Motivated by building more efficient THC protocols, we consider the setting where the adversary may statically, passively corrupt up to at most $n-1$ parties (only computational security is possible

* This is the full version of an article that will appear in ASIACRYPT 2022.

in such a setting). A series of works have resolved the feasibility question of THC in this setting. More concretely, the works of [25,22] built THC for graphs with logarithmic diameter[1]. Later, based on a special public-key encryption (PKE) scheme (aka PKCR encryption), the work of [3] built THC for several special graph classes that may have super-logarithmic diameter such as cycles, trees, and graphs with logarithmic circumference[2]. The feasibility of THC on any graph is established in the work of [1], which presented a construction of THC for *all* graphs by combining PKCR encryption and another novel technique called correlated random walks.

In this work, we focus on the efficiency of THC. In the undirected cycles, we follow the work of [3] and derive more efficient THC protocols for various tasks such as broadcast, sum and general computation (computing any circuit consists of addition and multiplication gates). We also extend some of our results and give several improvements for existing THC protocols on general graphs, including the topology-hiding broadcast (THB) protocol of [1] and the fully-homomorphic encryption (FHE) based general topology-hiding computation (GTHC) protocol of [23].

Other related works. There are also several works studying the feasibility of (computationally secure) THC in the fail-stop setting, where the adversary may instruct the corrupt parties to abort the protocol. The works of [6,23] showed how to construct THC protocols with small leakage. Some works studied the possibility of information-theoretic THC. [21] showed that information-theoretically secure MPC inherently leaks information about the graph topology to the adversary, which implies that information-theoretic THC on general graphs is impossible. A natural question is whether information-theoretic THC is possible for some subclasses of graphs, which is the main topic of [5]. Moreover, the work of [4] studied the feasibility of THC in different cryptographic setting: information-theoretic, given other cryptographic primitives such as key agreement and oblivious transfer. Finally, the work of [24] studied the feasibility of THC when assuming the network delay is not known (all other THC works assume the network delay has a known upper bound).

1.1 Our Contribution

As our first result, we give an optimization for the Akavia-Moran (AM) protocol (the state-of-the-art THB protocol for undirected cycles[3]) proposed by [3] and reduce its communication cost by a factor of $O(\kappa)$ in the amortized sense. Concretely, if one party wants to broadcast $O(\kappa)$ bits, the communication cost will be $O(n^2\kappa^2)$ bits using the AM protocol. Our optimization for the AM protocol can reduce the communication cost to $O(n^2\kappa)$ bits.

We then consider the sum and general computation functionalities. Before showing our results[4], we first clarify the state-of-the-art asymptotic communication complexity required for realizing these two functionalities, respectively. As noted in [25,22,3], given THB for some graph class and a PKE scheme, any functionality $\mathcal{F}$ can be topology-hidingly realized for the same graph class by using THB and PKE to simulate point-to-point channels in an MPC protocol realizing $\mathcal{F}$. Concretely, point-to-point channels are simulated as follows.

1. Each party uses THB to broadcast its public key in a setup phase.
2. To send a message x to P_j, P_i encrypts x using the public key of P_j and then uses THB to broadcast the resulting ciphertext.
3. Upon receiving the ciphertext, P_j can decrypt it to get x. Other parties know nothing about x because they do not know the decrypt key.

If the underlying PKE scheme satisfies that the ciphertext length is of the same order as the plaintext length (i.e., the ciphertext length is at most a positive constant multiple of the plaintext length)[5]

[1] The diameter of a graph is the greatest distance between two nodes in the graph.

[2] The circumference of a graph is the maximum length of a cycle in the graph.

[3] The original AM protocol is designed for directed cycles, and in particular, it assumes that all parties only know an upper-bound on n rather than the exact value of n. In this work, we extend this protocol to undirected cycles (which is direct) and moreover, we assume that all parties know the exact value of n. We remark that our optimization also works for the original AM protocol.

[4] Unlike the AM protocol and our optimization for the AM protocol, our THC protocols realizing sum and general computation functionalities rely on that the parties know the exact value of n.

[5] In fact, there are many PKE schemes, including the ElGamal [17] scheme and the Paillier [26] scheme, satisfy this property.

and the underlying THB protocol is instantiated with the AM protocol, we can conclude that the state-of-the-art asymptotic communication complexity of topology-hidingly sending $O(\kappa)$ bits on a cycle is $O(n^2\kappa^2)$ bits (we do not count in the communication cost of step 1 because it can be executed once for all).

As we have said, the only topology-hiding protocols realizing the sum and general computation functionalities are constructed by using THB to simulate point-to-point channels in an MPC protocol realizing these two functionalities. We have clarified the state-of-the-art asymptotic communication complexity of simulating point-to-point channels, hence the left problem is to clarify the state-of-the-art asymptotic communication complexity[6] of realizing these two functionalities (without hiding the topology).

For the sum functionality, to the best of our knowledge, the state-of-the-art asymptotic communication complexity is $O(n\kappa)$ bits, which can be constructed from additively homomorphic encryption (which can be instantiated with the Paillier scheme [26]) as follows.

1. In the setup phase, each party samples a public key and broadcasts it. Let pk be the product of all the public keys.
2. P_1 encrypts its input x_1 with pk and sends the resulting ciphertext c_1 to P_2.
3. For $t = 2$ to $n-1$, upon receiving the ciphertext c_{t-1}, P_t computes an encryption c_t of $\sum_{j=1}^{t} x_j$ by homomorphically adding x_t to c_{t-1} using the additive homomorphism. P_t sends c_t to P_{t+1}.
4. Upon receiving the ciphertext c_{n-1} from P_{n-1}, P_n computes an encryption c_n of $\sum_{j=1}^{n} x_j$ by homomorphically adding x_n to c_{n-1} using the additive homomorphism.
5. Finally, the parties execute a distributed decryption protocol to securely decrypt c_n.

The security of the above scheme is guaranteed by the semantic security of the underlying encryption scheme. If instantiating the additively homomorphic encryption scheme with the Paillier scheme, we argue that the communication cost of the above protocol will be $O(n\kappa)$ bits (we do not count in the communication cost of step 1 because it can be executed once for all), which can be derived from the following two points. Firstly, the ciphertext length of the Paillier scheme is of the same order as its plaintext length, which implies that the communication cost of step 2-4 is $O(n\kappa)$ bits. Secondly, we can find a distributed decryption protocol in [7] for Paillier ciphertexts with communication complexity $O(n\kappa)$ bits, which implies that the communication cost of step 5 can be $O(n\kappa)$ bits. Therefore, we conclude that the total communication cost is $O(n\kappa)$ bits.

Note that the state-of-the-art asymptotic communication complexity of sending or broadcasting $O(\kappa)$ bits is $O(n^2\kappa^2)$ bits, hence the state-of-the-art asymptotic communication complexity of topology-hidingly realizing the sum functionality is $O(n^3\kappa^2)$ bits. Our optimization for the AM protocol can reduce the communication cost to $O(n^3\kappa)$ bits. In this work, we give a new topology-hiding sum (THS) protocol which further reduces the communication cost to $O(n^2\kappa)$ bits.

Now we consider the general computation functionality which computes any circuit consisting of addition and multiplication gates. A THC protocol realizing the general computation functionality is called a GTHC protocol. To the best of our knowledge, in the presence of a passive adversary statically corrupting any number of parties, the state-of-the-art asymptotic communication complexity of MPC realizing the general computation functionality is $O((m+c)n\kappa)$ bits[7] where m and c are the number of inputs and multiplication gates in the circuit, which implies that the state-of-the-art asymptotic communication complexity of GTHC is $O((m+c)n^3\kappa^2)$ bits. Our optimization for the AM protocol can reduce the communication cost to $O((m+c)n^3\kappa)$ bits. In this work, we give a new GTHC protocol with communication complexity $O((m+c)n^2\kappa)$ bits.

Finally, we note that our optimization for the AM protocol also applies to the Akavia-LaVigne-Moran (ALM) protocol (the state-of-the-art THB protocol for general graphs) proposed by [1] and reduces its communication cost from $O(n^5\kappa^3)$ bits to $O(n^5\kappa^2)$ when the broadcast value is of length $O(\kappa)$ bits. Moreover, we consider the FHE-based GTHC protocol proposed by [23], which require the parties to communicate $O(n^8\kappa^2)$ FHE ciphertexts and $O(n^5\kappa)$ FHE public keys. We optimize this protocol such that the communication cost is reduced to $O(n^6\kappa)$ FHE ciphertexts and $O(n^5\kappa)$ FHE public keys.

[6] Because the communication cost of sending a bitstring m is of the same order as that of broadcasting m, we refer to the communication complexity as the number of bits that are sent or broadcast.

[7] Both the arithmetic version of the protocol from [16] and the passive version of the protocol from [15] has communication complexity $O((m+c)n\kappa)$ bits.

We summarize our results by the following theorem.

Theorem 1. *There exist the following THC protocols in the presence of a passive adversary statically corrupting any number of parties:*

- *A THB protocol for undirected cycles with communication cost $O(n^2\kappa)$ bits while the broadcast value is of length $O(\kappa)$ bits.*
- *A THS protocol for undirected cycles with communication cost $O(n^2\kappa)$ bits while each input is of length $O(\kappa)$ bits.*
- *A GTHC protocol for undirected cycles with communication cost $O((m+c)n^2\kappa)$ bits while the underlying ring is of size $2^{O(\kappa)}$.*
- *A THB protocol for general graphs with communication cost $O(n^5\kappa^2)$ bits while the broadcast value is of length $O(\kappa)$ bits.*
- *A GTHC protocol for general graphs with communication cost $O(n^6\kappa)$ FHE ciphertexts and $O(n^5\kappa)$ FHE public keys.*

A comparison of our results to previous works is presented in Table 1.

Topology-hiding protocols	Communication complexity	References
THB for cycles	$O(n^2\kappa^2)$ bits	[3]
	$O(n^2\kappa)$ bits	Sect. 3
THS for cycles	$O(n^3\kappa^2)$ bits	[3]
	$O(n^2\kappa)$ bits	Sect. 4
GTHC for cycles	$O((m+c)n^3\kappa^2)$ bits	[3]
	$O((m+c)n^2\kappa)$ bits	Sect. 5
THB for general graphs	$O(n^5\kappa^3)$ bits	[1]
	$O(n^5\kappa^2)$ bits	Sect. 6
FHE-based GTHC for general graphs	$O(n^8\kappa^2)$ hcts $+ O(n^5\kappa)$ hpks	[23]
	$O(n^6\kappa)$ hcts $+ O(n^5\kappa)$ hpks	Sect. 6

Table 1. For all the THC protocols on undirected cycles and the THB protocol for general graphs, we always assume the input size is $O(\kappa)$ bits. The communication costs of the work of [3] for realizing the sum and general computation functionalities are computed as the communication costs of the constructions of THS and GTHC compiled black-box from the AM protocol (assume the parties know the exact value of n in the AM protocol). Additionally, we abbreviate 'FHE ciphertexts' by 'hcts' and 'FHE public keys' by 'hpks'.

1.2 Technical Overview

Before showing how to derive our protocols, we first revisit the AM and ALM protocols. Both of these two THB protocols are only for broadcasting a bit (a bitstring can be broadcast bit-by-bit) and built by first presenting a topology-hiding OR protocol and then letting the broadcaster take the broadcast bit as input and each other party take 0 as input. We present them in the same framework, but with different parameters. The framework consists of two phases: an aggregate phase and a decrypt phase.

At the beginning of the aggregate phase, for each party P_i and each of its neighbor d, P_i samples a fresh public key and encrypts its input bit under this key, and sends the resulting ciphertext (together with the public key) to its neighbor d. At each following round, for each $i \in [n]$, P_i chooses a permutation σ of the set of its neighbors[8] and then for each of its neighbor d, P_i, upon receiving a ciphertext (together with a public key) from its neighbor d at the previous round, homomorphically OR's its own bit and adds a new public key layer to this ciphertext, and then sends the resulting

[8] The AM protocol uses the only non-identity permutation (i.e., each neighbor is mapped to the other neighbor). The ALM protocol uses a fresh random permutation.

ciphertext to its neighbor $\sigma(d)$. After T rounds[9], the parties execute the decrypt phase to decrypt the final ciphertexts. Concretely, each ciphertext is sent back through the same walk it traversed during the aggregate phase, and each party deletes its own public key layer in the reversed walk. Finally, each party derives a bit from each walk starting from itself and outputs the OR of these bits.

We can conclude that the communication cost is $4n(n-1) = O(n^2)$ ciphertexts and $2n(n-1) = O(n^2)$ public keys in the AM protocol and $4|E|\cdot 8n^3\kappa = O(n^5\kappa)$[10] ciphertexts and $2|E|\cdot 8n^3\kappa = O(n^5\kappa)$ public keys in the ALM protocol. The results of [3,1,23] showed that the underlying encryption scheme can be instantiated with the ElGamal scheme [17], the Cock scheme [14] or the Regev scheme [27]. The ciphertext length will be at least $O(\kappa)$ bits if using the ElGamal or Cock scheme and $O(\kappa \log \kappa)$ bits if using the Regev scheme. Moreover, the public key length will be at least $O(\kappa)$ bits if using the ElGamal or Cock scheme and $O(\kappa \log^2 \kappa)$ bits if using the Regev scheme. Therefore, we know that the state-of-the-art communication complexity of the AM and ALM protocols are $O(n^2\kappa)$ and $O(n^5\kappa^2)$ bits, respectively. Note that both of these two protocols can only be used to broadcast a bit, and if we want to broadcast $O(\kappa)$ bits, then the communication cost of the AM and ALM protocols will be $O(n^2\kappa^2)$ and $O(n^5\kappa^3)$ bits, respectively.

THB for undirected cycles and general graphs. The original AM protocol [3] and ALM protocol [1] require the underlying PKE scheme to be OR-homomorphic. In the work of [2], the journal version of [1], the authors observe that designing topology-hiding OR protocol in fact does not require any homomorphic property of the underlying encryption scheme. We restate this observation:

> *To compute OR, upon receiving an encryption of a bit c, the computing party holding a bit b outputs an encryption of c if $b = 0$ and an encryption of 1 otherwise.*

In this observation, whether the computing party changes the encrypted bit depends on what its input is. Our novel idea is that if we only consider broadcast (instead of OR), then we can further extend this observation as follows:

> *To design broadcast, upon receiving an encryption of a bit c, the computing party holding a bit b outputs an encryption of c if the computing party is not the broadcaster (which guarantees that the bit encrypted will not be changed if it has been the broadcast bit) and an encryption of b otherwise (which guarantees that the bit encrypted will be the broadcast bit if it is not yet the broadcast bit).*

The main difference between our observation and the original observation is that in our observation, whether the computing party changes the encrypted bit depends on whether it is the broadcaster rather than what its input is. If the parties act as in our observation, then it is obvious that they can also get the broadcast value even if the broadcast value is not a bit value.

Let us explain how to drive our optimization for the AM and ALM protocols from our observation. In the original AM and ALM protocols, the underlying encryption scheme can be instantiated with the ElGamal scheme. However, to encrypt bits, the actual ElGamal plaintext space is mapped to the set $\{0, 1\}$ while the ciphertext length is still $O(\kappa)$ bits. Note that the ciphertext length of the ElGamal scheme is of the same order as its plaintext length (more precisely, an ElGamal ciphertext is twice the length of the corresponding plaintext), and with our novel observation, any value in the ElGamal plaintext space (instead of $\{0, 1\}$ in the original AM and ALM protocols) can be the broadcast value, which can reduce the communication cost of the AM and ALM protocols by a factor of $O(\kappa)$ in the amortized sense.

THS for undirected cycles. Our THS protocol is based on a simple observation that each walk in the AM protocol passes through each party exactly once during the aggregate phase (which is not right in the original AM protocol where the parties only know an upper bound of n). If we let each party homomorphically add its input to each received ciphertext (assume the underlying encryption is additively homomorphic), then the final ciphertext of each walk is indeed an encryption of the sum of all the inputs. Because the standard ElGamal scheme does not have additive homomorphism, we instantiate the underlying encryption scheme with the scheme from [10] or [13]. Moreover, the ciphertext and public key lengths of both of these two schemes can be $O(\kappa)$ bits when the plaintext length is $O(\kappa)$ bits. Notice that the parties communicate $O(n^2)$ ciphertexts and $O(n^2)$ public keys as in the AM protocol, which leads to the claimed communication cost, i.e., $O(n^2\kappa)$ bits.

[9] T equals $n-1$ in the AM protocol and $8n^3\kappa$ in the ALM protocol.

[10] |E| is the number of edges in the communication graph, which is no more than $C_n^2 = n(n-1)/2$.

GTHC for undirected cycles. Our GTHC protocol also requires that the parties know the exact value of n. Concretely, we consider designing a GTHC protocol within the popular framework based on additive secret sharing. This framework consists of three phases: the input sharing phase, the circuit evaluation phase and the output recovery phase. In the input sharing phase, the parties generate additive sharings for the inputs. In the circuit evaluation phase, the parties perform a protocol to compute an additive sharing of the value of the computed function f (which is represented by an arithmetic circuit consisting of addition and multiplication gates) at the inputs. Finally, in the output recovery phase, the parties recover the output to the parties who are supposed to obtain the output. Because additive secret sharing is linearly homomorphic, the addition gates can be computed locally. Therefore, the key point for designing a GTHC protocol is how to compute a multiplication gate, i.e. how to securely compute an additive sharing of xy with x, y additively shared among the parties. Our starting point is that an additive sharing of xy can be computed by locally adding a public value $xy-r$ to an additive sharing of r where r is a random value. The additive sharing of r can be generated by letting each party P_i locally sample a random value r_i (set $r = \sum_{i\in[n]} r_i$). Now the goal is to publish the value $xy - r$. We present a topology-hiding protocol to achieve this goal in Sect. 5. We remark that the communication cost of this protocol is $O(n^2\kappa)$ bits, which implies the communication cost of computing a multiplication gate is $O(n^2\kappa)$ bits. Moreover, we use our THS protocol to execute the input sharing and output recovery phases such that the communication cost of sharing an input or recovering the output is $O(n^2\kappa)$ bits. Assume f has m inputs and c multiplication gates, then the total communication cost is $O((m + c)n^2\kappa)$ bits.

FHE-based GTHC for general graphs. The work of [23] gave a GTHC protocol based on FHE. We call this protocol the LZM^{3}T protocol. The main advantage of the LZM^{3}T protocol is its low round complexity, which amounts to the round complexity of the ALM protocol. However, if designing a GTHC protocol by compiling an MPC protocol π which realizes the general computation functionality from THB, then the round complexity of the resulting protocol will be k times that of the ALM protocol where k is the round complexity of π.

The LZM^{3}T protocol[11] is constructed by modifying the aggregate phase of the ALM protocol as follows. In the aggregate phase of the LZM^{3}T protocol, each party P_i *appends* the ciphertexts of its input x_i and its ID id_i to each received ciphertext. In such a way, at the end of the aggregate phase, each party P_i will receive $T = 8n^3\kappa$ pairs of ciphertexts $\{c_{t,b}\}_{t\in[T],b\in\{0,1\}}$ (together with the corresponding public key). Let $m_{t,b}$ be the decryption of $c_{t,b}$, then for each $t \in [T]$, there exists $i_t \in [n]$ such that $(m_{t,0}, m_{t,1}) = (x_{i_t}, id_{i_t})$. To compute a given function f, P_i compute an encryption of $f \circ \texttt{parse}$ on $(\{m_{t,b}\}_{t\in[T],b\in\{0,1\}})$, where $\texttt{parse}(\{m_{t,b}\}_{t\in[T],b\in\{0,1\}}) = (x_1, \ldots, x_n)$[12], using the full homomorphism of the underlying encryption. Finally, the parties execute the decrypt phase to decrypt the resulting ciphertexts. The LZM^{3}T has high communication cost because each party sends a ciphertext vector of length $O(t)$ at round t and the total rounds is $T = O(n^3\kappa)$, which yields at least $O((1 + 2 + \cdots + T) \cdot |E|) = O(T^2n^2) = O(n^8\kappa^2)$ ciphertexts communication during the aggregate phase. We optimize the aggregate phase such that $O(n^6\kappa)$ ciphertexts are sufficient[13].

Our idea is that in the aggregate phase, instead of appending an encryption of the input (together with an encryption of the ID) to each received ciphertext vector at each round, each party sends ciphertext vectors of length n at each round and for the i-th entry of the ciphertext vectors, the parties act exactly as in the optimized ALM protocol with P_i being the broadcaster and the input x_i of P_i being the broadcast value. This way, at the end of the aggregate phase, the last party in each walk will get a ciphertext vector of length n where the i-th entry is exactly an encryption of x_i. In particular, the ciphertexts in the same ciphertext vector are under the same public key, which allows the last party in each walk to compute an encryption of the given function using the full

[11] The original protocol works in the fail-stop model where the adversary may instruct any party to abort the execution at any time, but we consider the passive version of this protocol.

[12] The function $\texttt{parse}$ may be derived as follows. For each $i \in [n]$, define the piecewise function h_i such that $h_i(a, b) = a$ if $b = id_i$ and $h_i(a, b) = 0$ if $b \neq id_i$. Then we set $y_i = (\sum_{t\in[T]} h_i(m_{t,0}, m_{t,1}))(\sum_{t\in[T]} m_{t,0}^{-1} h_i(m_{t,0}, m_{t,1}))^{-1}$ and $\texttt{parse} = (y_1, \ldots, y_n)$. Assume (x_i, id_i) appears in the multiset $\{(m_{t,0}, m_{t,1})\}_{t\in[T]}$ k times (the protocol guarantees that $k \geq 1$ with overwhelming probability), then $y_i = kx_i \cdot k^{-1} = x_i$. Therefore, $\texttt{parse}(\{m_{t,b}\}_{t\in[T],b\in\{0,1\}})$ equals $(x_1, \ldots, x_n)$ with overwhelming probability.

[13] More precisely, we reduce the communication cost from $O(n^8\kappa^2)$ ciphertexts and $O(n^5\kappa)$ public keys to $O(n^6\kappa)$ ciphertexts and $O(n^5\kappa)$ public keys.

homomorphism of the underlying encryption. Finally, the decrypt phase is executed. It is obvious that our optimized aggregate phase only requires the parties to send $O(nT \cdot |E|) = O(n^6\kappa)$ ciphertexts.

2 Preliminaries

Notations. Let κ be the security parameter. For any positive integer m, $[m]$ denotes the set $\{1, \cdots, m\}$. We say a function $\varepsilon(\kappa)$ is negligible, denoted $\varepsilon(\kappa) = \mathsf{neg}(\kappa)$, if $\varepsilon(\kappa) = \kappa^{-\omega(1)}$. We say a function $\eta(\kappa)$ is overwhelming if $1 - \eta(\kappa)$ is negligible.

For any set A, let $|A|$ be the cardinality of A and $U(A)$ the uniform distribution over A. For a distribution D, let $x \leftarrow D$ denote the process of sampling x from D. For any two distributions X, Y, denote $\mathsf{SD}(X, Y)$ the statistical distance of X and Y. We say X and Y are identical, denoted $X \equiv Y$, if $\mathsf{SD}(X, Y) = 0$. We say X and Y are statistically indistinguishable, denoted $X \approx_s Y$, if $\mathsf{SD}(X, Y)$ is negligible. Finally, we say X and Y are computationally indistinguishable, denoted $X \approx_c Y$, if no efficient algorithm can distinguish them.

For any plaintext x and a public key pk, we denote $[\![x]\!]_{pk}$ an encryption of x under pk. If the public key is clear from the context, we will omit the public key and use $[\![x]\!]$ to represent an encryption of x under some public key.

2.1 Security Model

For all of our protocols, there are n parties $P_1, \ldots, P_n$ and the communication graph is modelled as an undirected graph $G = (V, E)$ where $V = [n]$ and $(i, j) \in E$ if and only if P_i and P_j can communicate with each other directly (we assume $(i, i) \notin E$ for every $i \in V$). We do not distinguish (i, j) and (j, i) because G is undirected. For any $i \in V$, the set $\mathcal{N}_i = \{j | (i, j) \in E\}$ represents the neighbors of P_i.

Adversarial model. The adversary we consider in this work can statically corrupt any number of parties and moreover, it is passive and computationally bounded (PPT).

Communication model. The concept of THC is formalized by [25], which gave the first (simulation-based) definition for topology hiding in the UC framework [11]. In the work of [1], a stronger variant of this definition is considered. In this work, we adopt this variant in our protocols.

In traditional UC model for MPC, the communication graph is assumed to be complete, i.e. each party can communicate directly with other parties. However, in the setting of THC, the communication graph is incomplete and private. To capture this, an ideal functionality $\mathcal{F}_{graph}$ is defined to describe what the parties can do in the communication graph and a special party P_{graph} is assumed to hold the communication graph. Concretely, $\mathcal{F}_{graph}$ consists of an initialization phase and a communication phase. In the initialization phase, $\mathcal{F}_{graph}$ receives the communication graph $G = (V, E)$ from P_{graph} and samples a label for each edge $e \in E$, and then send the labels of the edges in $\mathcal{N}_i$ to P_i for each $i \in [n]$[14]. We note that in such a way, any two parties can tell whether they share an edge, but can not tell whether they share a neighbor. The communication phase provides secure communication between any party and its neighbors, which receives a message and an edge label from some party and sends the message to the other party holding this edge label. The formal description of $\mathcal{F}_{graph}$ is shown in Fig. 1.

Note that in the ideal world, the adversary has the information that P_{graph} sent the corrupted parties because the initialization phase is executed whenever a functionality $\mathcal{F}$ is realized. To capture this, the functionality $\mathcal{F}_{neigh}$ containing only the initialization phase of $\mathcal{F}_{graph}$ is defined. For any functionality $\mathcal{F}$, we use $\mathcal{F}_{neigh}||\mathcal{F}$ to represent composing $\mathcal{F}$ with $\mathcal{F}_{neigh}$. Now we give the security definition of THC in the UC model.

Definition 2. *We say that a protocol topology-hidingly realizes a functionality $\mathcal{F}$ if it UC-realizes $\mathcal{F}_{neigh}||\mathcal{F}$ in the $\mathcal{F}_{graph}$-hybrid model.*

[14] In the definition of [25], $\mathcal{F}_{graph}$ gives $\mathcal{N}_i$ to P_i, which gives any two parties the ability to tell whether they share a neighbor.

Functionality $\mathcal{F}_{graph}$

The functionality involves $P_1, \ldots, P_n$ and a special party P_{graph} who takes an undirected graph $G = (V, E)$ as input.

Initialization Phase.

1. Receive the graph $G = (V, E)$ from the party P_{graph}.
2. Choose a random injective function $\psi : E \to [n^2]$ to label each edge with a random element from $[n^2]$.
3. Send $\mathcal{L}_i = \{\psi(i, j) : j \in \mathcal{N}_i\}$ to P_i for each $i \in [n]$.

Communication Phase.

1. Receive from a party P_i a triple (i, h, m) which indicates P_i wants to send a message m to the neighbor on the edge labeled with h.
2. Find j such that $h = \psi(i, j)$. Send (h, m) to P_j where h tells P_j that m is sent by its neighbor on the edge labeled with h.

Fig. 1. The graph functionality $\mathcal{F}_{graph}$

2.2 Privately Key-Commutative and Rerandomizable Encryption

The concept of privately key-commutative and rerandomizable (PKCR) encryption is introduced by [3]. Concretely, a PKCR encryption is a semantically secure PKE scheme $(\texttt{Keygen}, \texttt{Enc}, \texttt{Dec})$ with several additional properties. Denote $\mathcal{M}$ the plaintext space, $\mathcal{C}$ the ciphertext space, $\mathcal{PK}$ the public key space which forms an abelian group under the operation $\circledast$ and $\mathcal{SK}$ the secret key space. PKCR encryption requires the following properties.

- *Public-key rerandomizable*: For any $k \in \mathcal{PK}$, it holds that

$$\{k \circledast pk | (pk, sk) \leftarrow \texttt{Keygen}(1^\kappa)\} \approx_s \{pk | (pk, sk) \leftarrow \texttt{Keygen}(1^\kappa)\}.$$

- *Ciphertext rerandomizable*: There exists an efficient algorithm $\texttt{Rand} : \mathcal{C} \times \mathcal{PK} \to \mathcal{C}$ such that for any key pair (pk, sk) and any ciphertext $c = [\![x]\!]_{pk}$, it holds that

$$(x, pk, c, \texttt{Rand}(c, pk)) \approx_s (x, pk, c, \texttt{Enc}(x, pk))$$

  and

$$\texttt{Dec}(\texttt{Rand}(c, pk), sk) = x.$$

- *Privately key-commutative*: There exist two efficient algorithms $\texttt{AddLayer} : \mathcal{C} \times \mathcal{PK} \times \mathcal{SK} \to \mathcal{C}$ and $\texttt{DelLayer} : \mathcal{C} \times \mathcal{PK} \times \mathcal{SK} \to \mathcal{C}$ such that for any two key pairs $(pk_1, sk_1), (pk_2, sk_2)$ and any ciphertext $c = [\![x]\!]_{pk_1}$, it holds that

$$\texttt{AddLayer}(c, pk_1, sk_2) \approx_s \texttt{Enc}(x, pk_1 \circledast pk_2)$$

  and

$$\texttt{DelLayer}(c, pk_1, sk_2) \approx_s \texttt{Enc}(x, pk_1 \circledast pk_2^{-1}).$$

  For the special case that (pk, sk) is a pair of keys, we let $\texttt{DelLayer}(c, pk, sk)$ output $\texttt{Dec}(c, sk)$ instead of $\texttt{Enc}(x, 1)$.

In this work, some of our protocols require the PKCR to be homomorphic, hence we introduce the following additional properties for PKCR.

Equipping PKCR with homomorphism. Our THS protocol requires a PKCR with two additional properties.

– *Plaintext space forms a ring*: The plaintext space $\mathcal{M}$ is a ring $\mathcal{M}_r$ with the operations $+$ (addition) and $\cdot$ (multiplication).

– *Additively homomorphic*: There exists an efficient algorithm $\texttt{Add} : \mathcal{M}_r \times \mathcal{C} \times \mathcal{PK} \to \mathcal{C}$ such that for any plaintext $y \in \mathcal{M}_r$ and any ciphertext $c = [\![x]\!]_{pk}$, it holds that

$$\texttt{Add}(y, c, pk) \approx_s \texttt{Enc}(x + y, pk).$$

We call PKCR encryption with the above two properties additively homomorphic PKCR (ahPKCR) encryption.

Our GTHC protocol (for cycles) requires a stronger variant of ahPKCR, and we call this variant linearly homomorphic PKCR (lhPKCR) encryption. Concretely, lhPKCR requires a linear homomorphism described as follows.

– *Linearly homomorphic*: There exists an efficient algorithm $\texttt{Linear} : \mathcal{M}_r \times \mathcal{C}^2 \times \mathcal{PK} \to \mathcal{C}$ such that for any plaintext $a \in \mathcal{M}_r$ and any two ciphertexts $c_1 = [\![x]\!]_{pk}, c_2 = [\![y]\!]_{pk}$, it holds that

$$\texttt{Linear}(a, c_1, c_2, pk) \approx_s \texttt{Enc}(ax + y, pk).$$

Remark. The work of [3] has proved that the standard ElGamal scheme is a PKCR encryption. In Appendix A we prove that both schemes from [10] and [13] are lhPKCR encryption. In this work, we also instantiate ahPKCR with one of these two schemes (lhPKCR encryption is also ahPKCR encryption).

3 Topology-Hiding Broadcast for Undirected Cycles

The AM protocol [3] is designed for broadcasting a bit, which we abbreviate by bit-THB. We seek to design a THB protocol which directly broadcasts a bitstring instead of a bit, we abbreviate this by string-THB. Notice that string-THB protocol can be simply constructed by just calling the AM protocol bit-by-bit. However, we seek to derive more efficient constructions than this naive way.

In this section, our main result is an optimization for the AM protocol, which will reduce its communication complexity by a factor of $O(\kappa)$ in the amortized sense. Throughout this section, we use the following public parameters.

– $(\texttt{Keygen}, \texttt{Enc}, \texttt{Dec}, \texttt{Rand}, \texttt{AddLayer}, \texttt{DelLayer})$ is a PKCR encryption scheme.
– $\mathcal{M}$ is the plaintext space and $\alpha \in \mathcal{M}$ is a dummy value known by all parties (e.g., α is the identity element if $\mathcal{M}$ is a group).

We aim to design a topology-hiding protocol that can be used to broadcast any element in $\mathcal{M}$. Concretely, we seek to realize the functionality $\mathcal{F}_{bc}$ described in Fig. 4.

3.1 The Protocol

Similar to the AM protocol, our protocol π_{bc} consists of an aggregate phase and a decrypt phase. In our protocol, each party names its two neighbors 0 and 1. At the beginning of the aggregate phase, for each party P_i and each of its neighbor b, P_i samples a fresh public key and encrypts α with this key, and sends the resulting ciphertext (together with the public key) to its neighbor b. At each following round, for each $i \in [n]$ and $b \in \{0, 1\}$, upon receiving a ciphertext (together with a public key k) from the neighbor b at the previous round, P_i samples a fresh public key pk and then encrypts the broadcast value with the key $k \circledast pk$ if it is the broadcaster and adds the public key layer pk to the received ciphertext otherwise. Let c be the resulting ciphertext, then P_i sends c and $k \circledast pk$ to its neighbor $\bar{b} = 1 - b$. After $n - 1$ rounds, the parties execute a decrypt phase to decrypt the final ciphertexts (the decrypt phase is the same as in the AM protocol). Finally, the broadcaster outputs the broadcast value x and each other party outputs one of the decrypted values.

Protocol π_{bc}

Input: The broadcaster takes x as input. α is a dummy value known by all parties.
Output: All parties get x as output.

For each $i \in [n]$, P_i does the following.

Sample $(pk^{(t)}_{i\to b}, sk^{(t)}_{i\to b}) \leftarrow \texttt{Keygen}(1^\kappa)$ for each $t \in [n-1], b \in \{0,1\}$.
% Aggregate Phase
Compute $c^{(1)}_{i\to b} \leftarrow \texttt{Enc}(\alpha, pk^{(1)}_{i\to b})$ and set $k^{(1)}_{i\to b} = pk^{(1)}_{i\to b}$ for each $b \in \{0,1\}$.
Send $c^{(1)}_{i\to b}$ and $k^{(1)}_{i\to b}$ to neighbor b for each $b \in \{0,1\}$.
for $t = 1$ to $n-2$ **do**
  For each $b \in \{0,1\}$, let $c^{(t)}_{i\leftarrow b}$ and $k^{(t)}_{i\leftarrow b}$ be the ciphertext and public key received from neighbor b at the previous round.
  Compute $k^{(t+1)}_{i\to b} = k^{(t)}_{i\leftarrow \bar{b}} \circledast pk^{(t+1)}_{i\to b}$ for each $b \in \{0,1\}$.
  if P_i is the broadcaster **then**
    Compute $c^{(t+1)}_{i\to b} \leftarrow \texttt{Enc}(x, k^{(t+1)}_{i\to b})$ for each $b \in \{0,1\}$.
  else
    Compute $c^{(t+1)}_{i\to b} \leftarrow \texttt{AddLayer}(c^{(t)}_{i\leftarrow \bar{b}}, k^{(t)}_{i\leftarrow \bar{b}}, sk^{(t+1)}_{i\to b})$ for each $b \in \{0,1\}$.
  end if
  Send $c^{(t+1)}_{i\to b}, k^{(t+1)}_{i\to b}$ to neighbor b for each $b \in \{0,1\}$.
end for
For each $b \in \{0,1\}$, let $c^{(n-1)}_{i\leftarrow b}$ and $k^{(n-1)}_{i\leftarrow b}$ be the ciphertext and public key received from neighbor b at the previous round.
if P_i is the broadcaster **then**
  Compute $e^{(n-1)}_{i\to b} \leftarrow \texttt{Enc}(x, k^{(n-1)}_{i\leftarrow b})$ for each $b \in \{0,1\}$.
else
  Compute $e^{(n-1)}_{i\to b} \leftarrow \texttt{Rand}(c^{(n-1)}_{i\leftarrow b}, k^{(n-1)}_{i\leftarrow b})$ for each $b \in \{0,1\}$.
end if
% Decrypt Phase
for $t = n-1$ to 1 **do**
  Send $e^{(t)}_{i\to b}$ to neighbor b for each $b \in \{0,1\}$.
  for $b = 0$ to 1 **do**
    Let $e^{(t)}_{i\leftarrow b}$ be the ciphertext received from neighbor b at the previous round.
    Compute $e^{(t-1)}_{i\to \bar{b}} \leftarrow \texttt{DelLayer}(e^{(t)}_{i\leftarrow b}, k^{(t)}_{i\to b}, sk^{(t)}_{i\to b})$.
  end for
end for
if P_i is the broadcaster **then**
  return x.
else
  return $e^{(0)}_{i\to 0}$.
end if

Remark. We discuss a naive idea to halve the round complexity of π_{bc}, which evidences that hiding the topology is a non-trivial cryptographic task. In the protocol π_{bc}, there are two walks starting from each party P_i and P_i derives one value from each of these two walks at the end of the protocol. An observation is that to get the broadcast value, it is sufficient for each party P_i that one of the two walks starting from P_i passes the broadcaster: P_i just outputs the value that does not equal the dummy value α. To guarantee that at least one walk passes the broadcaster, it is sufficient that the aggregate phase takes $\lfloor n/2 \rfloor$ rounds instead of $n-1$ rounds. However, this idea is insecure. In fact, as long as the aggregate phase takes less than $n-1$ (and no less than $\lfloor n/2 \rfloor$) rounds, the protocol is not topology-hiding.

- If the aggregate phase takes $T = n-2$ rounds and the adversary $\mathcal{A}$ corrupts two parties P_i, P_j such that P_i and P_j are not neighbors and each of P_i and P_j derives different values from its two walks[15], then $\mathcal{A}$ will know that the distance between P_i and P_j is 2, leaking topology information when $n > 4$ (if $n \leq 4$, any two parties inherently know their distance anyhow). A figure illustration of the attack can be seen in Fig. 2 with parameters $n = 12$ and $T = 10$.
- If the aggregate phase takes $T \in [\lfloor n/2 \rfloor, n-2)$ rounds and the adversary $\mathcal{A}$ corrupts the broadcaster P_b and two parties P_i, P_j such that any two of P_i, P_j and P_b are not neighbors and moreover, P_i derives different values from its two walks and P_j derives the same value from its two walks[16], then $\mathcal{A}$ will know that the distance between P_i and P_b is less than the distance between P_j and P_b. A figure illustration of the attack can be seen in Fig. 3 with parameters $n = 12$ and $T = 6$.

[15] Such i, j exist if $n \geq 4$.
[16] Such i, j exist if $n \geq 8$ and n is even.

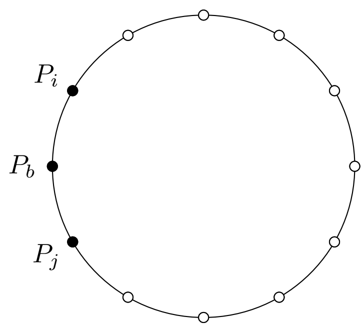

Fig. 2. For $n = 12$ and $T = 10$, P_i and P_j are the only two parties who derive two different values (the broadcast value x and the dummy value α) from the two walks.

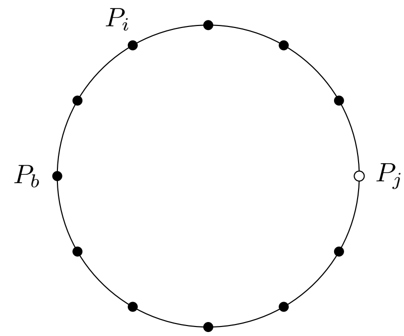

Fig. 3. For $n = 12$ and $T = 6$, P_j is the only party who derives the same value (the broadcast value x) from the two walks.

3.2 Complexity Analysis

Claim 3. *If the underlying PKCR encryption scheme is instantiated with the ElGamal scheme [17], then the communication cost of π_{bc} is $O(n^2\kappa)$ bits while the broadcast value is of length $O(\kappa)$ bits.*

Proof. In the protocol π_{bc}, each party sends each of its two neighbors a single ciphertext and a public key at each round of the aggregate phase and a single ciphertext at each round of the decrypt phase. Let l_1 be the plaintext length of the underlying encryption scheme, l_2 the ciphertext length and l_3 the public key length. Because both the aggregate phase and the decrypt phase takes $n-1$ rounds, the communication complexity of π_{bc} is $2n(n-1)(2l_2+l_3)$ bits. If instantiating the underlying PKCR encryption scheme with the ElGamal scheme [17] and setting $l_1 = O(\kappa)$, then we have $l_2 = 2l_1 = O(\kappa), l_3 = l_1 = O(\kappa)$. Namely, the communication cost of π_{bc} is $O(n^2\kappa)$ bits. □

3.3 Security Proof

Theorem 4. *If the underlying PKCR encryption scheme is semantically secure, then π_{bc} topology-hidingly realizes the functionality $\mathcal{F}_{bc}$ with passive security against any static adversary corrupting any number of parties.*

We defer the proof to Appendix D.1.

4 Topology-Hiding Sum for Undirected Cycles

In this section, we consider the sum functionality. As we have said, previous to this work, the only topology-hiding protocol realizing the sum functionality is constructed by using the AM protocol to simulate the pairwise channels in an MPC protocol realizing the sum functionality, which yields the state-of-the-art asymptotic communication complexity $O(n^3\kappa^2)$ bits. Our optimization for the AM protocol can reduce this communication cost to $O(n^3\kappa)$ bits. We give a new THS protocol which further reduces the communication cost to $O(n^2\kappa)$ bits.

Our starting point is to design THS without compiling black-box from THB, for which we need a PKCR encryption scheme with an additive homomorphism, i.e., an ahPKCR encryption scheme introduced in Sect. 2.2 (such a scheme can be instantiated with the scheme from [10] or [13] as showed in Appendix A). Throughout this section, we use the following parameters.

- $(\texttt{Keygen}, \texttt{Enc}, \texttt{Dec}, \texttt{Rand}, \texttt{AddLayer}, \texttt{DelLayer}, \texttt{Add})$ is an ahPKCR encryption scheme.
- $\mathcal{M}_r$ is the plaintext space, which is a ring[17].

We aim to design a topology-hiding protocol to realize the sum functionality $\mathcal{F}_{sum}$ described in Fig. 5.

[17] $\mathcal{M}_r$ is $\mathbb{Z}_N$ for an RSA modulus N if using the scheme from [10] or $\mathbb{Z}_p$ for a large prime p if using the scheme from [13].

4.1 The Protocol

Our protocol π_{sum} consists of an aggregate phase and a decrypt phase. In our protocol, each party names its two neighbors 0 and 1. At the beginning of the aggregate phase, for each party P_i and each of its neighbor b, P_i samples a fresh public key and encrypts its input x_i with this key, and sends the resulting ciphertext (together with the public key) to its neighbor b. At each following round, for each $i \in [n]$ and $b \in \{0,1\}$, upon receiving a ciphertext (together with a public key k) from its neighbor b at the previous round, P_i homomorphically adds its input to the received ciphertext using the additive homomorphism of ahPKCR. Let c be the resulting ciphertext, then P_i adds a fresh public key layer pk to c and sends the resulting (layered) ciphertext and $k \circledast pk$ to its neighbor $\bar{b} = 1 - b$. After $n-1$ rounds, the parties execute the decrypt phase to decrypt the final ciphertexts. Finally, each party outputs one of the decrypted values.

Protocol π_{sum}

Input: Each party P_i takes $x_i \in \mathcal{M}_r$ as input.
Output: All parties get $x = \sum_{i\in[n]} x_i$.

For each $i \in [n]$, P_i does the following.

```
Sample (pk^{(t)}_{i→b}, sk^{(t)}_{i→b}) ← Keygen(1^κ) for each t ∈ [n−1], b ∈ {0,1}.
% Aggregate Phase
Compute c^{(1)}_{i→b} ← Enc(x_i, pk^{(1)}_{i→b}) and set k^{(1)}_{i→b} = pk^{(1)}_{i→b} for each b ∈ {0,1}.
Send c^{(1)}_{i→b} and k^{(1)}_{i→b} to neighbor b for each b ∈ {0,1}.
for t = 1 to n − 2 do
    For each b ∈ {0,1}, let c^{(t)}_{i←b} and k^{(t)}_{i←b} be the ciphertext and public key received from neighbor b at the previous round.
    Compute k^{(t+1)}_{i→b} = k^{(t)}_{i←b̄} ⊛ pk^{(t+1)}_{i→b} for each b ∈ {0,1}.
    Compute c_b ← AddLayer(c^{(t)}_{i←b̄}, k^{(t)}_{i←b̄}, sk^{(t+1)}_{i→b}) for each b ∈ {0,1}.
    Compute c^{(t+1)}_{i→b} ← Add(x_i, c_b, k^{(t+1)}_{i→b}) for each b ∈ {0,1}.
    Send c^{(t+1)}_{i→b}, k^{(t+1)}_{i→b} to neighbor b for each b ∈ {0,1}.
end for
For each b ∈ {0,1}, let c^{(n−1)}_{i←b} and k^{(n−1)}_{i←b} be the ciphertext and public key received from neighbor b at the previous round.
Compute e^{(n−1)}_{i→b} ← Add(x_i, c^{(n−1)}_{i←b}, k^{(n−1)}_{i←b}) for each b ∈ {0,1}.
% Decrypt Phase
for t = n − 1 to 1 do
    Send e^{(t)}_{i→b} to neighbor b for each b ∈ {0,1}.
    for b = 0 to 1 do
        Let e^{(t)}_{i←b} be the ciphertext received from neighbor b at the previous round.
        Compute e^{(t−1)}_{i→b̄} ← DelLayer(e^{(t)}_{i←b}, k^{(t)}_{i→b}, sk^{(t)}_{i→b}).
    end for
end for
return e^{(0)}_{i→0}.
```

4.2 Complexity Analysis

Claim 5. *If the underlying ahPKCR encryption scheme is instantiated with the scheme from [10] or [13], then the communication cost of π_{sum} is $O(n^2\kappa)$ bits while each input is of length $O(\kappa)$ bits.*

Proof. In the protocol π_{sum}, each party sends each of its two neighbors a single ciphertext and a public key at each round of the aggregate phase and a single ciphertext at each round of the decrypt phase. Let l_1 be the plaintext length of the underlying encryption scheme, l_2 the ciphertext length and l_3 the public key length. Because both the aggregate phase and the decrypt phase takes $n-1$ rounds, the communication complexity of π_{sum} is $2n(n-1)(2l_2+l_3)$ bits. If the underlying ahPKCR encryption scheme is instantiated with the scheme from [10] or [13], then we can set $l_1 = O(\kappa), l_2 = O(\kappa)$ and $l_3 = O(\kappa)$. Namely, the communication cost of π_{sum} is $O(n^2\kappa)$ bits while each input is of length $O(\kappa)$ bits. $\square$

4.3 Security Proof

Theorem 6. *If the underlying ahPKCR encryption scheme is semantically secure, then π_{sum} topology-hidingly realizes the functionality $\mathcal{F}_{sum}$ with passive security against any static adversary corrupting any number of parties.*

We defer the proof to Appendix D.2.

5 General Topology-Hiding Computation for Undirected Cycles

In this section, we consider the general computation functionality which can compute any arithmetic circuit[18] consisting of addition and multiplication gates. As we have said, previous to this work, the only topology-hiding protocol realizing the general computation functionality is constructed by simulating the pairwise channels in an MPC protocol realizing the general computation functionality, which yields the state-of-the-art asymptotic communication complexity $O((m+c)n^3\kappa^2)$ bits where m and c are the number of inputs and multiplication gates in the circuit, respectively. Our optimization for the AM protocol can reduce the communication cost to $O((m+c)n^3\kappa)$ bits. We present a new GTHC protocol which further reduces the communication cost to $O((m+c)n^2\kappa)$ bits. Our GTHC protocol is designed in the popular MPC framework based on additive secret sharing. There are three phases in this framework: the input sharing phase, the circuit evaluation phase and the output recovery phase.

In the input sharing phase, the parties generate additive sharings for the inputs. In the circuit evaluation phase, the parties evaluate the circuit gate-by-gate. Throughout this phase, the parties maintain the invariant that for every gate, the parties hold additive sharings of the values on the two input wires and get an additive sharing of the value on the output wire. Finally, in the output recovery phase, the parties recover the value on the output wire of the final gate.

We show how to use our THS protocol to deal with the input sharing and output recovery phases in Sect. 5.2. For the circuit evaluation phase, we know that addition gates can be done locally, so the only left problem is how to topology-hidingly (and efficiently) compute the multiplication gates. In Sect. 5.1, we give an efficient topology-hiding protocol to securely compute the multiplication gates.

Throughout this section, we need a lhPKCR[19] encryption scheme introduced in Sect. 2.2 and use the following notations.

- ($\mathtt{Keygen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Rand}, \mathtt{AddLayer}, \mathtt{DelLayer}, \mathtt{Linear}$) is a lhPKCR encryption scheme.
- $\mathcal{M}_r$ is the plaintext space of the lhPKCR scheme.
- For any plaintext $y \in \mathcal{M}_r$ and any ciphertext $c = [\![x]\!]_{pk}$, we define the function $\mathtt{Add}(y, c, pk)$ which outputs $\mathtt{Linear}(1, c, [\![y]\!]_{pk}, pk)$.

Additive secret sharing. An additive sharing of a secret value x is a vector $\langle x \rangle = (x_1, \ldots, x_n)$ where each party P_i holds a share x_i satisfying that any $n-1$ shares leak nothing about x. Additive secret sharing is linearly homomorphic, which means that for any public value c and any two additive sharings $\langle x \rangle = (x_1, \ldots, x_n), \langle y \rangle = (y_1, \ldots, y_n)$, we have

$$\langle x \rangle + \langle y \rangle = \langle x + y \rangle, c\langle x \rangle = \langle cx \rangle, c + \langle x \rangle = \langle c + x \rangle$$

where $c + \langle x \rangle = (c + x_1, x_2, \ldots, x_n)$.

5.1 Computing Multiplication Gates

In this section, we give a topology-hiding protocol to securely compute the multiplication gates. Concretely, we realize the functionality $\mathcal{F}_{mult}$ which receives additive sharings of x and y from the parties and sends an additive sharing of xy to the parties. The detailed description of $\mathcal{F}_{mult}$ can be seen in Fig. 6.

Our starting point is that an additive sharing of xy can be computed as follows.

[18] In this work, we consider circuits over a ring of size $2^{O(\kappa)}$.

[19] Recall that such a scheme can be instantiated with the scheme from [10] or [13] as shown in Appendix A.

1. The parties generate an additive sharing $\langle r \rangle$ for a random value r where the share of P_i is r_i.
2. The parties execute a protocol to let all parties securely get the value $xy - r$.
3. The parties locally compute $\langle xy \rangle = xy - r + \langle r \rangle$.

It is easy to see that the above construction generates an additive sharing of xy. Notice that the generation of $\langle r \rangle$ can be done locally by letting each party sample a random value r_i and setting $r = \sum_{i \in [n]} r_i$. The left problem is how to securely publish the value $xy - r$. To solve this, we define and realize the mask functionality $\mathcal{F}_{mask}$ described in Fig. 7.

5.1.1 The protocol

Now we give a topology-hiding protocol π_{mask} which realizes the functionality $\mathcal{F}_{mask}$. This protocol consists of an aggregate phase and a decrypt phase. The aggregate phase can be viewed as two subphases and each takes $n-1$ rounds. In the first subphase, the parties act exactly as in the aggregate phase of our THS protocol: each party homomorphically adds its share of x to each received ciphertext using the homomorphism of lhPKCR. At the end of the first subphase, every party will get $[\![x]\!]$, an encryption of x, from each walk. Then the parties can execute the second subphase to compute encryptions of $xy - r$, which is based on two observations. The first observation is that $xy - r = \sum_{i \in [n]} (y_i x - r_i)$, which means that $[\![xy - r]\!]$ can be computed from $[\![y_1 x - r_1]\!], \ldots, [\![y_n x - r_n]\!]$ (under the same key) using the homomorphism of lhPKCR. The second observation is that every party P_i can compute $[\![y_i x - r_i]\!]$ from $[\![x]\!]$ using the homomorphism of lhPKCR.

We note that throughout the aggregate phase, each party adds a fresh public key layer to each received ciphertext at each round, which implies that each final ciphertext includes $2n-2$ public key layers (because the aggregate phase takes $2n-2$ rounds). Therefore, the parties execute the decrypt phase, which takes $2n-2$ rounds, to decrypt the final ciphertexts. The formal description of π_{mask} is deferred to the Appendix B.1.

Now we can present our protocol π_{mult} which realizes the functionality $\mathcal{F}_{mult}$ in the $\mathcal{F}_{mask}$-hybrid model.

Protocol π_{mult}

Input: The parties hold additive sharings $\langle x \rangle, \langle y \rangle$.
Output: The parties output $\langle xy \rangle$.

1. Each party P_i samples a random value $r_i \leftarrow U(\mathcal{M}_r)$.
2. The parties invoke the functionality $\mathcal{F}_{mask}$ where each party P_i takes x_i, y_i and r_i as inputs. Let z be the output.
3. P_1 outputs $z + r_1$ and each other party P_i outputs r_i.

5.1.2 Complexity Analysis

Claim 7. *If the underlying lhPKCR encryption scheme is instantiated with the scheme from [10] or [13] and the functionality $\mathcal{F}_{mask}$ is realized by the protocol π_{mask}, then the communication cost of π_{mult} is $O(n^2\kappa)$ bits while each input is of length $O(\kappa)$ bits.*

Proof. It is obvious that the communication complexity of π_{mult} is the same as that of π_{mask}. In the protocol π_{mask}, the aggregate phase takes $2n-2$ rounds, and where each party sends each of its two neighbors a ciphertext and a public key at each round of the first $n-1$ rounds and two ciphertexts and a public key at each round of the last $n-1$ rounds. The decrypt phase takes $2n-2$ rounds, and where each party sends each of its two neighbors a single ciphertext at each round. Let l_1 be the plaintext length of the underlying encryption scheme, l_2 the ciphertext length and l_3 the public key length, then the communication complexity is $2n(n-1)(5l_2 + 2l_3)$ bits. If instantiating the underlying lhPKCR encryption with the scheme from [10] or [13], we can set $l_1 = O(\kappa), l_2 = O(\kappa), l_3 = O(\kappa)$. Namely, the protocol π_{mult} has communication complexity $O(n^2\kappa)$ bits while each input is of length $O(\kappa)$ bits. $\square$

5.1.3 Security Proof

In this section, we first show that π_{mult} securely realizes the functionality $\mathcal{F}_{mult}$ in the $\mathcal{F}_{mask}$-hybrid model and then we show that π_{mask} securely realizes the functionality $\mathcal{F}_{mask}$.

Theorem 8. *Protocol π_{mult} topology-hidingly realizes the functionality $\mathcal{F}_{mult}$ in the $\mathcal{F}_{mask}$-hybrid model with passive security against any static adversary corrupting any number of parties.*

Proof. **Correctness.** The correctness of π_{mult} is guaranteed by the functionality $\mathcal{F}_{mask}$. Let $r = \sum_{i\in[n]} r_i$. The functionality $\mathcal{F}_{mask}$ guarantees that $z = xy - r$. At the end of π_{mult}, P_1 outputs $z_1 = z + r_1$ and each other party P_i outputs $z_i = r_i$. It holds that

$$\sum_{i\in[n]} z_i = z + r_1 + (r_2 + \cdots + r_n) = xy - r + r = xy.$$

Moreover, all r_is are random values, hence $\{z_i\}_{i\in[n]}$ is an additive sharing of xy.

Security. The security is obvious because the parties do not communicate with each other outside the invoking of $\mathcal{F}_{mask}$. □

Theorem 9. *If the underlying lhPKCR encryption scheme is semantically secure, then π_{mask} topology-hidingly realizes the functionality $\mathcal{F}_{mask}$ with passive security against any static adversary corrupting any number of parties.*

We defer the proof to Appendix D.3.

5.2 General Topology-Hiding Computation

In this section, we present our GTHC protocol π_{mpc}, which consists of three phases: the input sharing phase, the circuit evaluation phase and the output recovery phase.

Input sharing. The goal of input sharing is to generate additive sharings for the inputs. A subtle point is that we require that for any sharing $\langle x\rangle$ (assume x is the input of P_i), the adversary cannot know anything about the share of some party P_j if P_i and P_j are honest[20]. Now we consider a naive way with low communication cost to share an input x: the input holder P_i shares x among its closed neighborhood (including itself and its two neighbors) and each other party shares 0 among its closed neighborhood, and then each party takes the sum of the share it kept and the shares received from each of their neighbors as its final share. In this process, for any party P_j who is not in the closed neighborhood of the input holder P_i (i.e., P_j is neither P_i nor a neighbor of P_i), if the adversary corrupts the two neighbors of P_j, then the adversary knows the share of P_j[21].

A simple way to share an input x is that the holder of x samples an additive sharing of x and then sends the shares to the parties by using THB to simulate the point-to-point communication, which yields $O(mn^3\kappa)$ bits communication because there are $O(mn)$ shares ($n-1$ shares should be sent for each input) and sending a share (of length κ bits) costs $O(n^2\kappa)$ bits communication. We adopt a more efficient way to share an input. Assume P_i wants to additively share its input x, then if we let each party P_j sample a share x_j, then the share of P_i is $x_i = x - \sum_{j\neq i} x_j$. Our goal is to let P_i get the value x_i while other parties know nothing about x_i. To do this, we let P_i sample a random value r and the parties execute the protocol π_{sum} where P_i takes $x + r$ as input and each other party P_j takes $-x_j$ as input. At the end of the protocol, the parties will get $y = x + r - \sum_{j\neq i} x_j = x_i + r$. It is obvious that the parties know nothing about x_i because r is uniformly random. On the other hand, P_i can compute $x_i = y - r$. Moreover, the communication cost equals exactly the communication cost of π_{sum}, i.e., $O(n^2\kappa)$ bits. Therefore, the communication cost of sharing m inputs will be $O(mn^2\kappa)$ bits.

[20] If P_i is corrupt, we allow the adversary to know all the shares.

[21] The share of P_j is of the form $x_j = a + b + c$ where a, b are two shares received from its two (corrupted) neighbors (hence the adversary knows a, b) and c is the share it kept. Note that P_j share 0 among its closed neighborhood, which means that the sum of the two shares it sent its two neighbors is $-c$, and hence the adversary knows the value of c. Finally, the adversary can get the share of P_j by computing $a + b + c$.

Circuit evaluation. Let $f : \mathcal{M}_r^m \to \mathcal{M}_r$ be the circuit to be computed and $s_1, \ldots, s_m$ are the inputs. The parties compute the circuit in a precomputed topological order. After the input sharing phase, the parties have gotten the additive sharings of the inputs. For each gate g with inputs x and y, the parties have additive sharings $\langle x \rangle$ and $\langle y \rangle$. If g is an addition gate, the parties locally compute $\langle x + y \rangle = \langle x \rangle + \langle y \rangle$. If g is a multiplication gate, the parties execute the protocol π_{mult} and our protocol guarantees that the outputs of the parties form an additive sharing of $\langle xy \rangle$. At the end of the computation, the parties output $\langle f(s_1, \ldots, s_m) \rangle$, an additive sharing of $f(s_1, \ldots, s_m)$. Because the communication cost of computing a multiplication gate is $O(n^2\kappa)$ bits, the total communication cost of this phase is $O(cn^2\kappa)$ bits where c is the number of the multiplication gates.

Output recovery. Let f_i be the final share of P_i. Our protocol guarantees that $f(s_1, \ldots, s_m) = \sum_{i \in [n]} f_i$. If all parties want to get the value $f(s_1, \ldots, s_m)$, then a simple but inefficient way is that each party P_i uses our THB protocol to broadcast f_i, which will yield $O(n^3\kappa)$ bits communication. A more efficient way is that the parties execute our sum protocol π_{sum} where each party P_i takes f_i as input and the communication cost of this way is $O(n^2\kappa)$ bits.

If we only want one party P_j to get the output, then it can be realized by letting P_j add a random value r to its input and then subtract r from its output after the execution of the protocol π_{sum}.

The formal description of our GTHC protocol π_{mpc} is in the following.

Protocol π_{mpc}

Public parameters: $f : \mathcal{M}_r^m \to \mathcal{M}_r$ is a poly-size circuit over $\mathcal{M}_r$.
Input: The parties hold inputs $s_1, \ldots, s_m$.
Output: The parties output $f(s_1, \ldots, s_m)$.

Input sharing. For each input s_i, the parties do the followings.

1. Let P_j be the input holder of s_i. To share s_i, P_j samples a random value $r \in \mathcal{M}_r$ and each other party P_k samples a random value $s_{i,k} \in \mathcal{M}_r$.
2. The parties execute π_{sum} where P_j takes $s_i + r$ as input and each other party P_k takes $-s_{i,k}$ as input. Let y be the output.
3. P_j computes $s_{i,j} = y - r$. The sharing of s_i is $\langle s_i \rangle = (s_{i,1}, \ldots, s_{i,n})$.

Circuit evaluation. For each gate g, the parties do the followings.

1. Let $\langle a \rangle = (a_1, \ldots, a_n), \langle b \rangle = (b_1, \ldots, b_n)$ be the two sharings on the input wires of g.
2. If g is an addition gate, the parties locally compute $\langle a + b \rangle = \langle a \rangle + \langle b \rangle$.
3. If g is a multiplication gate, the parties execute the protocol π_{mult} where each party P_i takes a_i, b_i as inputs. Let c_i be the output of P_i. The result is $\langle ab \rangle = (c_1, \ldots, c_n)$, an additive sharing of ab.

Output recovery. The parties do the followings.

1. Let $\langle f(s_1, \ldots, s_m) \rangle = (f_1, \ldots, f_n)$ be the final sharing.
2. If all parties wants to get the value $f(s_1, \ldots, s_m)$, the parties execute π_{sum} where each party P_i takes f_i as input.
3. If only one party P_j wants to get the output, then P_j samples a random value $r \in \mathcal{M}_r$. The parties execute π_{sum} where P_j takes $f_j + r$ as input and each other party P_i takes f_i as input. Let y be the output. P_j outputs $f = y - r$.

Complexity analysis. We state the comunication cost of π_{mpc} by the following claim.

Claim 10. *The communication complexity of π_{mpc} is $O((m + c)n^2\kappa)$ bits.*

Proof. Note that the communcation costs of the input sharing, circuit evaluation and output recovery phases are $O(mn^2\kappa)$, $O(cn^2\kappa)$ and $O(n^2\kappa)$ bits, respectively. Therefore, the total communcation cost of π_{mpc} is $O((m + c)n^2\kappa)$ bits. ☐

Security proof. The security of π_{mpc} is guaranteed by the security of π_{sum} and π_{mult} and we omit the details.

6 Topology-Hiding Computation on General Graphs

In this section, we give optimizations for two existing topology-hiding protocols on general graphs. Both of these two protocols rely on the random walk approach [1]. This approach relies on the following lemma [1], which states that in an undirected connected graph G, the probability that a random walk of length $8|V|^3\tau$ covers G is at least $1 - 2^{-\tau}$.

Lemma 11 ([1]). *Let $G = (V, E)$ be an undirected connected graph. Furthermore, let $\mathcal{W}(u, \tau)$ be a random variable whose value is the set of vertices covered by a random walk starting from u and taking $8|V|^3\tau$ steps. It holds that*

$$\Pr_{\mathcal{W}}[\mathcal{W}(u, \tau) = V] \geq 1 - 2^{-\tau}.$$

6.1 Topology-Hiding Broadcast for General Graphs

As we have said, our optimization for the AM protocol also applies to the ALM protocol [1]. We know the ALM protocol is the state-of-the-art THB protocol for general graphs. Our optimization reduces the communication cost of the ALM protocol by a factor of $O(\kappa)$ in the amortized sense. If the broadcast value is of length $O(\kappa)$ bits, then the communication cost of the ALM protocol will be $O(n^5\kappa^3)$ bits. With our optimization, the communication cost can be reduced to $O(n^5\kappa^2)$ bits. Throughout this section, we use the following public parameters.

– $(\texttt{Keygen}, \texttt{Enc}, \texttt{Dec}, \texttt{Rand}, \texttt{AddLayer}, \texttt{DelLayer})$ is a PKCR encryption scheme.
– $\mathcal{M}$ is the plaintext space and $\alpha \in \mathcal{M}$ is a dummy value known by all parties (e.g., α is the identity element if $\mathcal{M}$ is a group).

The protocol. Our protocol π_{ggbc} consists of an aggregate phase and a decrypt phase. At the beginning of the aggregate phase, for each party P_i and each of its neighbor d, P_i samples a fresh public key and encrypts α under this key, and then sends the resulting ciphertext (together with the public key) to neighbor d. At each following round, for each $i \in [n]$ and each of its neighbor d, P_i, upon receiving a ciphertext c (together with a public key k) from its neighbor d at the previous round, samples a fresh public key pk and encrypts the broadcast value with the key $k \circledast pk$ if it is the broadcaster and adds the public key layer pk to the received ciphertext c otherwise, and then sends the resulting ciphertext to its neighbor $\sigma(d)$ (σ is a fresh random permutation of the set of the neighbors of P_i). After $T = 8n^3\kappa$ rounds, the parties execute a decrypt phase as in the ALM protocol to decrypt the final ciphertexts. Finally, the broadcaster outputs the broadcast value x and each other party outputs one of the decrypted values. The detailed description of our protocol π_{ggbc} is deferred to Appendix B.2.

Complexity analysis. The following lemma states the communication cost of our protocol π_{ggbc}.

Claim 12. *If the underlying PKCR encryption scheme is instantiated with the ElGamal scheme, then the communication cost of π_{ggbc} is $O(n^5\kappa^2)$ bits while the broadcast value is of length $O(\kappa)$ bits.*

Proof. In the protocol π_{ggbc}, each party sends each of its neighbors a single ciphertext and a public key at each round of the aggregate phase and a single ciphertext at each round of the decrypt phase. Let l_1 be the plaintext length of the underlying encryption scheme, l_2 the ciphertext length and l_3 the public key length. Because both the aggregate phase and the decrypt phase takes $T = 8n^3\kappa$ rounds, the communication cost of π_{ggbc} is $T \cdot 2|E| \cdot (l_2 + l_3) + T \cdot 2|E| \cdot l_2 = O(n^5\kappa \cdot (l_2 + l_3))$ bits. If instantiating the underlying PKCR encryption scheme with the ElGamal scheme and setting $l_1 = O(\kappa)$, then we have $l_2 = 2l_1 = O(\kappa), l_3 = l_1 = O(\kappa)$. Namely, the communication cost of π_{ggbc} is $O(n^5\kappa^2)$ bits while the broadcast value is of length $O(\kappa)$ bits. □

Security proof. We state the security of π_{ggbc} by the following theorem and defer the proof to Appendix D.4.

Theorem 13. *If the underlying PKCR encryption scheme is semantically secure, then π_{ggbc} topology-hidingly realizes the functionality $\mathcal{F}_{bc}$ with passive security against any static adversary corrupting any number of parties.*

6.2 General Topology-Hiding Computation for General Graphs

In [23], a GTHC protocol (we call it the LZM^{3}T protocol) based on FHE is presented. The main advantage of the LZM^{3}T protocol is its low round complexity, which amounts to the round complexity of the ALM protocol. However, if designing a GTHC protocol by compiling an MPC protocol π, which realizes the general computation functionality, from THB, then the round complexity of the resulting protocol will be k times that of the ALM protocol where k is the round complexity of π.

We first recall the LZM^{3}T protocol, which consists of an aggregate phase and a decrypt phase. At each round of the aggregate phase, each party *appends* encryptions of its input and ID to each of the received ciphertext vectors (hence each ciphertext vector in round t is of length $O(t)$) and sends each neighbor one of the resulting ciphertext vector (together with the corresponding public key). At the end of the aggregate phase, each party receives ciphertext vectors containing encryptions of the inputs and then computes encryptions of the given function f. Finally, the party execute the decrypt phase, where each party sends each of its neighbors a single ciphertext, to decrypt the ciphertexts. We remark that the original LZM^{3}T protocol is designed in the fail-stop model where the adversary may abort the protocol, but we consider its passive version in this work.

To clarify the communication cost of the LZM^{3}T protocol, we note that the underlying encryption scheme of the LZM^{3}T protocol is a so-called deeply fully-homomorphic public-key encryption (DFH-PKE) scheme (which can be viewed as an analogue of PKCR but offers full homomorphism). In the LZM^{3}T protocol, DFH-PKE is instantiated with an FHE scheme and the public keys in different rounds of the LZM^{3}T protocol are of different forms. Concretely, let $\mathcal{C}$ and $\mathcal{PK}$ be the ciphertext space and public key space of the FHE scheme, respectively, then during the aggregate phase of the LZM^{3}T protocol, the public keys sent at the first round are in $\mathcal{PK}$ and the public keys sent at each following round are in $\mathcal{PK} \times \mathcal{C}$ (the ciphertext space of DFH-PKE is always $\mathcal{C}$)[22]. Therefore, the communication cost of the LZM^{3}T protocol is $O(|E| + \sum_{t=2}^{T}(O(t) + 1)|E| + T|E|) = O(T^2|E|) = O(n^8\kappa^2)$ FHE ciphertexts and $T|E| = O(n^5\kappa)$ FHE public keys.

In this section, we give an optimization for the LZM^{3}T protocol such that the communication cost is reduced to $O(n^6\kappa)$ FHE ciphertexts and $O(n^5\kappa)$ FHE public keys. The goal of the aggregate phase of the LZM^{3}T protocol is to collect encryptions of all the inputs. We give an optimized aggregate phase to achieve this goal. Concretely, instead of appending an encryption of the input (together with the ID) to each received ciphertext vector at each round, each party send ciphertext vectors of length n at each round and for the i-th entry of the ciphertext vectors, the parties act exactly as in our optimized THB protocol π_{ggbc} with P_i being the broadcaster and the input x_i of P_i being the broadcast value.

Complexity analysis. Each party sends each of its neighbors n ciphertexts and a public key at each round of the aggregate phase, and a single ciphertext at each round of the decrypt phase. Recall that the public keys sent at the first round belong to $\mathcal{PK}$ and the public keys sent at each following round belong to $\mathcal{PK} \times \mathcal{C}$. Therefore, the total communication cost is $n|E| + (T-1)(n+1)|E| + T|E| = O(nT|E|) = O(n^6\kappa)$ FHE ciphertexts and $T|E| = O(n^5\kappa)$ FHE public keys.

Security proof. The correctness of π_{ggbc} guarantees that the probability p_0 that the i-th entry of a final ciphertext vector at the end of the aggregate phase is an encryption of x_i is overwhelming. Hence, the probability p that for each $i \in [n]$, the i-th entry of a final ciphertext vector is an encryption of x_i satisfies that

$$p = p_0^n = (1 - \mathsf{neg}(\kappa))^n \geq 1 - n \cdot \mathsf{neg}(\kappa),$$

which is overwhelming because $n = \mathsf{poly}(\kappa)$. Furthermore, the full homomorphism of the underlying DFH-PKE scheme guarantees each ciphertext at the beginning of the decrypt phase is an encryption of $f(x_1, \ldots, x_n)$ with overwhelming probability. Therefore, at the end of the decrypt phase, each party get the value $f(x_1, \ldots, x_n)$ with overwhelming probability.

As for the security, the simulator just sends encryptions of 0 during the aggregate phase and encryptions of $f(x_1, \ldots, x_n)$ during the decrypt phase (the public keys are simulated with fresh public keys). The semantic security of the underlying DFH-PKE scheme guarantees that the ciphertexts and public keys in the real world are indistinguishable from the simulated ciphertexts and public keys, respectively.

We omit the details of the security proof because the proof will be much like the proof of Theorem 13 (DFH-PKE provides the required properties for the security proof similar to PKCR).

[22] We refer to [23, Appendix C] for more details about DFH-PKE and its instantiation.

Remark. Another advantage of our optimized protocol is that we only require the underlying scheme to homomorphically compute the given function, which means that if the given function contains only linear gates (addition, addition-by-constant and multiply-by-constant gates), then we only require the underlying scheme has linear homomorphism, i.e. a lhPKCR scheme is sufficient. However, the LZM^{3}T protocol requires the underlying scheme to homomorphically compute a much more complicated function than the given function (as we explained in Sect. 1.2), which makes it impossible to just use a lhPKCR scheme even the given function contains only linear gates.

7 Optimizations

In this section, we give several optimizations to obtain better concrete efficiency.

Improving the concrete efficiency using multi-ElGamal. All of our protocols use ElGamal-like schemes as the underlying PKCR schemes (the ciphertexts are of form (g^r, xh^r) or $(g^r, f^x h^r)$). We can extend the plaintext space of ElGamal-like schemes as follows to obtain better concrete efficiency. Concretely, to encrypt l messages $x_1, \ldots, x_l$, one samples l key pairs $(sk_1, pk_1), \ldots, (sk_l, pk_l)$ and a random value r, and then compute the ciphertext as $(g^r, x_1 pk_1^r, \ldots, x_l pk_l^r)$ or $(g^r, f^{x_1} pk_1^r, \ldots, f^{x_l} pk_l^r)$. The ciphertext length of l messages is $l+1$ group elements. However, if encrypting the l messages independently, then the total length of the resulting ciphertext is $2l$ group elements. The semantic security of such a multi-ElGamal scheme is also based on the DDH assumption in the underlying group.

Better topology-hiding communication on cycles. We give a more efficient topology-hiding realization for point-to-point communication on undirected cycles with knowing n. As we have said, point-to-point communication can be realized by compiling black-box from THB as follows.

1. Each party uses THB to broadcast its public key in a setup phase.
2. To send a message m to P_j, P_i encrypts m with the public key of P_j and then uses THB to broadcast the resulting ciphertext.
3. Upon receiving the ciphertext, P_j can decrypt it to get m. Other parties know nothing about m because they do not know the decrypt key.

If simulating point-to-point communication as above, then the communication cost of topology-hidingly sending a message m will equal the communication cost of topology-hidingly broadcasting a public key and a ciphertext of m (under some PKE scheme). Now we present a better way to realize point-to-point communication such that the communication cost of topology-hidingly sending a message m equals the communication cost of using our optimized THB protocol to broadcast m (rather than a public key and a ciphertext of m), which achieves better concrete efficiency.

Recall that our optimized THB protocol instantiates the underlying PKCR scheme with the ElGamal scheme. The plaintext space of the ElGamal scheme is a group and the ElGamal scheme is homomorphic under the group operation (the group operation is called multiplication), i.e., for any group elements x and y, $[\![xy]\!]_{pk}$ can be efficiently computed given $[\![x]\!]_{pk}$, y and pk. Now we modify our THS protocol as follows. The underlying scheme is replaced with the ElGamal scheme (instead of the scheme from [10] or [13]); each party homomorphically multiplies (instead of adds) its input to each received ciphertext using the homomorphism of ElGamal. It can be easily seen that at the end of the resulting protocol (we call the resulting protocol the product protocol), all parties get the product of all the inputs, and moreover, the communication cost of this resulting protocol equals the communication cost of our optimized THB protocol because both of these two protocols instantiate the underlying encryption scheme with the ElGamal scheme.

Now we show how to use the product protocol to realize point-to-point communication without additional communication cost.

1. To send a message x to P_j, the parties execute this product protocol, and where P_i takes x as input and P_j takes a random group element r as input, and each other party takes the identity group element as input.
2. At the end of the protocol, all parties get the value $y = xr$. P_j computes yr^{-1} as output.

The above execution is a secure realization for point-to-point communication because no parties know the value of x except P_i and P_j, which is guaranteed by the fact that only P_i and P_j know r and other parties know nothing about r (P_i can infer r from x and y).

8 Conclusion and Open Problem

In this work, we give efficient topology-hiding protocols realizing various functionalities, including the broadcast, sum and general computation functionalities. Our results show that when realizing these functionalities in undirected cycles, hiding the topology introduces at most multiplicative overhead of $O(n)$ in the asymptotic communication complexity. An open problem is that whether $O(n)$ is the optimal overhead.

Another direction is to extend our results to the fail-stop setting where the adversary may instruct the corrupted parties to abort the protocol. One of our results is an optimization for the ALM protocol. The work of [23] extended the ALM protocol to the fail-stop setting. A natural question is whether their method also applies to our optimized ALM protocol.

References

1. Akavia, A., LaVigne, R., Moran, T.: Topology-hiding computation on all graphs. In: Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I. pp. 447–467 (2017). https://doi.org/10.1007/978-3-319-63688-7_15
2. Akavia, A., LaVigne, R., Moran, T.: Topology-hiding computation on all graphs. J. Cryptol. **33**(1), 176–227 (2020). https://doi.org/10.1007/s00145-019-09318-y
3. Akavia, A., Moran, T.: Topology-hiding computation beyond logarithmic diameter. In: Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III. pp. 609–637 (2017). https://doi.org/10.1007/978-3-319-56617-7_21
4. Ball, M., Boyle, E., Cohen, R., Kohl, L., Malkin, T., Meyer, P., Moran, T.: Topology-hiding communication from minimal assumptions. In: Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II. pp. 473–501 (2020). https://doi.org/10.1007/978-3-030-64378-2_17
5. Ball, M., Boyle, E., Cohen, R., Malkin, T., Moran, T.: Is information-theoretic topology-hiding computation possible? In: Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part I. pp. 502–530 (2019). https://doi.org/10.1007/978-3-030-36030-6_20
6. Ball, M., Boyle, E., Malkin, T., Moran, T.: Exploring the boundaries of topology-hiding computation. In: Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III. pp. 294–325 (2018). https://doi.org/10.1007/978-3-319-78372-7_10
7. Baum, C., Damgård, I., Toft, T., Zakarias, R.W.: Better preprocessing for secure multiparty computation. In: Applied Cryptography and Network Security - 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings. pp. 327–345 (2016). https://doi.org/10.1007/978-3-319-39555-5_18
8. Beimel, A., Gabizon, A., Ishai, Y., Kushilevitz, E., Meldgaard, S., Paskin-Cherniavsky, A.: Non-interactive secure multiparty computation. In: Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II. pp. 387–404 (2014). https://doi.org/10.1007/978-3-662-44381-1_22
9. Boyle, E., Cohen, R., Data, D., Hubácek, P.: Must the communication graph of MPC protocols be an expander? In: Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III. pp. 243–272 (2018). https://doi.org/10.1007/978-3-319-96878-0_9
10. Bresson, E., Catalano, D., Pointcheval, D.: A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In: Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30 - December 4, 2003, Proceedings. pp. 37–54 (2003). https://doi.org/10.1007/978-3-540-40061-5_3
11. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA. pp. 136–145 (2001). https://doi.org/10.1109/SFCS.2001.959888
12. Canetti, R., Lin, H., Tessaro, S., Vaikuntanathan, V.: Obfuscation of probabilistic circuits and applications. In: Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II. pp. 468–497 (2015). https://doi.org/10.1007/978-3-662-46497-7_19

13. Castagnos, G., Laguillaumie, F.: Linearly homomorphic encryption from $$\mathsf {DDH}$$. In: Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings. pp. 487–505 (2015). https://doi.org/10.1007/978-3-319-16715-2_26
14. Cocks, C.C.: An identity based encryption scheme based on quadratic residues. In: Cryptography and Coding, 8th IMA International Conference, Cirencester, UK, December 17-19, 2001, Proceedings. pp. 360–363 (2001). https://doi.org/10.1007/3-540-45325-3_32
15. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding. pp. 280–299 (2001). https://doi.org/10.1007/3-540-44987-6_18
16. Franklin, M.K., Haber, S.: Joint encryption and message-efficient secure computation. J. Cryptol. **9**(4), 217–232 (1996). https://doi.org/10.1007/BF00189261
17. Gamal, T.E.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings. pp. 10–18 (1984). https://doi.org/10.1007/3-540-39568-7_2
18. Goldwasser, S., Gordon, S.D., Goyal, V., Jain, A., Katz, J., Liu, F., Sahai, A., Shi, E., Zhou, H.: Multi-input functional encryption. In: Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings. pp. 578–602 (2014). https://doi.org/10.1007/978-3-642-55220-5_32
19. Halevi, S., Ishai, Y., Jain, A., Kushilevitz, E., Rabin, T.: Secure multiparty computation with general interaction patterns. In: Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016. pp. 157–168 (2016). https://doi.org/10.1145/2840728.2840760
20. Halevi, S., Lindell, Y., Pinkas, B.: Secure computation on the web: Computing without simultaneous interaction. In: Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings. pp. 132–150 (2011). https://doi.org/10.1007/978-3-642-22792-9_8
21. Hinkelmann, M., Jakoby, A.: Communications in unknown networks: Preserving the secret of topology. Theor. Comput. Sci. **384**(2-3), 184–200 (2007). https://doi.org/10.1016/j.tcs.2007.04.031
22. Hirt, M., Maurer, U., Tschudi, D., Zikas, V.: Network-hiding communication and applications to multi-party protocols. In: Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II. pp. 335–365 (2016). https://doi.org/10.1007/978-3-662-53008-5_12
23. LaVigne, R., Zhang, C.L., Maurer, U., Moran, T., Mularczyk, M., Tschudi, D.: Topology-hiding computation beyond semi-honest adversaries. In: Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part II. pp. 3–35 (2018). https://doi.org/10.1007/978-3-030-03810-6_1
24. LaVigne, R., Zhang, C.L., Maurer, U., Moran, T., Mularczyk, M., Tschudi, D.: Topology-hiding computation for networks with unknown delays. In: Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II. pp. 215–245 (2020). https://doi.org/10.1007/978-3-030-45388-6_8
25. Moran, T., Orlov, I., Richelson, S.: Topology-hiding computation. In: Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I. pp. 159–181 (2015). https://doi.org/10.1007/978-3-662-46494-6_8
26. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding. pp. 223–238 (1999). https://doi.org/10.1007/3-540-48910-X_16
27. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM **56**(6), 34:1–34:40 (2009). https://doi.org/10.1145/1568318.1568324
28. Yao, A.C.: Protocols for secure computations (extended abstract). In: 23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982. pp. 160–164 (1982). https://doi.org/10.1109/SFCS.1982.38

A lhPKCR Cryptosystem

In this section, we give two instantiations for lhPKCR encryption based on the schemes from [10] and [13][23], respectively. We present them in the same framework with different setups. If instantiating the

[23] The scheme of [13] works in the class groups of imaginary quadratic fields and we refer the reader to [13, Appendix B] for more background knowledge.

scheme with the setup described by the function $\texttt{Setup}^\star$, we use the scheme from [10]. If instantiating the scheme with the setup described by the function $\texttt{Setup}^\diamond$, we use the scheme from [13].

$\texttt{Setup}^\star(1^\kappa)$
Choose p, q, p', q' to be distinct odd primes with $p = 2p' + 1$ and $q = 2q' + 1$, and where p' and q' are both κ bits in length. Let $N = pq$ and $N' = p'q'$. Define $\mathcal{G} = QR_{N^2}$ to be the cyclic group of quadratic residues modulo N^2 with a generator g. The order of g is $e = NN'$. Let $\mathcal{F}$ be the subgroup of $\mathcal{G}$ generated by $f = 1+N$. The order of f is N. For any $X \in \mathcal{F}$, we can find the discrete logarithm of X with respect to $1 + N$ by computing $x = (X \ mod \ N^2 - 1)/N$. Set $\mathcal{M}_r = \mathbb{Z}_N, \mathcal{C} = \mathcal{G}^2, \mathcal{PK} = \mathcal{G}, \mathcal{SK} = \mathbb{Z}_{N^3}, \mathcal{R} = \mathbb{Z}_{N^3}$[24], and define the algorithm $\mathsf{Solve}(X) = (X \ mod \ N^2 - 1)/N$.

$\texttt{Setup}^\diamond(1^\kappa)$
Pick p a random $(\kappa - 2)$-bits prime and q a random $(\kappa + 2)$-bits prime such that $pq \equiv 3 \ (mod \ 4)$ and $(p/q) = -1$[25]. Set $\Delta_K = -pq$ and $\Delta_p = p^2 \Delta_K$. Set $f \leftarrow [(p^2, p)]$ in $C(\Delta_p)$ and $\mathcal{F} = \langle f \rangle$. Let r be a small prime such that $r \neq p$ and $(\Delta_K/r) = 1$. Let $\mathfrak{r}$ be a prime ideal of $\mathcal{O}_{\Delta_K}$ lying above r. Sample $k \leftarrow U(\mathbb{Z}_p^*)$ and set $g = [\varphi_p^{-1}(\mathfrak{r}^2)]^p f^k$ in $C(\Delta_p)$. Let $\mathcal{G} = \langle g \rangle$. Set $B = \lceil |\Delta_K|^{3/4} \rceil$. For any $X \in \mathcal{F}$, the algorithm to find the discrete logarithm of X to the base f, denoted by $\mathsf{Solve}(X)$, parses $\mathsf{Red}(X)$ as $(p^2, \tilde{x}p)$ and returns $x = \tilde{x}^{-1} (mod \ p)$[26]. Set $\mathcal{M}_r = \mathbb{Z}_p, \mathcal{C} = \mathcal{G}^2, \mathcal{PK} = \mathcal{G}, \mathcal{SK} = \mathbb{Z}_{Bp}$ and $\mathcal{R} = \mathbb{Z}_{Bp}$.

$\texttt{Keygen}(1^\kappa)$
Sample $sk \leftarrow U(\mathcal{SK})$ and compute $pk = g^{sk}$. Return (pk, sk).

$\texttt{Enc}(x, pk)$
To encrypt a message $x \in \mathcal{M}_r$, sample $r \leftarrow U(\mathcal{R})$ and compute $c^0 = g^r, c^1 = f^x \cdot pk^r$. Finally, return $c = (c^0, c^1)$.

$\texttt{Dec}(c = (c^0, c^1), sk)$
Return $x = \mathsf{Solve}(c^1 \cdot (c^0)^{-sk})$.

$\texttt{Rand}(c = (c^0, c^1), pk)$
Sample $r \leftarrow U(\mathcal{R})$ and return $(c^0 \cdot g^r, c^1 \cdot pk^r)$.

$\texttt{AddLayer}(c = (c^0, c^1), pk_1, sk_2)$
Return $\texttt{Rand}((c^0, c^1 \cdot (c^0)^{sk_2}), pk_1 \cdot g^{sk_2})$.

$\texttt{DelLayer}(c = (c^0, c^1), pk_1, sk_2)$
Return $\texttt{Rand}((c^0, c^1 \cdot (c^0)^{-sk_2}), pk_1 \cdot g^{-sk_2})$.

$\texttt{Linear}(a, c_1 = (c_1^0, c_1^1), c_2 = (c_2^0, c_2^1), pk)$
Return $\texttt{Rand}(((c_1^0)^a \cdot c_2^0, (c_1^1)^a \cdot c_2^1), pk)$.

We first prove the following claim for both schemes from [10] and [13].

Claim 14. *For any $r_0 \in \mathbb{Z}$, it holds that*

$$\{g^{r+r_0} | r \leftarrow U(\mathcal{R})\} \approx_s \{g^r | r \leftarrow U(\mathcal{R})\}.$$

Proof. We give proofs for these two schemes, respectively.

[24] The original scheme from [10] sets $\mathcal{SK} = \mathbb{Z}_n$ and $\mathcal{R} = \mathbb{Z}_{N^2}$. However, we set $\mathcal{SK} = \mathcal{R} = \mathbb{Z}_{N^3}$ to make the scheme a lhPKCR encryption scheme. We note that this modification does not influence the concrete efficiency if instantiating the underlying scheme with the BCP scheme because in our protocol, only ciphertexts and public keys are sent (the ciphertext space and the public key space remain the same).

[25] In the original scheme from [13], p is a random μ-bits prime and q is a random $(2\kappa - \mu)$-bits prime satisfying $\mu \leq \kappa - 2$. In this work, we set $\mu = \kappa - 2$ for simplicity.

[26] $\mathsf{Red}(X)$ outputs the two-integer representation of the unique reduced ideal equivalent to X. As noted in [13, Appendix B], existing results show that $\mathsf{Red}(X)$ can be efficiently computed given X. On the other hand, by the [13, Proposion 1], the output of $\mathsf{Red}(X)$ equals $(p^2, L(y)p)$ where y is the discrete logarithm of X to the base f and $L(y)$ is the odd integer in $[-p, p]$ such that $L(y) = y^{-1} \ mod \ p$.

Proof for the scheme from [10]. The order of g is $e = NN'$. Let $u = \lfloor N^3/e \rfloor$ and $v = N^3 \ mod\ e$. Let X be the distribution $\{g^{r+r_0} | r \leftarrow U(\mathbb{Z}_{N^3})\}$ and Y the distribution $\{g^r | r \leftarrow U(\mathbb{Z}_{N^3})\}$, then it holds that

$$\begin{aligned}
&2 \cdot \mathsf{SD}(X, Y) \\
=& \sum_{t \in \mathbb{Z}_e} |\Pr_{r \leftarrow U(\mathbb{Z}_{N^3})}[g^{r+r_0} \equiv g^t \ mod\ N^2] - \Pr_{r \leftarrow U(\mathbb{Z}_{N^3})}[g^r \equiv g^t \ mod\ N^2]| \\
=& \sum_{t \in \mathbb{Z}_e} |\Pr_{r \leftarrow U(\mathbb{Z}_{N^3})}[r + r_0 \equiv t \ mod\ e] - \Pr_{r \leftarrow U(\mathbb{Z}_{N^3})}[r \equiv t \ mod\ e]|.
\end{aligned}$$

Notice that if $(t \ mod\ e) < v$, then $\Pr_{r \leftarrow U(\mathbb{Z}_{N^3})}[r \equiv t \ mod\ e] = (u+1)/N^3$. If $(t \ mod\ e) \geq v$, then $\Pr_{r \leftarrow U(\mathbb{Z}_{N^3})}[r \equiv t \ mod\ e] = u/N^3$. Therefore, if we define the function $I : \mathbb{Z}_e \rightarrow \{0, 1\}$ as

$$I(x) = \begin{cases} 1, & \text{if } x < v \\ 0, & \text{if } x \geq v \end{cases}$$

then we have $\Pr_{r \leftarrow U(\mathbb{Z}_{N^3})}[r \equiv t \ mod\ e] = (u + I(t \ mod\ e))/N^3$. Thus, it holds that

$$\begin{aligned}
&2 \cdot \mathsf{SD}(X, Y) \\
=& \sum_{t \in \mathbb{Z}_e} |(u + I((t - r_0) \ mod\ e))/N^3 - (u + I(t \ mod\ e))/N^3| \\
=& \sum_{t \in \mathbb{Z}_e} |I((t - r_0) \ mod\ e) - I(t \ mod\ e)|/N^3 \\
\leq& e/N^3.
\end{aligned}$$

It follows from $e = NN' < N^2$ that $e/N^3 < 1/N$, hence $\mathsf{SD}(X, Y)$ is negligible in κ. Therefore, X and Y are statistically indistinguishable.

Proof for the scheme from [13]. The following result has been shown in [13, Section 3.1],

$$\{g^r | r \leftarrow U(\mathcal{R})\} \approx_s \{y | y \leftarrow U(\mathcal{G})\} \tag{1}$$

Because $\mathcal{G}$ is a group, we have

$$\{y | y \leftarrow U(\mathcal{G})\} \equiv \{y \cdot g^{r_0} | y \leftarrow U(\mathcal{G})\} \tag{2}$$

Combine (1) and (2) we have

$$\{g^{r+r_0} | r \leftarrow U(\mathcal{R})\} \approx_s \{g^r | r \leftarrow U(\mathcal{R})\}.$$

This completes the proof. □

To prove that both schemes from [10] and [13] are lhPKCR encryption, we need to prove the following lemmas.

Lemma 15. *For any $k \in \mathcal{PK}$, it holds that*

$$\{k \cdot pk | (pk, sk) \leftarrow \texttt{Keygen}(1^\kappa)\} \approx_s \{pk | (pk, sk) \leftarrow \texttt{Keygen}(1^\kappa)\}.$$

Proof. For both these two schemes, we know that

$$\{pk | (pk, sk) \leftarrow \texttt{Keygen}(1^\kappa)\} \equiv \{g^{sk} | sk \leftarrow U(\mathcal{SK})\}.$$

Notice that in both of these two schemes, we have $\mathcal{SK} = \mathcal{R}$. Therefore, by Claim 14, it holds that

$$\{k \cdot g^{sk} | sk \leftarrow U(\mathcal{SK})\} \approx_s \{g^{sk} | sk \leftarrow U(\mathcal{SK})\},$$

which implies that

$$\{k \cdot pk | (pk, sk) \leftarrow \texttt{Keygen}(1^\kappa)\} \approx_s \{pk | (pk, sk) \leftarrow \texttt{Keygen}(1^\kappa)\}.$$

This completes the proof. □

Lemma 16. *For any key pair (pk, sk) and any ciphertext $c = [\![x]\!]_{pk}$, it holds that*

$$(x, pk, c, \texttt{Rand}(c, pk)) \approx_s (x, pk, c, \texttt{Enc}(x, pk))$$

and

$$\texttt{Dec}(\texttt{Rand}(c, pk), sk) = x.$$

Proof. There exists r_0 such that $c = (c^0, c^1) = (g^{r_0}, f^x \cdot pk^{r_0})$, then by the definition of $\texttt{Rand}$, we have

$$\begin{aligned}
&(x, pk, c, \texttt{Rand}(c, pk)) \\
\equiv&\{(x, pk, (g^{r_0}, f^x \cdot pk^{r_0}), (g^{r_0} \cdot g^r, f^x \cdot pk^{r_0} \cdot pk^r))|r \leftarrow U(\mathcal{R})\} \\
\equiv&\{(x, pk, (g^{r_0}, f^x \cdot pk^{r_0}), (g^{r_0+r}, f^x \cdot pk^{r_0+r}))|r \leftarrow U(\mathcal{R})\}.
\end{aligned}$$

By Claim 14, we have

$$\{(g^{r_0+r}, f^x \cdot pk^{r_0+r})|r \leftarrow U(\mathcal{R})\} \approx_s \{(g^r, f^x \cdot pk^r)|r \leftarrow U(\mathcal{R})\} \equiv \texttt{Enc}(x, pk).$$

Therefore, it holds that

$$(x, pk, c, \texttt{Rand}(c, pk)) \equiv (x, pk, c, \texttt{Enc}(x, pk)).$$

In addition, let c_0 be any output of $\texttt{Rand}(c, pk)$. By the definition of the function $\texttt{Rand}$, there exist r_0 such that $c_0 = (c_0^0, c_0^1) = (c^0 \cdot g^{r_0}, c^1 \cdot pk^{r_0})$, then we have

$$\texttt{Dec}(c_0, sk) = \mathsf{Solve}(c_0^1 \cdot (c_0^0)^{-sk}) = \mathsf{Solve}(c^1 \cdot (c^0)^{-sk}) = x.$$

This completes the proof. ◻

Lemma 17. *For any two key pairs $(pk_1, sk_1), (pk_2, sk_2)$ and any ciphertext $c = [\![x]\!]_{pk_1}$, it holds that*

$$\texttt{AddLayer}(c, pk_1, sk_2) \approx_s \texttt{Enc}(x, pk_1 \cdot pk_2)$$

and

$$\texttt{DelLayer}(c, pk_1, sk_2) \approx_s \texttt{Enc}(x, pk_1 \cdot pk_2^{-1}).$$

Proof. By the difinitions of $\texttt{AddLayer}$ and $\texttt{DelLayer}$, we know

$$\begin{aligned}
\texttt{AddLayer}(c = (c^0, c^1), pk_1, sk_2) &\equiv \texttt{Rand}((c^0, c^1 \cdot (c^0)^{sk_2}), pk_1 \cdot g^{sk_2}) \\
\texttt{DelLayer}(c = (c^0, c^1), pk_1, sk_2) &\equiv \texttt{Rand}((c^0, c^1 \cdot (c^0)^{-sk_2}), pk_1 \cdot g^{-sk_2}).
\end{aligned}$$

We argue that $(c^0, c^1 \cdot (c^0)^{sk_2})$ is an encryption of x under $pk_1 \cdot pk_2$ and $(c^0, c^1 \cdot (c^0)^{-sk_2})$ an encryption of x under $pk_1 \cdot pk_2^{-1}$, which is implied by

$$\begin{aligned}
c^1 \cdot (c^0)^{sk_2} \cdot (c^0)^{-(sk_1+sk_2)} &= c^1 \cdot (c^0)^{-sk_1} = f^x \\
c^1 \cdot (c^0)^{-sk_2} \cdot (c^0)^{-(sk_1-sk_2)} &= c^1 \cdot (c^0)^{-sk_1} = f^x.
\end{aligned}$$

Therefore, by Lemma 16, we have

$$\begin{aligned}
\texttt{Rand}((c^0, c^1 \cdot (c^0)^{sk_2}), pk_1 \cdot pk_2) &\approx_s \texttt{Enc}(x, pk_1 \cdot pk_2) \\
\texttt{Rand}((c^0, c^1 \cdot (c^0)^{-sk_2}), pk_1 \cdot pk_2^{-1}) &\approx_s \texttt{Enc}(x, pk_1 \cdot pk_2^{-1}).
\end{aligned}$$

The proof is completed. ◻

Lemma 18. *For any message $a \in \mathcal{M}_r$ and any two ciphertexts $c_1 = [\![x]\!]_{pk}, c_2 = [\![y]\!]_{pk}$, it holds that*

$$\texttt{Linear}(a, c_1, c_2, pk) \approx_s \texttt{Enc}(ax + y, pk).$$

Proof. By the difinitions of $\texttt{Linear}$, we know

$$\texttt{Linear}(a, c_1 = (c_1^0, c_1^1), c_2 = (c_2^0, c_2^1), pk) \equiv \texttt{Rand}(((c_1^0)^a \cdot c_2^0, (c_1^1)^a \cdot c_2^1), pk).$$

We argue that the ciphertext $((c_1^0)^a \cdot c_2^0, (c_1^1)^a \cdot c_2^1)$ is an encryption of $ax+y$ under pk, which is implied by

$$(c_1^1)^a \cdot c_2^1 \cdot ((c_1^0)^a \cdot c_2^0)^{-sk} = (c_1^1 (c_1^0)^{-sk})^a (c_2^1 (c_2^0)^{-sk}) = f^{ax} \cdot f^y = f^{ax+y}.$$

Therefore, by Lemma 16, we have

$$\texttt{Rand}(((c_1^0)^a \cdot c_2^0, (c_1^1)^a \cdot c_2^1), pk) \approx_s \texttt{Enc}(ax + y, pk).$$

This completes the proof. ◻

Combine Lemma 15, 16, 17 and 18 we know that both schemes from [10] and [13] are lhPKCR encryption.

B Topology-Hiding Protocols

B.1 Detailed Description of π_{mask}

Protocol π_{mask}

Input: Each party P_i has inputs x_i, y_i and r_i.
Output: All parties output $xy - r$ where $x = \sum_{i\in[n]} x_i, y = \sum_{i\in[n]} y_i$ and $r = \sum_{i\in[n]} r_i$.

For each $i \in [n]$, P_i does the following.

Sample $(pk^{(t)}_{i\to b}, sk^{(t)}_{i\to b}) \leftarrow \mathtt{Keygen}(1^\kappa)$ for each $t \in [2n-2], b \in \{0,1\}$.
% Aggregate Phase
Compute $c^{(1)}_{i\to b} \leftarrow \mathtt{Enc}(x_i, pk^{(1)}_{i\to b})$ and set $k^{(1)}_{i\to b} = pk^{(1)}_{i\to b}$ for each $b \in \{0,1\}$.
Send $c^{(1)}_{i\to b}$ and $k^{(1)}_{i\to b}$ to neighbor b for each $b \in \{0,1\}$.
for $t = 1$ to $n-2$ **do**
  for $b = 0$ to 1 **do**
    Let $c^{(t)}_{i\leftarrow b}$ and $k^{(t)}_{i\leftarrow b}$ be the ciphertext and public key received from neighbor b at the previous round.
    Compute $k^{(t+1)}_{i\to \bar{b}} = k^{(t)}_{i\leftarrow b} \circledast pk^{(t+1)}_{i\to\bar{b}}$.
    Compute $c \leftarrow \mathtt{AddLayer}(c^{(t)}_{i\leftarrow b}, k^{(t)}_{i\leftarrow b}, sk^{(t+1)}_{i\to\bar{b}})$.
    Compute $c^{(t+1)}_{i\to\bar{b}} \leftarrow \mathtt{Add}(x_i, c, k^{(t+1)}_{i\to\bar{b}})$.
    Send $c^{(t+1)}_{i\to\bar{b}}, k^{(t+1)}_{i\to\bar{b}}$ to neighbor $\bar{b}$.
  end for
end for
for $b = 0$ to 1 **do**
  Let $c^{(n-1)}_{i\leftarrow b}$ and $k^{(n-1)}_{i\leftarrow b}$ be the ciphertext and public key received from neighbor b at the previous round.
  Compute $k^{(n)}_{i\to\bar{b}} = k^{(n-1)}_{i\leftarrow b} \circledast pk^{(n)}_{i\to\bar{b}}$.
  Compute $c \leftarrow \mathtt{AddLayer}(c^{(n-1)}_{i\leftarrow b}, k^{(n-1)}_{i\leftarrow b}, sk^{(n)}_{i\to\bar{b}})$.
  Compute $c^{(n,0)}_{i\to\bar{b}} \leftarrow \mathtt{Add}(x_i, c, k^{(n)}_{i\to\bar{b}})$.
  Compute $c_r \leftarrow \mathtt{Enc}(-r_i, k^{(n)}_{i\to\bar{b}})$.
  Compute $c^{(n,1)}_{i\to\bar{b}} \leftarrow \mathtt{Linear}(y_i, c^{(n,0)}_{i\to\bar{b}}, c_r, k^{(n)}_{i\to\bar{b}})$.
  Send $c^{(n,0)}_{i\to\bar{b}}, c^{(n,1)}_{i\to\bar{b}}$ and $k^{(n)}_{i\to\bar{b}}$ to neighbor $\bar{b}$.
end for
for $t = n$ to $2n-3$ **do**
  for $b = 0$ to 1 **do**
    Let $c^{(t,0)}_{i\leftarrow b}, c^{(t,1)}_{i\leftarrow b}$ and $k^{(t)}_{i\leftarrow b}$ be the ciphertexts and public key received from neighbor b at the previous round.
    Compute $k^{(t+1)}_{i\to\bar{b}} = k^{(t)}_{i\leftarrow b} \circledast pk^{(t+1)}_{i\to\bar{b}}$.
    Compute $c^{(t+1,0)}_{i\to\bar{b}} \leftarrow \mathtt{AddLayer}(c^{(t,0)}_{i\leftarrow b}, k^{(t)}_{i\leftarrow b}, sk^{(t+1)}_{i\to\bar{b}})$.
    Compute $c_1 \leftarrow \mathtt{AddLayer}(c^{(t,1)}_{i\leftarrow b}, k^{(t)}_{i\leftarrow b}, sk^{(t+1)}_{i\to\bar{b}})$.
    Compute $c_2 \leftarrow \mathtt{Add}(-r_i, c_1, k^{(t+1)}_{i\to\bar{b}})$.
    Compute $c^{(t+1,1)}_{i\to\bar{b}} \leftarrow \mathtt{Linear}(y_i, c^{(t+1,0)}_{i\to\bar{b}}, c_2, k^{(t+1)}_{i\to\bar{b}})$.
    Send $c^{(t+1,0)}_{i\to\bar{b}}, c^{(t+1,1)}_{i\to\bar{b}}$ and $k^{(t+1)}_{i\to\bar{b}}$ to neighbor $\bar{b}$.
  end for
end for
for $b = 0$ to 1 **do**
  Let $c^{(2n-2,0)}_{i\leftarrow b}, c^{(2n-2,1)}_{i\leftarrow b}$ and $k^{(2n-2)}_{i\leftarrow b}$ be the ciphertexts and public key received from neighbor b at the previous round.
  Compute $c \leftarrow \mathtt{Add}(-r_i, c^{(2n-2,1)}_{i\leftarrow b}, k^{(2n-2)}_{i\leftarrow b})$.
  Compute $e^{(2n-2)}_{i\to b} \leftarrow \mathtt{Linear}(y_i, c^{(2n-2,0)}_{i\leftarrow b}, c, k^{(2n-2)}_{i\leftarrow b})$.
end for
% Decrypt Phase
for $t = 2n-3$ to 0 **do**
  Send $e^{(t+1)}_{i\to b}$ to neighbor b for each $b \in \{0,1\}$.
  for $b = 0$ to 1 **do**
    Let $e^{(t+1)}_{i\leftarrow b}$ be the ciphertext received from neighbor b at the previous round.
    Compute $e^{(t)}_{i\to\bar{b}} \leftarrow \mathtt{DelLayer}(e^{(t+1)}_{i\leftarrow b}, k^{(t+1)}_{i\to b}, sk^{(t+1)}_{i\to b})$.
  end for
end for
return $e^{(0)}_{i\to 0}$.

B.2 Detailed Description of π_{ggbc}

Protocol π_{ggbc}

Input: The broadcaster takes x as input. Let d_i be the number of the neighbors of P_i.
Output: All parties output x.

For each $i \in [n]$, P_i does the following.

Set $T = 8\kappa n^3$.
Generate Td_i key pairs: sample $(pk^{(t)}_{i\to d}, sk^{(t)}_{i\to d}) \leftarrow \mathtt{Keygen}(1^\kappa)$ for each $t \in [T], d \in [d_i]$.
Generate $T-1$ random permutations on $[d_i]$: $\sigma_1, \dots, \sigma_{T-1}$. Let σ_0 be the identity permutation.
% Aggregate Phase
Compute $c^{(1)}_{i\to d} \leftarrow \mathtt{Enc}(\alpha, pk^{(1)}_{i\to d})$ and set $k^{(1)}_{i\to d} = pk^{(1)}_{i\to d}$ for each $d \in [d_i]$.
Send $c^{(1)}_{i\to d}$ and $k^{(1)}_{i\to d}$ to neighbor d for each $d \in [d_i]$.
for $t = 1$ to $T-1$ **do**
    for each $d \in [d_i]$ **do**
        Let $c^{(t)}_{i\leftarrow d}$ and $k^{(t)}_{i\leftarrow d}$ be the ciphertext and public key received from neighbor d at the previous round.
        Set $d' = \sigma_t(d)$.
        Compute $k^{(t+1)}_{i\to d'} = k^{(t)}_{i\leftarrow d} \circledast pk^{(t+1)}_{i\to d'}$.
        if P_i is the broadcaster **then**
            Compute $c^{(t+1)}_{i\to d'} \leftarrow \mathtt{Enc}(x, k^{(t+1)}_{i\to d'})$.
        else
            Compute $c^{(t+1)}_{i\to d'} \leftarrow \mathtt{AddLayer}(c^{(t)}_{i\leftarrow d}, k^{(t)}_{i\leftarrow d}, sk^{(t+1)}_{i\to d'})$.
        end if
        Send $c^{(t+1)}_{i\to d'}$ and $k^{(t+1)}_{i\to d'}$ to neighbor d'.
    end for
end for
for each $d \in [d_i]$ **do**
    Let $c^{(T)}_{i\leftarrow d}$ and $k^{(T)}_{i\leftarrow d}$ be the ciphertext and public key received from neighbor d at the previous round.
    if P_i is the broadcaster **then**
        Compute $e^{(T)}_{i\to d} \leftarrow \mathtt{Enc}(x, k^{(T)}_{i\leftarrow d})$.
    else
        Compute $e^{(T)}_{i\to d} \leftarrow \mathtt{Rand}(c^{(T)}_{i\leftarrow d}, k^{(T)}_{i\leftarrow d})$.
    end if
end for
% Decrypt Phase
for $t = T-1$ to 0 **do**
    Send $e^{(t+1)}_{i\to d}$ to neighbor d for each $d \in [d_i]$.
    for each $d \in [d_i]$ **do**
        Let $e^{(t+1)}_{i\leftarrow d}$ be the ciphertext received from neighbor d at the previous round.
        Set $d' = \sigma_t^{-1}(d)$.
        Compute $e^{(t)}_{i\to d'} \leftarrow \mathtt{DelLayer}(e^{(t+1)}_{i\leftarrow d}, k^{(t+1)}_{i\to d}, sk^{(t+1)}_{i\to d})$.
    end for
end for
if P_i is the broadcaster **then**
    return x.
else
    return $e^{(0)}_{i\to 1}$.
end if

C Involved Functionalities

Some important functionalities are presented in this section.

Functionality $\mathcal{F}_{bc}$

1. Receive a message $x \in \mathcal{M}$ from the broadcaster.
2. Send x to all parties $P_1, \dots, P_n$.

Fig. 4. The broadcast functionality $\mathcal{F}_{bc}$

Functionality $\mathcal{F}_{sum}$

1. Receive a message $x_i \in \mathcal{M}_r$ from each party P_i.
2. Send $\sum_{i \in [n]} x_i$ to all parties $P_1, \ldots, P_n$.

Fig. 5. The sum functionality $\mathcal{F}_{sum}$

Functionality $\mathcal{F}_{mult}$

1. Receive the sharings $\langle x \rangle$ and $\langle y \rangle$ from the parties.
2. Recover x and y. Sample an additive sharing $\langle xy \rangle = (z_1, \ldots, z_n)$.
3. Send z_i to P_i for each $i \in [n]$.

Fig. 6. The multiplication functionality $\mathcal{F}_{mult}$

Functionality $\mathcal{F}_{mask}$

1. Receive x_i, y_i and r_i from each party P_i.
2. Compute $z = \sum_{i \in [n]} x_i \sum_{i \in [n]} y_i - \sum_{i \in [n]} r_i$.
3. Send z to all parties.

Fig. 7. The mask functionality $\mathcal{F}_{mask}$

D Missing Security Proofs

All of our proofs will borrow the skeleton of the security proof from [1] for the ALM protocol. Throughout this section, we use the following symbols.

Notations. Let $\mathcal{C} \subsetneq [n]$ be the set of corrupted parties. Define $\mathcal{Q} = \{v \in \mathcal{C} : \mathcal{N}_v \not\subset \mathcal{C}\}$. For each $v \in \mathcal{Q}$, define $\mathcal{H}_v = \{u \in \mathcal{N}_v : u \notin \mathcal{C}\}$ to represent the honest neighbors of P_v.

D.1 Security Proof of Theorem 4

Proof. **Correctness.** We prove that each party P_i will output the broadcast value x at the end of π_{bc}. Note that the broadcaster outputs the broadcast value anyhow, so we only show that each party P_i who is not the broadcaster will output the broadcast value x. Let $w = (P_{i_1} = P_i, P_{i_2}, \ldots, P_{i_n})$ be one walk starting from P_i such that P_{i_2} is the neighbor 0 of P_i (recall that P_i outputs the decryption of the ciphertext received from its neighbor 0). From the view of the walk w, our protocol proceeds as follows.

1. At the beginning of the aggregate phase, P_i samples a key pair (pk_1, sk_1) and encrypts the dummy value α with $k_1 = pk_1$ into c_1. Then, P_i sends c_1, k_1 to P_{i_2}.
2. For $t = 2$ to $n-1$, upon receiving c_{t-1}, k_{t-1} from $P_{i_{t-1}}$, P_{i_t} samples a key pair (pk_t, sk_t) and computes $k_t = k_{t-1} \circledast pk_t$. Then, P_{i_t} computes $c_t \leftarrow \texttt{Enc}(x, k_t)$ if it is the broadcaster and $c_t \leftarrow \texttt{AddLayer}(c_{t-1}, k_{t-1}, sk_t)$ otherwise. Finally, P_{i_t} sends c_t, k_t to $P_{i_{t+1}}$.
3. Upon receiving c_{n-1}, k_{n-1} from $P_{i_{n-1}}$, P_{i_n} computes $e_{n-1} \leftarrow \texttt{Enc}(x, k_{n-1})$ if P_{i_n} is the broadcaster and $e_{n-1} \leftarrow \texttt{Rand}(c_{n-1}, k_{n-1})$ otherwise. Finally, P_{i_n} sends e_{n-1} *back* to $P_{i_{n-1}}$.
4. For $t = n-1$ to 2, upon receiving e_t from $P_{i_{t+1}}$, P_{i_t} computes $e_{t-1} \leftarrow \texttt{DelLayer}(e_t, k_t, sk_t)$ and then sends e_{t-1} to $P_{i_{t-1}}$.
5. Upon receiving the ciphertext e_1 from P_{i_2}, P_i deletes its layer. In fact, P_i decrypts $e_0 \leftarrow \texttt{Dec}(e_1, sk_1)$ because $k_1 = pk_1$. Finally, P_i outputs e_0.

Now we prove that e_0 must be the broadcast value. Because the communication graph is a cycle, the walk w contains all parties and particularly, w passes through each party only once. Assume P_{i_s} is the broadcaster, then by our protocol, c_s is an encryption of the broadcast value. On the other hand, each of $P_{i_{s+1}}, \ldots, P_{i_n}$ just adds a layer to the received ciphertext without changing the underlying plaintext, hence the final ciphertext e_{n-1} is still an encryption of the broadcast value. Because e_0 is the decryption of e_{n-1}, e_0 must be the broadcast value.

Security. The simulator $\mathcal{S}_{bc}$ simulates the view of all parties in $\mathcal{C}$. In fact, $\mathcal{S}_{bc}$ only simulates the view of parties in $\mathcal{Q}$ because the parties in $\mathcal{C}\backslash\mathcal{Q}$ do not interact with honest parties. Recall that α is a dummy value known by all parties. The simulator $\mathcal{S}_{bc}$ needs to play P_u to interact with P_v for each $v \in \mathcal{Q}$ and $u \in \mathcal{H}_v$. $\mathcal{S}_{bc}$ sends the inputs of the parties in $\mathcal{C}$ to $\mathcal{F}_{bc}$ and receives the broadcast value x. $\mathcal{S}_{bc}$ simulates the messages sent by P_u as follows.

Simulator $\mathcal{S}_{bc}$

Simulating the aggregate phase. At each round t (from 1 to $n-1$) of the aggregate phase, $\mathcal{S}_{bc}$ first samples $(pk_t, sk_t) \leftarrow \texttt{Keygen}(1^\kappa)$ and then computes $c_t \leftarrow \texttt{Enc}(\alpha, pk_t)$. $\mathcal{S}_{bc}$ sends c_t and pk_t to P_v. In this round, $\mathcal{S}_{bc}$ receives a public key k_t from P_v.

Simulating the decrypt phase. At each round t (from $n-1$ to 1) of the decrypt phase, $\mathcal{S}_{bc}$ computes $e_t \leftarrow \texttt{Enc}(x, k_t)$ and sends e_t to P_v.

To prove that our protocol is UC secure, we will show that no environment can distinguish whether it is interacting with the simulator $\mathcal{S}_{bc}$ in the ideal world or with the adversary $\mathcal{A}$ in the real world with non-negligible probability. We define the following hybrids (in the $\mathcal{F}_{graph}$-hybrid model).

Hybrid 0. $\mathcal{S}_0$ acts as in the real execution.

Hybrid 1. $\mathcal{S}_1$ acts as $\mathcal{S}_0$ except that during the aggregate phase, the actual (layered) keys P_u sent P_v are replaced with freshly generated keys.

Hybrid 2. $\mathcal{S}_2$ acts the same as $\mathcal{S}_1$ except that during the aggregate phase, each ciphertext P_u sent P_v is an encryption of α instead of the actual value under the same public key.

Hybrid 3. $\mathcal{S}_3$ acts the same as $\mathcal{S}_2$ except that during the decrypt phase, each ciphertext P_u sent P_v is a fresh encryption under the same key (P_v has sent this key to P_u during the aggregate phase) instead of the actual unlayered encryption. Indeed, $\mathcal{S}_3$ acts exactly as $\mathcal{S}_{bc}$.

Now we will show that no environment can distinguish two consecutive hybrids with noticeable probability.

Hybrid 0 and Hybrid 1 are indistinguishable. The two hybrids differ only in the public keys. In Hybrid 0, each public key P_u sent P_v is a product key of form $k \circledast pk$ where k is a public key P_u received from its the other neighbor and pk is a fresh public key sampled by P_u. Because the underlying lhPKCR encryption scheme is public-key rerandomizable, the distribution of $k \circledast pk$ is statistically indistinguishable from that of a freshly sampled public key. This implies the distribution of all the public keys in Hybrid 0 is statistically indistinguishable from that in Hybrid 1. Therefore, Hybrid 0 and Hybrid 1 are statistically indistinguishable.

Hybrid 1 and Hybrid 2 are indistinguishable. The two hybrids differ only in the encrypted messages sent during the aggregate phase. In Hybrid 2, each ciphertext P_u sent P_v is replaced with an encryption of α. Since each ciphertext sent by P_u is an encryption under a fresh public key sampled by P_u, the key is unknown to the adversary (because P_u is honest). Therefore, the semantic security of the underlying lhPKCR scheme guarantees that the ciphertexts P_u sent P_v during the aggregate phase in Hybrid 1 and Hybrid 2 are computationally indistinguishable, which implies that Hybrid 1 and Hybrid 2 are computationally indistinguishable.

Hybrid 2 and Hybrid 3 are indistinguishable. The two hybrids differ only in how the ciphertexts are derived in the decrypt phase. In Hybrid 3, each ciphertext P_u sent P_v is derived by deleting the public key layer of P_u from some ciphertext. In Hybrid 3, each ciphertext P_u sent P_v is a fresh encryption (of the same value under the same public key as in Hybrid 2). Because the lhPKCR encryption is privately key-commutative, the distribution of a ciphertext computed by deleting a public key layer is statistically indistinguishable from that of a fresh encryption under the resulting public key. This implies that the ciphertexts P_u sent P_v during the decrypt phase in Hybrid 2 and Hybrid 3 are statistically indistinguishable. Therefore, Hybrid 2 and Hybrid 3 are statistically indistinguishable. □

D.2 Security Proof of Theorem 6

Proof. **Correctness.** We prove that each party P_i will output the sum of all the inputs, i.e., the value $x = \sum_{j \in [n]} x_j$, at the end of π_{sum}. Let $w = (P_{i_1} = P_i, P_{i_2}, \ldots, P_{i_n})$ be one walk starting from P_i such that P_{i_2} is the neighbor 0 of P_i. From the view of the walk w, our protocol proceeds as follows.

1. At the beginning of the aggregate phase, P_i samples a key pair (pk_1, sk_1) and encrypts its input x_i with $k_1 = pk_1$ into c_1. Finally, P_i sends c_1, k_1 to P_{i_1}.
2. For $t = 2$ to $n-1$, upon receiving c_{t-1}, k_{t-1} from $P_{i_{t-1}}$, P_{i_t} samples a key pair (pk_t, sk_t) and computes $k_t = k_{t-1} \circledast pk_t$. Then, P_{i_t} computes $c \leftarrow \texttt{AddLayer}(c_{t-1}, k_{t-1}, sk_t)$ and $c_t \leftarrow \texttt{Add}(x_{i_t}, c, k_t)$. Finally, P_{i_t} sends c_t, k_t to $P_{i_{t+1}}$.
3. Upon receiving c_{n-1}, k_{n-1}, P_{i_n} computes $e_{n-1} \leftarrow \texttt{Add}(x_{i_n}, c_{n-1}, k_{n-1})$. Finally, P_{i_n} sends e_{n-1} *back* to $P_{i_{n-1}}$.
4. For $t = n-1$ to 2, upon receiving e_t from $P_{i_{t+1}}$, P_{i_t} deletes its layer by computing $e_{t-1} \leftarrow \texttt{DelLayer}(e_t, k_t, sk_t)$ and then sends e_{t-1} to $P_{i_{t-1}}$.
5. Upon receiving the ciphertext e_1 from P_{i_2}, P_i deletes its layer. In fact, P_i decrypts $e_0 \leftarrow \texttt{Dec}(e_1, sk_1)$ because $k_1 = pk_1$. Finally, P_i outputs e_0.

Now we prove that e_0 must be the sum of all the inputs. Because the communication graph is a cycle, the walk w contains all parties and particularly, w passes through each party only once. Note that each party P_{i_t} homomorphically adds its input x_{i_t} to the received ciphertext, hence the final ciphertext e_{n-1} is an encryption of $\sum_{t \in [n]} x_{i_t} = \sum_{j \in [n]} x_j$. Because e_0 is the decryption of e_{n-1}, we have $e_0 = \sum_{j \in [n]} x_j$.

Security. The proof is almost the same as the security proof of Theorem 4 and we only show how does the simulator $\mathcal{S}_{sum}$ work. $\mathcal{S}_{sum}$ sends the inputs of the parties in $\mathcal{C}$ to $\mathcal{F}_{sum}$ and receives the sum x. The simulator $\mathcal{S}_{sum}$ plays P_u to interact with P_v for each $v \in \mathcal{Q}$ and $u \in \mathcal{H}_v$. Concretely, $\mathcal{S}_{sum}$ simulates the messages sent by P_u as follows.

Simulator $\mathcal{S}_{sum}$

Simulating the aggregate phase. At each round t (from 1 to $n-1$) of the aggregate phase, $\mathcal{S}_{sum}$ first samples $(pk_t, sk_t) \leftarrow \texttt{Keygen}$ and then computes $c_t \leftarrow \texttt{Enc}(0, pk_t)$. $\mathcal{S}_{sum}$ sends c_t, pk_t to P_v. In this round, $\mathcal{S}_{sum}$ receives a public key k_t from P_v.

Simulating the decrypt phase. At each round t (from $n-1$ to 1) of the decrypt phase, $\mathcal{S}_{sum}$ computes $e_t \leftarrow \texttt{Enc}(x, k_t)$ and sends e_t to P_v.

The proof that no environment can distinguish whether it is interacting with the simulator $\mathcal{S}_{sum}$ in the ideal world or with the adversary $\mathcal{A}$ in the real world with non-negligible probability is the same as the security proof of Theorem 4 and we omit the details. □

D.3 Security Proof of Theorem 9

Proof. **Correctness.** We will show that each party P_i will output the value $xy - r$ at the end of π_{mask}. Let $w = (P_{i_1} = P_i, \ldots, P_{i_n}, P_i, \ldots, P_{i_{n-1}})$ be one walk starting from P_i such that P_{i_2} is the neighbor 0 of P_i. From the view of the walk w, our protocol proceeds as follows. Note that if $n < t < 2n$, $i_t = i_{t-n}$.

1. At the beginning of the aggregate phase, P_i samples a key pair (pk_1, sk_1) and encrypts its input x_i with $k_1 = pk_1$ into c_1. Finally, P_i sends c_1, k_1 to P_{i_1}.
2. For $t = 2$ to $n-1$, upon receiving c_{t-1}, k_{t-1} from $P_{i_{t-1}}$, P_{i_t} samples a key pair (pk_t, sk_t) and computes $k_t = k_{t-1} \circledast pk_t$. Then, P_{i_t} computes $c \leftarrow \texttt{AddLayer}(c_{t-1}, k_{t-1}, sk_t)$ and $c_t \leftarrow \texttt{Add}(x_{i_t}, c, k_t)$. Finally, P_{i_t} sends c_t, k_t to $P_{i_{t+1}}$.

3. Upon receiving c_{n-1}, k_{n-1} from $P_{i_{n-1}}$, P_{i_n} samples a key pair (pk_n, sk_n). Then, P_{i_n} computes $k_n = k_{n-1} \circledast pk_n, c \leftarrow \texttt{AddLayer}(c_{n-1}, k_{n-1}, sk_n)$ and $c_n^0 \leftarrow \texttt{Add}(x_{i_n}, c, k_n)$. Now P_{i_n} computes $c_r \leftarrow \texttt{Enc}(-r_{i_n}, k_n)$ and $c_n^1 \leftarrow \texttt{Linear}(y_{i_n}, c_n^0, c_r, k_n)$. Finally, P_{i_n} sends c_n^0, c_n^1 and k_n to P_i.
4. For $t = n+1$ to $2n-2$, upon receiving c_{t-1}^0, c_{t-1}^1 and k_{t-1} from $P_{i_{t-1}}$, P_{i_t} samples a key pair (pk_t, sk_t). Then, P_{i_t} computes $k_t = k_{t-1} \circledast pk_t$, $c_t^0 \leftarrow \texttt{AddLayer}(c_{t-1}^0, k_{t-1}, sk_t)$ and $c_1 \leftarrow \texttt{AddLayer}(c_{t-1}^1, k_{t-1}, sk_t)$. Now P_{i_t} computes $c_2 \leftarrow \texttt{Add}(-r_{i_t}, c_1, k_t)$ and $c_t^1 \leftarrow \texttt{Linear}(y_{i_t}, c_t^0, c_2, k_t)$. Finally, P_{i_t} send c_t^0, c_t^1 and k_t to its neighbor $P_{i_{t+1}}$.
5. Upon receiving c_{2n-2}^0, c_{2n-2}^1 and k_{2n-2} from $P_{i_{2n-2}}$, $P_{i_{2n-1}}$ computes $c \leftarrow \texttt{Add}(-r_{i_{2n-1}}, c_{2n-2}^1, k_{2n-2})$ and $e_{2n-2} \leftarrow \texttt{Linear}(y_{i_{2n-1}}, c_{2n-2}^0, c, k_{2n-2})$. Finally, $P_{i_{2n-1}}$ sends e_{2n-2} *back* to $P_{i_{2n-2}}$.
6. For $t = 2n-2$ to 2, upon receiving e_t from $P_{i_{t+1}}$, P_{i_t} deletes its layer by computing $e_{t-1} \leftarrow \texttt{DelLayer}(e_t, k_t, sk_t)$ and then sends e_{t-1} to $P_{i_{t-1}}$.
7. Upon receiving the ciphertext e_1 from P_{i_2}, P_i deletes its layer. In fact, P_i decrypts $e_0 \leftarrow \texttt{Dec}(e_1, sk_1)$ because $k_1 = pk_1$. Finally, P_i outputs e_0.

Now we show that e_0 must be the value $xy - r$. During Step 1-3, each party adds its share of x to the received ciphertext, hence the ciphertext c_n^0 is an encryption of x. In fact, each c_t^0 for $t \geq n$ is an encryption of x. Moreover, c_n^1 is an encryption of $y_{i_n}x - r_{i_n}$ by the property of the function $\texttt{Linear}$. During Step 4-5, each party P_{i_t} homomorphically adds $y_{i_t}x - r_{i_t}$ to c_{t-1}^1, hence the final ciphertext e_{2n-2} at the end of Step 5 is an encryption of $\sum_{t=n}^{2n-1} y_{i_t}x - r_{i_t} = \sum_{t=1}^{n} y_{i_t}x - r_{i_t} = xy - r$. Note that e_0 is the decryption of e_{2n-2}, hence we have $e_0 = xy - r$.

Security. The simulator $\mathcal{S}_{mask}$ sends the inputs of the parties in $\mathcal{C}$ to $\mathcal{F}_{mask}$ and receives the output $z = xy - r$. The simulator $\mathcal{S}_{mask}$ plays P_u to interact with P_v for each $v \in \mathcal{Q}$ and $u \in \mathcal{H}_v$. Concretely, $\mathcal{S}_{mask}$ simulates the messages sent by P_u as follows.

Simulator $\mathcal{S}_{mask}$

Simulating the first $n-1$ **rounds of the aggregate phase**. At each round t (from 1 to $n-1$) of the aggregate phase, $\mathcal{S}_{mask}$ first samples $(pk_t, sk_t) \leftarrow \texttt{Keygen}$ and then computes $c_t \leftarrow \texttt{Enc}(0, pk_t)$. $\mathcal{S}_{mask}$ sends c_t, pk_t to P_v. In this round, $\mathcal{S}_{mask}$ receives a public key k_t from P_v.

Simulating the last $n-1$ **rounds of the aggregate phase**. At each round t (from n to $2n-2$) of the aggregate phase, $\mathcal{S}_{mask}$ first samples $(pk_t, sk_t) \leftarrow \texttt{Keygen}$ and then computes $c_t^0 \leftarrow \texttt{Enc}(0, pk_t)$ and $c_t^1 \leftarrow \texttt{Enc}(0, pk_t)$. $\mathcal{S}_{mask}$ sends c_t^0, c_t^1, pk_t to P_v. In this round, $\mathcal{S}_{mask}$ receives a public key k_t from P_v.

Simulating the decrypt phase. At each round t (from $2n-2$ to 1) of the decrypt Stage, $\mathcal{S}_{mask}$ computes $e_t \leftarrow \texttt{Enc}(z, k_t)$ and sends e_t to P_v.

The proof that no environment can distinguish whether it is interacting with the simulator $\mathcal{S}_{mask}$ in the ideal world or with the adversary $\mathcal{A}$ in the real world with non-negligible probability is similar to the security proof of Theorem 4. $\square$

D.4 Security Proof of Theorem 13

Proof. **Correctness.** Note that the broadcaster outputs the broadcast value anyhow, so we only show that each party P_i who is not the broadcaster will output the broadcast value x at the end of π_{ggbc}. Let $w = (P_{i_0} = P_i, P_{i_1}, \ldots, P_{i_T})$ be one walk starting from P_i such that P_{i_1} is the neighbor 1 of P_i (recall that P_i outputs the decryption of the ciphertext received from its neighbor 1). From the view of this walk, our protocol proceeds as follows.

1. At the beginning of the aggregate phase, P_i samples a key pair (pk_1, sk_1) and encrypts the dummy value α with $k_1 = pk_1$ into c_1. Finally, P_i sends c_1, k_1 to P_{i_1}.
2. For $t = 1$ to $T-1$, upon receiving c_t, k_t from $P_{i_{t-1}}$, P_{i_t} samples a key pair (pk_{t+1}, sk_{t+1}) and computes $k_{t+1} = k_t \circledast pk_{t+1}$. Then, P_{i_t} computes $c_{t+1} \leftarrow \texttt{Enc}(x, k_{t+1})$ if P_{i_t} is the broadcaster and $c_{t+1} \leftarrow \texttt{AddLayer}(c_t, k_t, sk_{t+1})$ otherwise. Finally, P_{i_t} sends c_{t+1}, k_{t+1} to $P_{i_{t+1}}$.

3. P_{i_T}, upon receiving c_T, k_T, computes a ciphertext without adding a public key layer. Concretely, P_{i_T} computes $e_T \leftarrow \texttt{Enc}(x, k_T)$ if P_{i_T} is the broadcaster and $e_T \leftarrow \texttt{Rand}(c_T, k_T)$ otherwise. Finally, P_{i_T} sends e_T *back* to $P_{i_{T-1}}$.
4. For $t = T-1$ to 1, upon receiving e_{t+1} from $P_{i_{t+1}}$, P_{i_t} deletes its layer by computing $e_t \leftarrow \texttt{DelLayer}(e_{t+1}, k_{t+1}, sk_{t+1})$ and then sends e_t to $P_{i_{t-1}}$.
5. Upon receiving the ciphertext e_1 from P_{i_1}, P_i deletes its layer. In fact, P_i decrypts $e_0 = \texttt{Dec}(e_1, sk_1)$ because $k_1 = pk_1$. Finally, P_i outputs e_0.

Now we prove that if the walk w includes the broadcaster, then e_0 must be the broadcast value. Let $s \in \{0, 1, \cdots, T\}$ be the largest number such that P_{i_s} is the broadcaster, then by our protocol, c_{s+1} is an encryption of the broadcast value. On the other hand, all of $P_{i_{s+1}}, \ldots, P_{i_T}$ just add a layer to the received ciphertext without changing the underlying plaintext, hence the final ciphertext e_T is still an encryption of the broadcast value. Note that e_0 is the decryption of e_T, hence e_0 is the broadcast value.

We now prove that the probability of the event A that the walk w_i starting from P_i and neighbor 1 of P_i covers the graph for each $i \in [n]$ is overwhelming, which implies that all parties output the broadcast value with overwhelming probability. Let A_i be the event that w_i does not cover the graph (by Lemma 11, A_i happens with negligible probability), then it holds that

$$\Pr(A) = \Pr(\cap_{i\in[n]} \bar{A}_i) = 1 - \Pr(\cup_{i\in[n]} A_i) \geq 1 - \sum_{i\in[n]} \Pr(A_i) = 1 - n \cdot \mathsf{neg}(\kappa),$$

which is overwhelming because $n = \mathsf{poly}(\kappa)$.

Security. The simulator $\mathcal{S}_{ggbc}$ needs to play P_u to interact with P_v for each $v \in \mathcal{Q}$ and $u \in \mathcal{H}_v$. $\mathcal{S}_{ggbc}$ sends the inputs of the parties in $\mathcal{C}$ to $\mathcal{F}_{bc}$ and receives the broadcast value x. Recall that α is a dummy value known by all parties. $\mathcal{S}_{ggbc}$ simulates the messages sent by P_u as follows.

Simulator $\mathcal{S}_{ggbc}$

Simulating the aggregate phase. At each round t (from 1 to T) of the aggregate phase, $\mathcal{S}_{ggbc}$ first samples $(pk_t, sk_t) \leftarrow \texttt{Keygen}(1^\kappa)$ and then computes $c_t \leftarrow \texttt{Enc}(\alpha, pk_t)$. $\mathcal{S}_{ggbc}$ sends c_t and pk_t to P_v. In this round, $\mathcal{S}_{ggbc}$ receives a public key k_t from P_v.

Simulating the decrypt phase. At each round t (from T to 1) of the decrypt phase, $\mathcal{S}_{ggbc}$ computes $e_t \leftarrow \texttt{Enc}(x, k_t)$ and sends e_t to P_v.

To prove that our protocol is UC secure, we will show that no environment can distinguish whether it is interacting with the simulator $\mathcal{S}$ in the ideal world or with the adversary $\mathcal{A}$ in the real world with non-negligible probability. We define the following hybrids.

Hybrid 0. $\mathcal{S}_0$ acts as in the real execution.
Hybrid 1. $\mathcal{S}_1$ acts as $\mathcal{S}_0$ except that during the aggregate phase, the actual (layered) keys P_u sent P_v are replaced with freshly generated keys.
Hybrid 2. $\mathcal{S}_2$ acts the same as $\mathcal{S}_1$ except that during the aggregate phase, each ciphertext P_u sent P_v is an encryption of 0 instead of the actual value under the same public key.
Hybrid 3. $\mathcal{S}_3$ acts the same as $\mathcal{S}_2$ except that during the decrypt phase, each ciphertext P_u sent P_v is a fresh encryption under the same key (P_v sent this key to P_u during the aggregate phase) instead of the actual unlayered encryption.
Hybrid 4. $\mathcal{S}_4$ acts the same as $\mathcal{S}_3$ except that during the decrypt phase, the ciphertexts P_u sent P_v are replaced with encryptions of x instead of the actual values. Note that $\mathcal{S}_4$ acts exactly as $\mathcal{S}_{ggbc}$.

The proof that Hybrid 0, Hybrid 1, Hybrid 2 and Hybrid 3 are indistinguishable is the same as in the security proof of Theorem 4. Now we only show that no environment can distinguish Hybrid 3 and Hybrid 4.

Hybrid 3 and Hybrid 4 are indistinguishable. The two hybrids differ only on condition that the ciphertexts P_u sent P_v are not encryptions of x, i.e., there exists one walk which does not cover the graph. We want to argue that the probability of such an event is negligible, which implies that Hybrid

3 and Hybrid 4 are indistinguishable. Note that there are total $a \leq n(n-1)$ walks $w_1, \ldots, w_a$. Let A_i be the event that w_i does not cover the graph (by Lemma 11, A_i happens with negligible probability), then the probability p that there exists some walk which does not cover the graph satisfies that

$$p = \Pr(\cup_{i \in [a]} A_i) \leq \sum_{i \in [a]} \Pr(A_i) = a \cdot \mathsf{neg}(\kappa) \leq n(n-1) \cdot \mathsf{neg}(\kappa),$$

which is negligible because $n(n-1) = \mathsf{poly}(\kappa)$. $\square$